RWTH edition

RWTH AACHEN UNIVERSITY

Wolfgang Marquardt · Jan Morbach
Andreas Wiesner · Aidong Yang

OntoCAPE

A Re-Usable Ontology for Chemical Process Engineering

Springer

RWTHedition

RWTH Aachen

Wolfgang Marquardt · Jan Morbach ·
Andreas Wiesner · Aidong Yang

OntoCAPE

A Re-Usable Ontology
for Chemical Process Engineering

Springer

Prof. Dr. Wolfgang Marquardt
RWTH Aachen University
AVT-Process Systems Engineering
52056 Aachen
Germany
wolfgang.marquardt@avt.rwth-aachen.de

Dipl. Ing. Jan Morbach
RWTH Aachen University
AVT-Process Systems Engineering
52056 Aachen
Germany
jan.morbach@web.de

Dipl. Ing. Andreas Wiesner
RWTH Aachen University
AVT-Process Systems Engineering
52056 Aachen
Germany
andreas.wiesner@avt.rwth-aachen.de

Dr. Aidong Yang
University of Surrey
Fac. Engineering & Physical Sciences
Guildford
United Kingdom GU2 7XH
a.yang@surrey.ac.uk

ISSN 1865-0899 e-ISSN 1865-0902
ISBN 978-3-642-04654-4 e-ISBN 978-3-642-04655-1
DOI 10.1007/978-3-642-04655-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2009939134

Cover design: deblik, Berlin, Germany

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Motivation for this Book

Ontologies have received increasing attention over the last two decades. Their roots can be traced back to the ancient philosophers, who were interested in a conceptualization of the world. In the more recent past, ontologies and ontological engineering have evolved in computer science, building on various roots such as logics, knowledge representation, information modeling and management, and (knowledge-based) information systems. Most recently, largely driven by the next generation internet, the so-called Semantic Web, ontological software engineering has developed into a scientific field of its own, which puts particular emphasis on the theoretical foundations of representation and reasoning, and on the methods and tools required for building ontology-based software applications in diverse domains. Though this field is largely dominated by computer science, close relationships have been established with its diverse areas of application, where researchers are interested in exploiting the results of ontological software engineering, particularly to build large knowledge-intensive applications at high productivity and low maintenance effort.

Consequently, a large number of scientific papers and monographs have been published in the very recent past dealing with the theory and practice of ontological software engineering. So far, the majority of those books are dedicated to the theoretical foundations of ontologies, including philosophical treatises and their relationships to established methods in information systems and ontological software engineering. Only few of these contributions deal with the topic of a concrete, formal ontology, which targets a particular application domain, and even less also include a thorough and comprehensive description of ontology design, usage, and maintenance.

The majority of the existing domain- and application-centered publications address problems in the life sciences, such as diagnostic systems in medicine or knowledge management systems in molecular and cell biology. Only little activity can be observed in the engineering sciences. In particular, very few ontologies have been elaborated for the domain of chemical engineering. And those few notable engineering ontologies available today are often found to be inappropriate for use in a context different from the one they have actually been developed for. Hence, there is hardly any ontology in the engineering sciences which can be broadly used and which is actually applicable.

The objective of this book is to contribute to closing the gap between theory and practice in ontological software engineering. In the first place, it provides a fully elaborated formal ontology for the domain of chemical engineering. This ontology provides a reasonable conceptualization of the chemical engineering domain, which is a prerequisite for establishing a shared understanding of concepts and terms in a certain scientific field and for fostering the communication in a typical cross-disciplinary engineering design team. The ontology is also supposed to support and to simplify the development of future software applications in computer-aided process engineering (CAPE), a sub-discipline of chemical engineering. The intention associated with the development of this ontology has been expressed in its name OntoCAPE, linking ontologies with computer-aided process engineering.

Besides this very concrete engineering objective, the development of OntoCAPE also aimed at the elucidation and the benchmarking of architectural principles for the design of large-scale ontologies, which can be reused in the same domain for different applications or even across related (engineering) domains. The strive for said architectural principles is closely related to the derivation of a set of guidelines, which assist the ontology engineer in capturing, structuring, formalizing, and documenting the knowledge in a complex engineering domain. Hence, we hope that the development of OntoCAPE – the process as well as the resulting architecture and design principles – can serve as a role model or at least as a best-practice example to guide related efforts in other (engineering) domains.

OntoCAPE is the result of close to two decades of research in knowledge representation and information modeling applied to chemical engineering, which was performed by the research group lead by Wolfgang Marquardt. In the first decade, the primary research objective was to find a conceptualization of the chemical engineering domain that can support the mathematical modeling of chemical process systems. In particular, the construction of phenomena-based computer-aided modeling tools has been envisioned in order to reach beyond the established equation-oriented and block-oriented modeling tools. Later, the research activities have been extended to cover the complete lifecycle of chemical process and plant design with its enormous variety of tasks and model-based solution techniques.

The development of OntoCAPE, has been conducted as part of two large, interdisciplinary research projects, the IMPROVE[1] project (Marquardt and Nagl 2004; Nagl and Marquardt 2008) funded by DFG, the German Research Foundation, as part of the collaborative research center CRC 476, and the COGents project (Braunschweig et al. 2002; Yang et al. 2008), funded by the European Commission in the 5th Framework Program as part of the Information Society Technologies track. The former has been concerned with the development of novel methods and software tools for the support of collaborative design processes in chemical engineering, while the latter has explored a future approach to numerical simulation enabling the assembly of large simulation models from model components provided by libraries distributed on the internet.

[1] IMPROVE has been continued by the Transfer Center 61 (Nagl and Marquardt 2008), the goal of which has been the transfer of selected research results obtained in IMPROVE into industrial practice.

The conceptual basis for OntoCAPE has been established by the information model CLiP (Bayer 2003). Developed in IMPROVE, CLiP itself has been built on previous work carried out prior to IMPROVE and resulting in the knowledge-base VeDa (Bogusch 2001; Baumeister 2000). Version 1.0 of OntoCAPE (Yang and Marquardt 2004; Yang et al. 2008) was developed in the COGents project using CLiP as a starting point. At that time, the development had still a quite narrow focus resulting in an overall structure and content which is very different from the current version of OntoCAPE presented in this book. After completion of the COGents project, the further development and extension of OntoCAPE was taken over by IMPROVE. In 2007, version 2.0 of OntoCAPE was released. Since then, OntoCAPE has been continuously improved to reach the status described in this book. A more extensive overview on the history of OntoCAPE can be found in Chapter 11.1 of this book.

Target Audience

The book is targeted at three major groups of readers. Firstly, the book addresses potential users of the ontology, i.e., practitioners in chemical engineering who are interested in the development and employment of intelligent software applications. They can derive a customized knowledge-base from OntoCAPE, which contains the knowledge they consider to be relevant for an intended application. Such a customization may include specializations of OntoCAPE to incorporate concepts on a more fine-grained, application-specific layer as well as extensions relating to other process engineering tasks which have not yet been addressed by the current version of OntoCAPE.

Secondly, OntoCAPE may serve as an example for knowledge engineers who are willing to develop an ontology for a related, but different (engineering) domain. Such an adaptation of (i) the architecture and of (ii) part of the concepts covered by OntoCAPE requires that the knowledge can be represented according to the principles of general systems theory (Bunge 1979) and systems engineering (Thomé 1993). Candidate domains for an adaptation of OntoCAPE are all engineering domains such as energy, automotive, aerospace, civil, or production engineering. Technical systems in various manifestations are at the heart of these engineering disciplines. Consequently, the more abstract concepts of the ontology should be transferable without the need for modification. In addition, applications in these and other domains are expected to explicitly benefit from the general design patterns employed in OntoCAPE, since they have been explicitly incorporated to support the applications developers in refining, extending, or changing the ontology to their particular needs.

Finally, this book is supposed to be of interest to experts in ontology engineering. OntoCAPE is expected to serve as a good example to illustrate how the plethora of design principles reported largely in the computer science literature can be put in practice to organize a complex ontology in an engineering domain. This book summarizes our experience gained during two decades of research in the design of engineering ontologies. We hope that it can serve as a guideline for ontology engineering experts to illustrate how the suggested design principles support and facilitate the development of a complex ontology.

Accessing and Using OntoCAPE

OntoCAPE is publicly accessible at http://www.avt.rwth-aachen.de/Ontocape, where it is distributed under the terms of the GNU General Public License. It is available as an informal and as a formal specification. Both specifications of OntoCAPE can be accessed via this website free of charge.
For the formal specification of OntoCAPE, the Ontology Web Language OWL has been chosen (OWL-DL, in particular, cf. Bechhofer et al. 2004). The model development was done by means of the ontology editor Protégé (Stanford 2008). For verification, the reasoner RacerPro (Racer Systems 2007) has been used. The current release of OntoCAPE consists of 62 OWL files, each of which includes one module of the ontology.
The informal specification currently takes the form of six technical reports, which jointly comprise about 500 pages. It serves the double function of (i) a users manual and (ii) a reference guide. These reports present the organization and structure of OntoCAPE; the conceptualizations of various topic areas are described in great detail. Special emphasis was placed on making OntoCAPE as user-friendly as possible by (i) providing a detailed description of concepts, relations and instances and an explanation of the corresponding interrelations in descriptive UML-like diagrams, by (ii) defining the terms of proper usage of the ontology, and by (iii) highlighting the important design decision and principles leading to the organization and structure at hand. This informal specification complements the contents of this book. While the book focuses on the design principles and architecture of OntoCAPE, concept definitions comprising class, relations and individuals definitions are only sketched and detailed in few instances for illustration purposes. In contrast, the informal specification provides all the concept definitions in a comprehensive manner for reference.
OntoCAPE complies with the two principal types of usage for an ontology that are typically mentioned in the computer science literature: Firstly, an ontology may serve as a library of knowledge components to efficiently build intelligent systems. To this aim, the generic ontology is to be transformed (i.e., extended and customized) into a knowledge-base according to the requirements of the respective application. The second type of usage refers to a shared vocabulary for communication between interacting human and/or software agents. According to their respective functions, the communicating agents may have different knowledge-bases, but all the knowledge-bases must be consistent with the ontology (Gruber

1995). Both types of usage require a consensual knowledge representation that is reusable in different application contexts.

About the Authors

Wolfgang Marquardt, born in 1956 in Germany, studied Chemical Engineering at Stuttgart University, Germany, where he graduated in 1982. Subsequently, he joined the Institute of System Dynamics and Control of Stuttgart University where he completed his PhD dissertation in 1988. From 1989 to 1990, Wolfgang did postdoctoral research at the University of Wisconsin, Madison, USA. After returning to Stuttgart University, he completed his habilitation in the field of dynamics and operations of chemical process systems.
Wolfgang was appointed a Professor for Process Systems Engineering at RWTH Aachen University, Germany, in 1993. His research interest cover a number of areas in process systems engineering, including modeling methodologies, process synthesis, control and operations, numerical methods as well as information technologies for the support of chemical engineering design processes.
Wolfgang has been honored with a number of awards and appointments, including the Award of the Alumni Foundation of Stuttgart University in 1988, the Arnold-Eucken Award of VDI-GVC, the German national association of chemical engineers, in 1990, the appointment as a member of the North Rhine-Westphalian Academy of the Sciences in 1998, and of acatech, the German National Academy of Engineering Sciences, in 2001, the Leibniz-Preis of the German Research Foundation in 2001, the promotion to IFAC Fellow in 2007, and the distinction as the Danckwerts Lecturer of AIChE and EFCE, the US and European associations of chemical engineering, in 2008.

Jan Morbach, born in 1976 in Germany, studied Mechanical Engineering majoring Chemical Engineering at RWTH Aachen University, Germany, and Carnegie Mellon University, Pittsburgh, USA. He graduated with a Dipl.-Ing. degree from RWTH Aachen University in 2002. He worked at AVT – Process Systems Engineering, RWTH Aachen University from 2002 to 2007 as a research assistant. At AVT, his areas of interest included data integration in chemical engineering, ontology engineering, and computer-aided process design. Jan completed his PhD dissertation in 2008.

Since 2007, he works for Bayer Technology Services at Leverkusen, Germany, in the field of chemical process design.

Andreas Wiesner, born in 1981 in Germany, studied Mechanical Engineering majoring Chemical Engineering at RWTH Aachen University, Germany, and at Imperial College, London, UK. He received his Dipl.-Ing. degree in 2006 from RWTH Aachen University. Since 2006, he works as a research assistant at AVT – Process Systems Engineering, RWTH Aachen University. His areas of interest include data integration and management in chemical engineering, ontological engineering, and computer-aided process design.

Aidong Yang, born in 1971 in China, received a B. Eng. degree from Hebei University of Technology in 1992 and a Ph.D. degree from Dalian University of Technology in 1997, both in chemical engineering. During his post-doctoral career, he worked for several institutions in the area of process systems engineering, with a focus on information modeling and mathematical modeling of chemical process systems. In particular, he was a research fellow at AVT – Process Systems Engineering, RWTH Aachen University, from 1999 to 2004, where his research was mainly devoted to the development and applications of ontologies in process engineering.
Aidong is currently a lecturer in Process Systems Engineering at the Faculty of Engineering and Physical Sciences of the University of Surrey, UK. His research interests include the mathematical modeling and optimization of process systems, the application of knowledge engineering methods in process and product development and in manufacturing, and the design of engineering software systems.

Acknowledgements

This research would not have been possible with the substantial financial support from various funding organizations. NATO funded a post-doctoral fellowship of Wolfgang Marquardt at the University of Wisconsin, where he started to learn about knowledge representation and information modeling. The very first attempts to create the knowledge-base VEDA date back to this period. DFG – the German Research Foundation – as well as Stuttgart University and RWTH Aachen University – funded a number of projects on the foundations of computer-aided process

modeling under different funding schemes. This research formed one of the pillars of our research on ontological engineering. A second pillar are the CAPE-OPEN and Global CAPE-OPEN projects funded by the EU under the Brite-EuRam action line. These industry-driven projects on the standardization of interfaces in and between process engineering software systems provided us with a great deal of insight in many practical issues of software engineering, information modeling, and data integration in the context of process simulation and design in the process industries.

Most of the research described in this book has been performed in the context of the IMPROVE project, funded by DFG as CRC 476 and TC 61, and the COGents project, funded by the EU under the IST program in the fifth Framework Program. The generous financial support through these projects is gratefully acknowledged.

We are also indebted to a large number of individuals who directly or indirectly contributed to the results presented in this book. We enjoyed the very close and lasting collaboration with a number of research groups in computer science in areas related to ontological engineering. Most notably, we interacted with Franz Baader and his former PhD students Ulrike Sattler and Ralph Molitor, with Bertrand Braunschweig, with Matthias Jarke and his former PhD students Sebastian C. Brandt, Michalis Miatidis, Klaus Pohl and Klaus Weidenhaupt as well as with Manfred Nagl and his former PhD students Simon Becker, Markus Heller and Bernhard Westfechtel. This interdisciplinary interaction, and the sharing of the various perspectives on knowledge representation and information modeling, integration, and management contributed significantly to shape our understanding of the relevant issues. The reconciliation of the diverse schools of thought in computer science has been an important stepping stone towards OntoCAPE, its architectural principles and the design guidelines as presented in this book. This book would definitely not have been possible without the fruitful (though time-consuming) discussions with all these colleagues.

OntoCAPE 1.0 has been jointly developed by a number of colleagues under the project leadership of Aidong Yang during the CoGents projects. Contributions from Bertrand Braunschweig, Eric S. Fraga, Didier Paen, Daniel Pinol, Pascal Roux, Manel Serra and Ian Stalker are particularly acknowledged. Also we would like to thank Gabriela Henning and Horacio Leone for the good cooperation.

The OntoCAPE documentation had been sent to a number of computer scientists and experts in computer-aided process engineering and related areas for their comments. The efforts made by these external reviewers are highly appreciated. We are particularly grateful for the feedback we have received on OntoCAPE 1.0 from Rafael Batres (Tokyo Institute of Technology), Patrick Linke (University of Quatar), Michael Stollberg (University of Innsbruck), and Matthew West (Shell Information Technology International).

We want to particularly acknowledge the work of our former colleagues at AVT – Process Systems Engineering who participated in the VeDa project and built the sound foundation on which OntoCAPE is based, namely Markus Baumeister, Ralf Bogusch, Claudia Krobb, Bernd Lohmann, Daniel Souza, and Lars von Wedel.

Furthermore we like to thank all the members of the IMPROVE project who contributed directly or indirectly to OntoCAPE. We want to mention particularly Birgit Bayer, Markus Eggersmann, Ri Hai, Ralph Schneider, Manfred Theißen, and Lars von Wedel, whose contributions have strongly influenced the development of OntoCAPE. A great deal of their ideas and passion is reflected by the content of this book.
Furthermore, the authors would like to thank a number of master students who contributed to this book. We want especially point out Raoul Frings. He helped to implement major parts of OntoCAPE in its current version. Raoul Frings, Yassin Mbarek and Sandra Kurpiers spent a lot of effort and time with the many revisions of this text to get the layout of this book in proper shape.
Last but not least we want to thank our families and fiancés for their patience and support when we spent many week-ends and long evenings in preparing and finalizing the manuscript of this book.
We made a major effort to deliver a flawless manuscript. However, if you find any errors, or if you have any other suggestions for improvement, please do not hesitate to let us know (email: Wolfgang.Marquardt@avt.rwth-aachen.de).

July 2009

Wolfgang Marquardt
Jan Morbach
Andreas Wiesner
Aidong Yang

Contents

1 Introduction

1.1 The Need of Knowledge-Based Systems

Data, information, knowledge, understanding, and wisdom (Ackoff 1989) drive the society, economy, and science. Data refers to a collection of symbols without any meaning beyond its existence. Information refers to a set of data which have been given a meaning by formulating relations between the data elements in a given context. Knowledge constitutes a collection of information with the intention of a certain kind of use. Understanding (or reasoning) refers to an analytic and cognitive process, which takes some knowledge as its input to infer new knowledge as its output by some kind of interpolation. In contrast to understanding, wisdom is an extrapolative and non-deterministic process to provide (i) understanding where there was no understanding before and (ii) a kind of knowledge which cannot be inferred solely by analytical means from available knowledge. Wisdom not only calls upon all previous levels of consciousness, but also extends to human programming such as moral and ethical codes. Wisdom is a human feature and therefore very different from data, information, and knowledge, which can be stored, processed, and even extended by computers by algorithmic reasoning.

Knowledge comes in two forms, either tacit or explicit (Nonaka and Takeuchi 1995). Tacit knowledge is implicit – it is difficult to grasp for the individual holding it therefore hard to communicate. To be of value to other individuals, knowledge has to be made explicit by some kind of articulation, codification, and storage by means of some media to facilitate communication to others. Technical reports, patents, journal articles, monographs, textbooks, or encyclopedias are classical media where the information and knowledge contained is codified by means of natural language. Despite its explicit representation, knowledge is often hard to access and difficult to process, because natural language representations often lack precision and coherence resulting in texts of ambiguous meaning.

This deficiency is not necessarily a consequence of the imperfect presentation skills of the author, but is rather due to the lack of a common vocabulary and common understanding, which is a prerequisite for a shared memory and shared meaning across different domains of discourse (Konda et al. 1992). Typically, this unavoidable shortcoming of natural language knowledge representations is remedied by the intellectual skills, i.e., the wisdom of the reader, who is often well-trained in the domain providing the context of the text.

The codification of tacit knowledge has to go beyond the use of natural language to facilitate sharing, use, and reuse of information and knowledge. The scientific and engineering disciplines have come up with very specific ways to address this representation problem well before computers have been introduced and used for information and knowledge management. For reasons of conciseness, we will focus here and in the remainder of this book only on the engineering perspective.

Engineering knowledge – that is general domain knowledge of the different engineering disciplines as well as specific knowledge evolving, for example, during an engineering design project – is not only represented and stored as text documents in natural language. Rather, more structured and formalized means of representation are being used, such as sets of linked text documents with a prescribed structure, structured worksheets, forms and tables, mathematical models, graphical sketches, technical drawings, and the like. These auxiliary, and at least to some extent formalized, representational schemes are supposed to enhance the expressiveness of natural language, thereby facilitating access and use of information and knowledge. However, the exchange of information and knowledge – be it within a project team or between different organizations – is still hampered because there is no general agreement on the precise syntax and semantics of these representational formalisms.

The problem has not only a logical, but also a technological dimension: That is, the syntactic and semantic heterogeneity is aggravated by the diversity of electronic means and formats for storage, communication, and processing of information and knowledge. For example, during the individual stages of a process and plant design project, information is created and manipulated by diverse software tools and stored in heterogeneous proprietary formats, such as electronic documents, data bases, Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) systems, simulation files, or asset management tools. The lack of integration between these software tools and their associated data stores unavoidably creates a significant overhead for the project engineers, since much time has to be spent on re-entering of data, interpreting and understanding the data, manually reconciling overlapping data sets, and searching for data. NIST, the National Institute of Standards and Technology in the U.S., has recently analyzed the efficiency losses resulting from inadequate interoperability among computer-aided design, engineering, and software systems (Gallaher et al. 2004). According to this study, insufficient interoperability causes costs of 15.8 billion dollars in the US capital facilities industries, compared to a hypothetical scenario where the exchange of data and the access to information are not restricted by technical or organizational boundaries.

In order to improve this situation, we need to introduce new methods and tools that enable computer-based information and knowledge management in interdisciplinary and cross-institutional engineering projects – methods and tools, which help to cope with the diverse and therefore heterogeneous application software infrastructure. As a *prerequisite* for the solution of these extremely demanding interoperability problems, one has to define a common vocabulary in order to establish a shared understanding of concepts and terms. Such a shared understanding constitutes the basis of the *shared memory of an organization*[2], regardless of whether the organization is a project team, an enterprise, or a collection of cooperating institutions. Regardless of its technical realization, a shared memory can cover data, information, and – at least to some extent – knowledge implemented by algorithmic reasoning, but not wisdom in the sense of the definitions introduced above. Also, the management of data, information, and knowledge relies on a carefully designed and implemented shared memory. Its realization constitutes a true scientific and technological challenge, given the complexity of the engineering domains and industrial design projects.

There is a much simpler, but still similar problem, which has been successfully addressed in the past decades – namely, the design and implementation of software systems for the mathematical modeling and simulation of technical systems. The approach to this particular problem might serve as a useful role model for the development of future knowledge-based systems supporting engineering work processes. We will briefly draw on this analogy next:

Modeling and simulation systems[3] have originated from an academic environment. A broad acceptance and routine use in the chemical and process industries can be traced back to the 1980s, when commercial systems became widely available. The success of these systems relied (and still relies) on three distinct modules: editors with tailored representational schemes for the formulation of mathematical models; libraries providing reusable mathematical models for standard devices; and, finally, numerical solution techniques for the simulation-based evaluation of the physicochemical knowledge encoded in the models.

We conjecture that for the successful implementation of knowledge-based tools in engineering projects, three comparable modules are required: Knowledge editors providing a semantically rich formalism for knowledge representation; knowledge libraries with self-contained and easily accessible chunks of reusable knowledge; and, finally, efficient reasoning capabilities to interpret the knowledge encoded in the system and to derive new knowledge from it. In fact, mathematical modeling and simulation are themselves a kind of knowledge-based application, working with chunks of reusable knowledge formalized in a special way, namely by means of mathematical equations. The seamless integration of mathematical modeling in the traditional sense with information modeling and knowledge representation to support model-based engineering work processes is widely considered to be an emerging trend (Marquardt et al. 2000; Subrahmanian and Rachuri 2008; Venkatasubramanian 2009).

[2] Such interoperability problems in collaborative and distributed chemical engineering are covered in depth in the recent monograph edited by Nagl and Marquardt (2008). Their solution relies on the concept of a shared memory of an organization, but has to reach far beyond.

[3] There is a vast literature on this subject, often with emphasis on a certain engineering domain. We exemplarily cite the book of Braunschweig and Gani (2002) for a chemical engineering perspective.

As a result of the research and development related to the Semantic Web – the next-generation internet – semantic technologies are ready for application in other scientific disciplines. They are based on *ontologies* and include exactly the three enabling building blocks, namely knowledge editors, knowledge representation (or modeling) languages, and powerful reasoning algorithms.

1.2 The Role of Ontologies

In the context of this book, ontologies are primarily seen as a means to efficiently build the knowledge-based software necessary to effectively support engineering work processes. Such software – also referred to as 'intelligent systems', 'artificial intelligence', 'AI systems', or 'expert systems' – comprises two basic software components: the *knowledge base*, which contains generic domain knowledge as well as concrete facts about the case under consideration, and the *inference engine* (also known as *reasoner*), which processes the knowledge and facts stored in the knowledge base and autonomously inferences a solution for the case at hand.

Traditionally, intelligent systems were built from scratch. For large systems, however, this proceeding turned out to be too costly and time consuming. Particularly, the construction of the knowledge bases proved to be the main cost-driver that hindered the further development of intelligent systems in the late 1980s. Neches et al. (1991) diagnosed: "knowledge base construction remains one of the major costs in building an AI system […] As a result, most systems remain small to medium in size. […] The cost […] will become prohibitive as we attempt to build larger and larger systems."

To overcome this economic barrier, Neches et al. (1991) proposed a new approach for the building of intelligent systems: "Building knowledge-based systems today usually entails constructing new knowledge bases from scratch. It could be instead done by assembling reusable components. System developers would then only need to worry about creating the specialized knowledge […] new to the specific task of the system […] In this way, declarative knowledge […] and reasoning services would all be shared among systems."

Besides the obvious *economic benefits* that can be achieved by reusing existing knowledge components, the strategy has other considerable advantages:

– First to mention is the *reduced error rate* of the software: The robustness of a software system increases to the extent to which well-tested parts can be reused (Neches et al. 1991). Plus, due to the continuous revision of the knowledge components, the number of remaining errors will decrease with each reuse cycle.
– A further advantage results from a mandatory change of system architecture required by the new approach: Traditionally, the knowledge representation was heavily intertwined with the reasoning services and the program code in order to optimize the performance of the overall system. As a re-

sult, the knowledge was only accessible to developers with programming experience; domain experts (i.e., the actual knowledge holders) had to get acquainted with the program code first before being able to enter knowledge into the system or to maintain and customize the knowledge base to their particular needs. In practice, this often proved too great an obstacle for the users to overcome. The new approach, by contrast, enforces a strict separation of knowledge base, inference engine, and application-specific program logic. This novel software architecture enables a domain expert to focus on the representation of the knowledge and shields him or her from the implementation details. Therefore, the domain expert is likely to create a knowledge base of *improved quality*. At the same time, the *maintainability* of the entire system is *enhanced*, since reasoner, program code, and knowledge base can be maintained independently by software engineers, application programmers, and domain experts, respectively.

Within the suggested approach, *ontologies* have the function of providing a consensual knowledge representation, which can be reused and shared across software systems and by different groups of users. *Domain ontologies*, in particular, aim at capturing the knowledge of an entire application domain, such as physics, chemistry, or engineering. Note that, in order to be widely applicable, the knowledge represented in an ontology must be generic; that is, the ontology is expected to provide "a conceptual foundation for a range of anticipated tasks", but not to "include vocabulary sufficient to express all the knowledge relevant to those tasks" (Gruber 1995). Thus, to convert an ontology into a knowledge base for a particular application, the knowledge must be specialized and customized.

1.3 The Reusability-Usability Trade-off Problem

Principally, any ontology has to meet two major goals: to be *usable* and to be *reusable*.

– According to the IEEE Standard Glossary of Software Engineering Terminology, *reusability* is defined as "the degree to which a software module or other work product can be used in more than one computing program or software system" (IEEE 1990). *Ontology reusability*, in particular, can be defined as "the adaptation capability of an ontology to arbitrary application contexts" (Pâslaru-Bontaş 2007), including those contexts "that were not envisioned at the time of the creation of the ontology" (Russ et al. 1999). Note that it is neither feasible nor desirable to design an ontology that is equally appropriate for all application contexts (Borst 1997); rather, the goal of reusability is to come up with an ontology that can be adapted to a preferably large number of applications.

- *Usability*, on the other hand, denotes the degree to which the software component is useful for a specific task or application. The term also has the connotation of "ease of use", pertaining to the effort required by a user to utilize a given (software) system. By definition, an ontology is never ready for use, but must always be adapted and refined to a knowledge base for the envisioned application. Therefore, the goal of *ontology usability* can be phrased as minimizing "the effort required to customize the ontology so that it can be used by humans or machines in a given application context" (Pâslaru-Bontaş 2007).

A subtle but important difference between ontology usability and reusability is pointed out by Jarrar and Meersmann (2002):

> "Increasing the reusability of knowledge implies the maximization of using this knowledge among several kinds of (autonomously specified) tasks, while increasing ontology usability could mean just maximizing the number of different applications using an ontology for the same kind of task".

Consequently, it is difficult to simultaneously achieve high degrees of usability and reusability: Specializing in one kind of task makes the ontology more useable for this particular task, but it also decreases the likelihood of its reusability; a highly abstract ontology, on the other hand, may be applicable to a variety of different tasks, but it is unlikely to prove very useful for any of these without extensive modification and detailing. This challenge is known as the *reusability-usability trade-off problem* (Klinker et al. 1991) in the literature.
This trade-off problem has to be one of the drivers for research on ontologies, not only for academic, but also for very practical reasons.

- This problem is academically challenging and rewarding. It constitutes an exciting research problem at the interface between computer science and its applications in science and engineering. Its solution is of great significance to both the theory and practice of ontological engineering.
- The development and maintenance of any major IT system requires a significant effort. The software industry has established development processes based on proven technologies to reduce cost to the extent possible. The introduction of a new paradigm and associated technologies not only requires some reference systems, which demonstrate the improved capabilities from a technological or even from an end-user's perspective. Rather, the economical advantage over established software technologies has to be clearly demonstrated in order to motivate a software company to take the significant risk of integrating the principles and technologies of ontological engineering into their software engineering processes.

Based on this assessment, an appropriate solution to the reusability-usability trade-off problem should be considered as the major enabler for a future use of ontologies in the software industries. Therefore, this trade-off problem has shaped the major guiding principle for our research, the results of which are presented in this book.

1.4 Objective and Outline of the Book

This book presents OntoCAPE, a general-purpose ontology for applications in the domain of computer-aided process engineering (CAPE). CAPE is a sub-discipline of chemical engineering focusing on IT methods and tools to support the design, planning, construction, commissioning, and operation of chemical process systems and plants. We will discuss the architecture of OntoCAPE, thereby putting particular emphasis on the design rationale we followed. We will show how OntoCAPE reconciles the trade-off between reusability and usability, and is thus broadly applicable to a variety of chemical and process engineering tasks with only moderate customization effort.

The content of his book is organized as follows.

Chapter 2 reviews the scientific background and establishes the terminology required for discussing ontologies, thus providing the basis for the subsequent chapters. It starts off by contrasting the similar but different perceptions of ‘ontology’ in the areas of philosophy and computer science. Next, the specification of ontologies through informal and formal languages is discussed; the latter option is further elaborated by describing the modeling capabilities of formal ontology languages. Having established these basic facts, it is argued that an ontology must be both formally and informally specified in order to be of practical use. The modeling language OWL is briefly introduced for the sake of completeness. Moreover, it is clarified what differentiates a “true” ontology (i.e., a reusable knowledge representation, as defined in Sect. 1.2) from so-called pseudo-ontologies and lightweight ontologies. The chapter closes with a classification of ontology types according to their respective functions.

Chapter 3 gives an overview on the scope and content of the OntoCAPE ontology. Initially, a short overview is given on its three structural elements – layers , modules and partial models – by which the ontology is organized. Furthermore, the representation and dissemination of OntoCAPE are presented. Finally, the scope and content of the individual parts that constitute OntoCAPE are briefly summarized.

Chapter 4 introduces the Meta Layer, which is located on top of the OntoCAPE ontology. The Meta Layer explicitly represents the underlying design principles of OntoCAPE and introduces common standards for the design and organization of the ontology. In particular, domain-independent root concepts and a theory of mereotopology are introduced on the Meta Layer.

Chapter 5 introduces a number of key concepts, which establish the principles of general systems theory and systems engineering, according to which the ontology is organized. Important systems-theoretical and physicochemical primitives complement the mereotopological concepts adopted from the Meta Layer. It also establishes the means to model a system from a particular viewpoint. These viewpoints are used to partition the representation of complex systems into manageable parts and to emphasize a certain perspective the system is viewed from. Furthermore, a representation of vectors and higher-order tensors is suggested. We introduce a concept for coordinate system, which serves as a frame of reference for the observation of system properties. Finally, we establish the principles of network theory to lay a solid foundation for a structured representation of any kind of complex system showing a network character.

Chapter 6 holds fundamental notions such as space, time, physical dimensions, SI-units, mathematical relations, etc., which do not directly belong to the CAPE domain but are required for the definition of or as supplements to the domain concepts. Since OntoCAPE is not supposed to conceptualize domains beyond the scope of CAPE, this partial model is only rudimentarily elaborated.

Chapter 7 collects all the concepts which are required to provide an abstract description of materials processed in a chemical plant. This partial model comprises the essential concepts for the description of pure chemical substances and mixtures thereof at the macroscopic and atomic scales. Mechanisms and stoichiometry of chemical reactions are presented next. Finally, the principles and concepts for a rigorous description of the thermodynamic behavior of materials in a certain physical context are described.

Chapter 8 presents all those concepts which are directly related to the processing of materials and to plant operations. This part of the ontology is of particular interest for chemical process design. The concepts are modeled on a conceptual as well as on a more concrete, application-oriented level by adding classes and relations needed for a specific use of the ontology. This includes the extension towards two alternative classification schemata for unit operations as well as exemplary descriptions of typical process units.

Chapter 9 defines the notions required for a representation of mathematical models. It introduces the basics concepts for mathematical modeling, including model variables and equations, as well as concepts for representing the composition of a model from sub-models and their connections. Some specialized types of models are presented, including models for representing the behavior of materials and process units as well as models for the estimation of investment costs.

Chapter 10 presents the major design principles that guided the development of OntoCAPE: These principles, which subsume the plethora of recommendations stated in the literature, are *coherence*, *conciseness*, *intelligibility*, *adaptability*, *minimal ontological commitment*, and *efficiency*. The principles are defined individually, and their general implication on ontology design is critically assessed. Finally, we describe the translation of these principles into concrete design decisions to be taken during the realization of OntoCAPE.

Chapter 11 gives a review of related work. The earlier efforts in information modeling at the authors' institute are summarized first, since the results of this research have laid the foundation for the development of OntoCAPE. Next, related work of other research groups is reviewed and compared with OntoCAPE. The review is confined to ontologies which are of particular relevance in the context of this work. In particular, only those ontologies are considered, which bear close resemblance to OntoCAPE with respect to both scope and level of complexity, or which had a significant influence on the development of OntoCAPE.

Chapter 12 describes some software applications, which have been realized on the basis of OntoCAPE and thus demonstrate the ontology's potential for use and reuse. At first, two early applications in the area of mathematical modeling and simulation are presented. Next, an ontology-based knowledge management system is described, which has been based on version 2.0 of OntoCAPE. The last example refers to an ongoing project, which realizes information integration and management across the plant lifecycle in an industrial setting. The last part of the chapter gives an assessment of the improvement achieved by transitioning from OntoCAPE version 1.0 to version 2.0 by means a few quantitative measures.

Chapter 13 concludes the book with a brief summary of the major results, with an assessment of the design rationale, and with a review of the continuous improvement process chosen. Future research opportunities are identified, particularly an extension of OntoCAPE in scope to also cover work processes.

1.5 References

Ackoff RL (1989) From data to wisdom. *J. Appl. Syst. Anal.* **16**:3–9.

Borst P, Akkermans JM, Top JL (1997) Engineering ontologies. *Int. J. Hum Comput Stud.* **46**:365–406.

Braunschweig B, Gani R, eds. (2002) *Software Architecture and Tools for Computer Aided Process Engineering*. Elsevier Science B.V.

Gallaher MP, O'Connor AC, Dettbarn Jr JL, Gilday LT (2004) *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. Technical Report (NIST GCR 04-867), National Institute of Standards and Technology, Gaithersburg, Maryland.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

IEEE (1990) *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Standard 610.12-1990, Institute for Electrical and Electronics Engineering, New York.

Jarrar M, Meersmann R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine SSGRR2002*.

Klinker G, Bhola C, Dallemagne G, Marques D, McDermott J (1991) Usable and reusable programming constructs. *Knowl. Acquis.* **3** (2):117–135.

Konda S, Monarch I, Sargent P, Subrahmanian E (1992) Shared memory in design: A unifying theme for research and practice. *Res. Eng. Des.* **4**:23–42.

Marquardt W, von Wedel L, Bayer B (2000) Perspectives on lifecycle process modeling. In: Malone MF, Trainham JA, Carnahan B (eds.): *Foundations of Computer–Aided Process Design*. AIChE:192–214.

Nagl M, Marquardt W, eds. (2008) *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support.* Springer, Berlin.

Neches R, Fikes R, Finin T, Gruber T, Patil R, Senator T, Swartout WR (1991) Enabling technology for knowledge sharing. *AI Mag.* **12** (3):36–56.

Nonaka I, Takeuchi H (1995) *The Knowledge Creating Company.* Oxford University Press, New York.

Pâslaru-Bontaş E (2007) *Contextual Approach to Ontology Reuse: Methodology, Methods and Tools for the Semantic Web*. PhD Thesis, FU Berlin.

Russ T, Valente A, MacGregor R, Swartout W (1999) Practical experiences in trading off ontology usability and reusability. In: *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publication.

Subramanian E, Rachuri S (2008) Guest editorial: special issue on “engineering informatics”. *Trans. ASME, J. Computing and Information Science in Engineering*. **8**:010301-4.

Venkatasubramanian V (2009) Drowning in data: informatics and modeling challenges in a data-rich and networked world. *AIChE J.* **55**:2–8.

2 Scientific Background

2.1 Ontology in Philosophy

Originally, Ontology is a philosophical discipline concerned with the question of what exists and what is the essence of things. The term ‘Ontology’ stems from ancient Greek and can be translated as ‘theory of existence’[4]. The discipline of Ontology was founded by Greek philosophers, such as Parmenides of Elea and Aristotle, during the 4th Century BC. Ontology has been a topic of active research throughout the Middle Ages and Modern Age until today, with contributions from such renowned philosophers as Kant and Wittgenstein. Nowadays, Ontology constitutes an important area of contemporary philosophy, covering large research projects and reaching out to such different areas as artificial intelligence, database theory, and natural language processing.
According to the Stanford Encyclopedia of Philosophy (cf. Hofweber 2005), the discipline of modern Ontology comprises four different aspects, denoted by (O1) to (O4):

(O1) The study of what there is, what exists.

(O2) The study of the most general features and relations of the entities which do exist.

A prerequisite for (O1) is to clarify in which things one must (initially) believe before one may reason about the existence of other things. Therefore, Ontology also includes

(O3) the study of ontological commitment, i.e., to become aware of what one is committed to.

Generally, an ontological commitment to the existence of an entity (A) becomes necessary in order to make a statement about the existence of another entity (B). In other words: the existence of entity A is presupposed or implied by asserting the existence of entity B. A typical commitment would be the choice of a modeling language (cf. Sect. 2.3); that is, one commits to abstract entities, such as classes or relations, or to particular theories, such as second order logic
Finally, the field of Ontology incorporates

(O4) the study of Meta-Ontology, i.e., saying what task it is that the discipline of Ontology should aim to accomplish, if any, how the questions it aims

[4] ὄ (Ontos), the genitive of ὄ (On), means ‘*of being*‘; the suffix - (-logia) denotes a *science*, *study*, or *theory*. So originally, the word signifies ‘*theory of being*’.

to answer should be understood, and with what methodology they can be answered.

In the following, a particular ontological theory is referred to as an *ontology*[5]. The individual ontologies considered in the context of this work will mainly focus on the aspects (O1) and (O2).
An ontology can be specified on different levels of formality. According to Uschold and Grüninger (1996) and Hofweber (2005), an ontology is designated as

– informal if expressed in natural language;
– *semi-informal* if expressed in a restricted and structured form of natural language;
– *semi-formal* if expressed in an artificial and formally defined language; and
– *(rigorously) formal* if the ontology contains precise mathematical definitions of certain entities in terms of their properties and their relations to other entities. Such definitions are usually given in form of axioms formulated in a logic-based language. This allows proving certain properties about an ontology, such as its consistency[6].

Formal ontologies have proven to be applicable in numerous areas; a particularly popular field of application is based on utilizing a formal ontology as a framework for *information representation.*. Information represented in such a framework is easily accessible to automated information processing. For that reason, ontologies have become a subject of intensive research in the area of computer science.

2.2 Ontology in Computer Science

Over the last decades, the term 'ontology' has been adopted by computer scientists, firstly in the field of artificial intelligence (AI) and more recently in other areas, as well. Within this community, the term is used in a more narrow sense than in the context of philosophy, denoting a *formal ontology for information representation*[7] (see above). Viewed from the perspective of an AI system, this conception of an ontology is equivalent to the original philosophical definition of Ontology as a "theory of existence", since, as Gruber (1995) put it, "for AI systems, what 'exists' is that which can be [formally] represented".

[5] Adopting a proposal of Guarino and Giaretta (1995), we use the uncountable noun 'Ontology' (with capital 'O') to refer to the philosophical discipline; in contrast, the countable noun 'ontology' (with lowercase 'o') refers to a specific ontological theory, such as 'Aristotle's ontology' or 'the Cyc ontology'.

[6] An ontology is said to be consistent if it does not contain any logically conflicting statements.

[7] 'Knowledge representation' is often used synonymously with 'information representation'.

In computer science, there are two principal types of usage for an ontology:

- The first type of usage has already been explicated in Chap. 1: An ontology serves as a library of knowledge components to efficiently build intelligent systems. To this aim, the generic ontology is to be transformed (i.e., extended and customized) into a knowledge base according to the requirements of the respective application.
- The second type of usage is as a *shared vocabulary* for communication between interacting human and/or software agents. According to their respective functions, the communicating agents may have different knowledge bases, but all the knowledge bases must be consistent with the ontology (Gruber 1995).

Both types of usage make the same demand on the ontology: They both require a consensual knowledge representation that is reusable in different application contexts. For the first case, this is obvious and has been extensively discussed in Chap. 1. As for the second case, the communicating agents perform different tasks requiring different knowledge bases, and thus the ontology must be suitable for each of these[8]. Thus, a properly crafted ontology should be applicable to both types of usage. As will be explained in Chaps. 11 and 12 OntoCAPE originally started as a shared vocabulary, but later evolved to a library for building knowledge-based systems.
Guarino (1998) points out that in philosophy, the term 'ontology' denotes a conceptual framework, whereas in computer science, 'ontology' often (but not always) refers to the engineering artifact used to represent such a conceptual framework:

> "In the philosophical sense, we may refer to an ontology as a particular system of categories accounting for a certain vision of the world. As such, this system does not depend on a particular language: Aristotle's ontology is always the same, independently of the language used to describe it.
> On the other hand, in its most prevalent use in AI, an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words."

In this book, the term 'ontology' is used ambiguously with both meanings. If necessary, we will use the phrases 'ontology at the syntactic level' to refer to the engineering artifact, and 'ontology at the semantic level' to refer to the abstract conceptual framework[9].

[8] The only difference is that, in the first case, the ontology is directly reused for building the knowledge base, whereas this is not necessarily true in the second case. Yet even if a knowledge base has not developed directly from the ontology, it must still be consistent with the ontology's definitions.

[9] The synonymous terms 'symbolic level' and 'knowledge level', first suggested by Newell (1982), are also used in the literature.

Note that in computer science, the term ‘ontological commitment’ has a special meaning, as well: If some human or software agents agree on using an ontology for a given task in a consistent manner, they are said to commit to that ontology (Gruber and Olsen 1994; Studer et al. 1998). In other words, “an agent commits to an ontology if its observable actions are consistent with the definitions in the ontology” (Gruber 1995).

2.3 Representation of Formal Ontologies

Ontologies can be modeled with different modeling techniques, and they can be implemented in various kinds of languages (Uschold and Grüninger 1996). Examples of common modeling techniques, or *modeling paradigms*, include frames (e.g., Minsky 1975), first-order logic (e.g., Hodges 1983), description logic (abbr. DL; e.g., Baader et al. 2003), database modeling techniques (e.g., Chen 1976), and rule-based languages (a.k.a. rule languages; e.g., Lloyd 1987); for each paradigm, multiple implementations, or *modeling languages*, exist.
In spite of their diversity, the different modeling languages share structural similarities and have comparable modeling elements. In particular, most languages provide constructs for *classes*, *individuals*, *relations*, and *attributes*, although they may be named differently in the respective implementations. Moreover, some language allow for the definition of *axioms*. In the following, these different model components will be described in detail.

2.3.1 Classes and Individuals

A *class* represents a collection of entities that share a common characteristic. Depending on the respective modeling paradigm, classes are also denoted as *concepts* or *frames*. If referred to in the text, class identifiers are highlighted by *italicized sans-serif font*.
Entities that belong to a particular class are said to be *instances* or *members* of that class; for example, **water** and **ethanol** are instances of the *substance* class. Some modeling languages allow for the definition of *metaclasses*, the instances of which are again classes. The instances of an ordinary class are called *individuals*. Throughout this book, individuals are accentuated by **bold sans-serif font**.
Classes can be hierarchically organized by means of *subsumption relations*, which are also known as *specialization relations* or *subclassing relations*: The class *B* is said to be a specialization or a subclass of the class *A* if every instance of *B* is also an instance of *A*. In this case, *B* is said to be *subsumed* by *A*, and *A* is called a *superclass* of *B*.

By means of *axioms* (see below), it is possible to state certain properties about a class, such as the existence of *relations* (see below). In this context, two types of classes can be distinguished:

- *Primitive classes* have only necessary conditions (expressed in terms of their properties) for membership: An instance of a primitive class must always comply with the properties of that class, but there may be other individuals with the same properties which are not members of that class. Consequently, membership to a primitive class must be explicitly stated.
- *Defined classes* are characterized by necessary and sufficient conditions for membership. Thus, an individual whose properties match those of a defined class is automatically inferred to be a member of that class. Similarly, the subclasses of a defined class can be inferred if their properties match the class definition.

Most languages support *inheritance* between the classes in a subsumption hierarchy; that is, a subclass inherits all the properties of its superclass. Some languages allow for *multiple inheritance*, which means that a particular class can inherit properties from more than one superclass.

2.3.2 Relations

A *relation* represents an interrelation between some classes; depending on the respective modeling paradigm, relations are also called *properties*, *roles*, *slots*, or *associations*. While most modeling languages only provide modeling constructs for *binary relations* (i.e., relations between exactly two classes), a few have built-in constructs for *higher-arity relations* (a.k.a. *n-ary relations*) involving three or more classes. In the following, the term 'relation' is synonymously used for 'binary relation'. Relation identifiers will be denoted by sans-serif font throughout the text.

By default, a relation is (uni-)directional, which means that it points from a particular *domain* class to a designated *range* class: As an example, consider the relation hasReactant, which refers from a *chemical reaction* (its domain) to a *substance* (its range).

A relation can be instantiated, which means that it can be applied between an instance of the domain class and an instance of the range class. For example the above hasReactant relation can refer from the **esterification of acetic acid** (an instance of *chemical reaction*) to the individual **ethanol**. Unlike a class instance, an instantiated relation is not given a specific name but is identified via its domain and range individuals.

Some languages allow to further specify the relations by means of *relation properties* (sometimes called *property characteristics*). The following relation properties are quite common, although a single language does not necessarily support all of them:

- A relation may be associated with another relation denoting its *inverse* – for example, isReactantOf would be the inverse of hasReactant, thus pointing from a *substance* to a *chemical reaction*.
- Alternatively, a relation may be declared to be *symmetric* – in this case, it is equivalent to its own inverse: A concrete example is the isEqualTo relation – it implies that, if A is equal to B, then B is equal to A, as well.
- A different property is *antisymmetry*, which is defined as follows: Given an antisymmetric relation R and two entities, A and B. If A is R-related to B, and B is R-related to A, then A and B must be identical. Note that symmetry and antisymmetry are not mutually exclusive – for instance, the isEqualTo relation is both symmetric and antisymmetric.
- Additionally, a relation may be declared to be *transitive*. This means that if entities A and B are related via a transitive relation R, and so are B and C, then A and C must also be R-related. A concrete example would again be the isEqualTo relation – if A equals B, and if B equals C, then A equals C.
- A relation may be declared to be *reflexive*, meaning that each entity to which a reflexive relation R is applicable is R-related to itself. For instance, the isEqualTo relation is reflexive since each entity is equal to itself.
- Alternatively, a relation R may be declared to be *irreflexive*; in consequence, an entity can never be R-related to itself. The relation isGreaterThan is a typical example of an irreflexive relation.
- A *functional* relation (sometimes also referred to as a *function*) cannot have more than one unique range individual; if a domain individual is related to more than one range individual via a functional relation, it will be concluded that the range individuals are identical. For obvious reasons, this property should not be combined with transitivity.
- The opposite effect is caused by an *inverse-functional* relation: If two domain individuals are related to the same range individual via an inverse-functional relation, it will be inferred that the domain individuals are identical. Thus, the range individuals of an inverse-functional relation can be utilized as unique identifiers for the domain individuals. Note that the inverse of a functional relation is automatically an inverse-functional relation.

A few modeling languages treat subsumption as a special case of a (transitive, reflexive, and antisymmetric) relation. Other languages allow for a hierarchical organization of relations, which is similar to that of classes. Unlike in class hierarchies, a *subrelation* may have properties different from those of its *superrelation*.

2.3.3 Attributes

Attributes represent features, characteristics, or parameters of classes and their instances. An attribute is identified by its name; it takes one or several values, which are specific to the class or instance the attribute is attached to. Usually, the values of a particular attribute are restricted to a specific datatype such as boolean, string, or integer.
Often, the same modeling constructs are used for the representation of relations and attributes; they differ from each other only with respect to their ranges: The range of a relation is given by its range class, whereas the range of an attribute is specified by its datatype. Due to the absence of a range class, most of the above relation properties cannot be applied to attributes. However, it is possible to declare an attribute to be functional or inverse functional; also, attributes may be hierarchically ordered.

2.3.4 Axioms

An *axiom* models a proposition or sentence that is always true. Generally, axioms provide an additional means for knowledge representation: They allow formalizing such knowledge that goes beyond stating the mere existence of classes, relations, and instances. Therefore, modeling paradigms that include axioms have a greater expressiveness than those without. In particular, axioms serve

- to explicitly define the semantics (or at least to constrain the possible interpretations and uses) of an ontological concept by imposing constraints on its values and/or its interactions with other concepts in the ontology;
- to verify the consistency of the knowledge represented in the ontology; and
- to infer new (i.e., formerly implicit) knowledge from the explicitly stated facts.

Formal axioms may be embedded in class or relation definitions, where they specify the properties of the respective class or relation. In fact, the declaration of the above introduced relation properties is usually realized by means of embedded axioms.
The following are common types of class-embedded axioms, stating

- the *disjointness* of classes – if the classes A and B are declared to be disjoint, then an instance of class A cannot simultaneously be an instance of class B;
- the *equivalence* of classes, meaning that such classes have precisely the same instances;
- the *extension* of a class by means of an explicit *enumeration* of its members.

Another common type of class-embedded axioms puts constraints on the relations originating from the respective class. Unlike relation properties, which are universally valid, these constraints are specific to the domain class, i.e., they are only locally valid. These *local constraints* include, but are not restricted to

- *(local) range restrictions*, stating that the range of a relation originating from the domain class is restricted to certain classes;[10]
- *cardinality constraints*, which specify either the exact number or the maximum/minimum number of range individuals for a given relation;
- *qualified cardinality restrictions* (a.k.a. qualified cardinality constraints, abbr.: QCR), which, in addition to specifying the number of range individuals, also prescribe the range class of which the individuals are to be instantiated from.[11]

The above introduced basic axiom types can be combined to more complex expressions. To this end, ontology languages provide additional constructors, such as the set operators of *union*, *intersection*, and *complement*.
Finally, *rules* constitute a further, very powerful mechanism for stating axioms. A rule axiom consists of an *antecedent* (or *rule body*) and a *consequent* (or *rule head*). Both the antecedent and the consequent are logical expressions, which are formulated in terms of the other constructs of the modeling language. Whenever the expression specified in the antecedent holds true, then the expression specified in the consequent must also hold. Thus, if an antecedent matches the current state of the ontology, then the consequent is affirmed, i.e., added to the ontology. Note that, while the antecedent is not necessarily true, the rule as a whole is universally valid, and therefore matches the above definition of an axiom; 'classical' axioms (i.e., axioms without a precondition) can be modeled as rules with an empty rule body.

2.3.5 Modularization

Virtually all of the modern ontology modeling languages support the modularization of ontologies, i.e., the subdivision of an ontology into small, manageable pieces. This requires two complementary mechanisms: (1) a *clustering mechanism* for grouping a subset of interdependent model components (classes, instances, relations, attributes, and accompanying axioms) into a common module, and (2) an *inclusion* or *import mechanism*, which allows including the model components of

[10] Local range restrictions are typically formulated by means of the universal quantifier (∀).

[11] Postulating the existence of at least one instance of a particular range class is a special case, which can be formulated by means of the existential quantifier (∃).

some ontology module into another module[12]. That way, an ontology can be organized as an inclusion hierarchy of interdependent subontologies.

2.3.6 Notation of Modeling Elements

Having established the major elements of ontology modeling languages, we will now introduce a graphical notation for these elements. This notation, which is based on the UML notation for class diagrams (e.g., Fowler 1997), will be applied throughout this book. Its main components are depicted in Fig. 2.1.

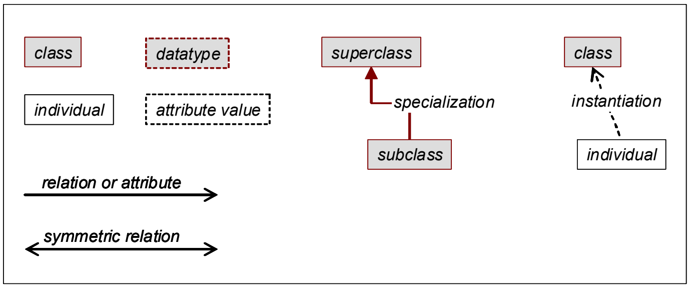

Fig. 2.1: Basic elements for the graphical representation of ontologies

Grey shaded boxes with solid boundary lines represent *classes*, white boxes represent *individuals*. *Datatypes* are denoted by grey shaded boxes with dashed boundary lines, *attribute values* by white boxes with dashed boundary lines. *Specialization* is depicted through a solid line with a solid arrowhead pointing from the subclass to the superclass. A dashed line with an open arrowhead denotes *instantiation*. Binary *relations* are depicted through solid lines, thereby distinguishing three different cases: a line with one open arrowhead represents the standard case of an *unidirectional* relation; a line with two open arrowheads represents a *symmetric* relation; finally, a line without any arrowheads represents a *relation and its inverse* (cf. Fig. 2.2). Please note, that there are further specializations of relations (i.e. for aggregation and composition) which are introduced in detail in Sect. 5.1.3. *Cardinality constraints* are depicted by numbers placed close to the range class of the respective relation. No particular symbols are provided for the other types of axioms.

[12] Inclusion means that if module A includes module B, the model components specified in B are valid in A and can thus be directly used (i.e., extended, refined …) in A. Inclusion is transitive, that is, if module B includes another module C, the ontological definitions specified in C are valid and usable in A, as well.

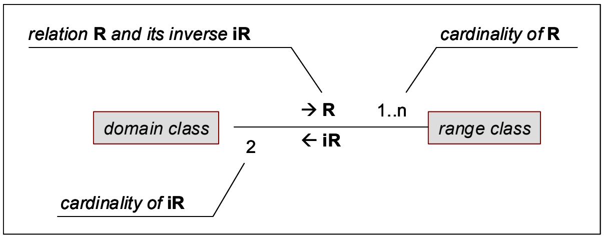

Fig. 2.2: Graphical notation for cardinalities and inverse relations

Generally, classes and relations will be named in accordance with the CamelCase[13] naming convention: UpperCamelCase notation is used to denote identifiers of classes, while relation identifiers are represented in lowerCamelCase notation. No particular naming convention is followed for identifiers of individuals. For better readability, the UpperCamelCase notation is not applied in the text; instead, the individual words that constitute the class identifiers are written separately and in lowercase (e.g., *class identifier*).

2.4 Informal and Formal Specification of an Ontology

Next, we need to discuss the overall form that an ontology must have at the syntactic level in order to be of practical use. An often quoted definition for an ontology stipulates that "an ontology may take a variety of forms, but it will necessarily include a *vocabulary of terms* and some *specification of their meaning*. This includes definitions and an indication of *how concepts are inter-related*" (Uschold et al. 1998). Smith (1996) further postulates that "the ontology should be […] explained in ways which make its content intelligible to human beings, and […] implemented in ways which make this content accessible to computers". From these statements, it can be concluded that two different representations of the ontology are required for practical use, which are referred to as *formal specification* and *informal specification* hereafter[14]. The formal specification is to be processed by AI systems, while the informal specification addresses the human users of the ontology.

– The *formal specification* constitutes an implementation of the ontology in machine-readable form. It specifies the meaning of the vocabulary terms

[13] CamelCase is the practice of writing compound words joined without spaces; each word is capitalized within the compound. While the UpperCamelCase notation also capitalizes the initial letter of the compound, the lowerCamelCase notation leaves the first letter in lowercase.

[14] The informal and the formal specification are different ontologies at the syntactic level, but they represent the same ontology at the semantic level.

and constrains their interrelations (and thus their possible uses) by means of axiomatic definitions, which are stated in a formal modeling language.

– The *informal specification* expresses the definitions of the formal specification in human-readable form. Particularly, it clarifies the meaning of the ontological vocabulary by giving precise term definitions in natural language. Additionally, the interrelations of the terms and their intended usage are described in some appropriate way (e.g., through UML-like diagrams and/or textual descriptions). Some further documentation may be provided, which goes beyond the knowledge stated in the formal specification – for instance, user guidelines for the extension of the ontology.

One of the most common current formal modelling languages is the *OWL Web Ontology Language* (Smith et al. 2004; Bechhofer et al. 2004). OWL and its predecessor *DAML+OIL* (Connolly et al. 2001) are ontology markup languages that have been developed for publishing and sharing ontologies in the Web. Their syntax is based on existing Web markup languages, the most prominent of which is XML (W3C 2006). By now, DAML+OIL has been superseded by its successor OWL, which has been endorsed as a *W3C recommendation*[15]. As OWL is derived from DAML+OIL, it shares most of its features (a listing of the differences between the two languages can be found in Appendix D of Bechhofer et al. 2004). Therefore, only OWL will be discussed in the following.

Model entities are represented through *classes* and *individuals* in OWL. Classes can be hierarchically ordered, thereby allowing multiple inheritances. They can also be further specified through class-embedded axioms stating the *disjointness* of classes, the *equivalence* of classes, or the *extension* of a class. These basic axiom types can be combined by means of the set operators of *union*, *intersection*, and *complement*.

Furthermore, OWL provides language primitives for *attributes* (called 'datataype properties') and *binary relations* (called 'object properties'); higher-arity relations must be represented through classes in OWL. Attributes and relations can be hierarchically ordered, and their usage can be restricted through *range* and *cardinality constraints*. Relations may be further specified through axioms declaring a relation to be *transitive*, *symmetric*, *functional*, or *inverse-functional* (the latter two are also applicable to attributes). Additionally, two distinct relations can be declared to be *equivalent* to, or the *inverse* of, each other. Modularization is supported by the *import* mechanism of OWL, which allows including the definitions and axioms of other ontologies into the current ontology.

The OWL language provides three increasingly expressive sublanguages, called *OWL Lite*, OWL DL, and *OWL Full*. Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can

[15] A W3C recommendation is the final stage of a ratification process of the World Wide Web Consortium (W3C) concerning a standard for the Web. It is the equivalent of a published standard in other industries.

be validly concluded (Smith et al. 2004). Save for a few exceptions, the representation of OntoCAPE, is restricted to the OWL DL subset. This sublanguage is compatible with a particular type of description logic (DL) called *SHOIN(D)* (Horrocks and Patel-Schneider 2004). As a consequence, the models represented in OWL DL can be processed with standard DL reasoners.

The current release of OWL (version 1.0) lacks certain language constructs, such as those for the relation properties of antisymmetry and reflexivity, or for the representation of qualified cardinality constraints. These (and other) language constructs will be included in the next release of OWL DL, which will move from the *SHOIN(D)* Description Logic to the more expressive *SROIQ(D)* Description Logic (Patel-Schneider and Horrocks 2006).

Rules are currently not part of OWL; however there are plans for an additional rule language that is to be defined on top of OWL (Horrocks et al. 2004). Yet the problem of how to efficiently combine logic-based reasoning and rule-based reasoning still remains to be solved.

2.5 What an Ontology Is and Isn't

With the growing popularity of web-enabled ontology languages like OWL (Smith et al. 2004; Bechhofer et al. 2004), the term 'ontology' is more and more being used in an inflationary manner to denote all kinds of knowledge representation structures. In many of these cases, it is erroneously assumed that the mere use of an ontology modeling language qualifies the respective structure as an ontology. However, this is definitely not the case: Being represented in an ontology modeling language is only a necessary, but not a sufficient criterion for being considered a (formal) ontology.

To better illustrate our point of view, we will below identify two types of ontology-like structures that we do not categorize as full-fledged ontologies: We refer to them as *pseudo ontologies* and *lightweight ontologies*, respectively. In the following, we will define these terms and explain why they do not comply with our – admittedly quite strict – conception of an ontology.

By "pseudo ontology" we mean a part of a software system that is formulated in a formal ontology language such as OWL, but has not been explicitly designed for reuse. A typical example would be the knowledge base of an intelligent system: In our judgment, such a knowledge base – or rather the state-independent part of that knowledge base (cf. Sect. 2.6) –can only be considered an ontology if it is reusable and can thus be shared across software applications and by different groups of

users[16] (cf. Chap. 1). If, on the other hand, the knowledge base has been designed for a single purpose only, we refer to it as a pseudo ontology.
In addition to pseudo ontologies, a second class of ontology-like structures must be differentiated from "true" ontologies. Unlike before, the differentiating factor is not the reusability of the respective structure, but its semantic richness: Structures of this class are not considered full-fledged ontologies as they do not formally define the semantics of the vocabulary terms through axiomatic definitions. Due to their simple internal design, they are sometimes referred to as 'lightweight ontologies' in the literature – as opposed to 'heavyweight ontologies', which model the domain in a deeper way and provide more restrictions on domain semantics (Gómez-Pérez et al. 2004). While a lightweight ontology may be represented in a formal ontology modeling language, it utilizes only a subset of the available modeling elements – that is, a lightweight ontology is built using classes, sometimes instances, and possibly relations, but it does not include relation properties, local constraints, or other forms of axioms. Four types of lightweight ontologies may be distinguished:

– A **controlled vocabulary** is a list of predefined, authorized terms with an unambiguous description given in natural language. The terms may be modeled as classes or instances, but there are no further axiomatic specifications of the meaning of terms.
– A **taxonomy** is a controlled vocabulary that is organized in a hierarchical structure; the hierarchy is usually modeled by means of subsumption relations.
– A **thesaurus** is a taxonomy that additionally specifies certain semantic relationships between its vocabulary terms. Unlike a semantic network (see below), a thesaurus includes only very few types of semantic relationships (typically the synonyms or near-synonyms and the antonyms of a term). These relationships can be modeled through associative relations.
– A **semantic network** is a knowledge representation formalism, which describes terms their relationships in form of a network consisting of labeled nodes and arcs. Typically, the labels of the nodes are nouns, and the labels of the arcs are verbs; that way, the triple formed by two nodes and the interconnecting arc represents a declarative sentence of the form subject-predicate-object. The nodes can be modeled as classes and/or instances, and the arcs can be modeled through associative relations.

Some ontologists (e.g., Guarino 1998; Lassila and McGuinness 2001) prefer a gradual approach to defining ontologies. They do not draw a clear distinction between lightweight and heavyweight ontologies, but postulate an "ontology spectrum" (McGuinness 2002), which ranges from simple taxonomies to sophisticated

[16] This view is supported by numerous ontologists, such as Neches et al. (1991), Borst (1997), Studer et al. (1998), Chandrasekaran et al. (1999), Jarrar and Meersman (2002), Gómez-Pérez et al. (2004), Smith (2006), or Pâslaru-Bontaş (2007).

heavyweight ontologies: Originating from taxonomies, the level of complexity is incrementally increased by adding instances, relations, relation properties, local constraints, and finally global axioms.

2.6 Classification of Ontologies

As the final topic of this theory chapter, a classification framework for ontologies is introduced, and the interdependencies between the different ontology types are discussed. In the later chapters of this book, the classification framework will serve as a frame of orientation to clarify the roles of the individual subontologies that constitute OntoCAPE as well as the roles of those ontologies that are related to OntoCAPE.

According to Guarino (1997b), ontologies can be classified into the following types, which are distinguished by their level of dependence on a particular task or point of view:

- *Top-level ontologies* define general-purpose concepts like object, state, action, etc., which are independent of a particular problem or domain and can therefore be universally applied. In the literature, top-level ontologies are also referred to as *abstract ontologies* (e.g., Borst 1997), *generic ontologies* (e.g., van Heijst et al. 1997a), *foundation(al) ontologies* (e.g., Schneider 2003), or *upper (level) ontologies* (e.g., Guarino 1998). Prominent examples of top-level ontologies are the Top-Elements Classification by Sowa (1995), UpperCyc (Lenat and Guha 1990), or the Suggested Upper Level Merged Ontology SUMO (Niles and Pease 2001).
- *Domain ontologies* capture the knowledge of a domain of expertise, such as medicine or engineering. A domain ontology is not specifically tailored to a particular task or application; instead, it defines general domain knowledge that is relevant for a wide range of different tasks and applications. The goal of a domain ontology is to be universally applicable (and thus reusable) within the respective domain of expertise.
- A *task ontology* (often also referred to as *method ontology*) describes general problem-solving methods that can be applied in different contexts. Such methods are task-specific, but the task itself should be generic in the sense that it occurs in different applications and domains of expertise. An example of a generic task would be graph searching, for which different search methods (e.g., depths-first search or breadth-first search) could be specified in a task ontology. Note that a task ontology does not actually realize (i.e., implement) the method, but only specifies the "terminology for expressing the competence and the knowledge requirements of a method" (Fensel et al. 1996). For a graph searching method, the terminology could, for example, include the concepts of 'current node', 'visited node',

'search depth', etc. Do also note that domain ontology and task ontology have different but complementary objectives with respect to reusability: the former is applicable to different tasks but restricted to a particular application domain; the latter is designated for a particular task but reusable across domains.

– Finally, an *application ontology* provides the concepts that are required for a particular application. To clarify the difference between an application ontology and a knowledge base, Guarino (1997b) proposed the following definition: An application ontology comprises only state-independent information (i.e., facts that are always true), whereas a knowledge base may also hold state-dependent information (i.e., facts and assertions related to a particular state of affairs).

The interdependencies between these four ontology types are depicted in Fig. 2.3: According to Borst (1997) and Guarino (1997b), a task ontology may import the terminology from a top-level ontology and utilize it for the specification of methods. In a similar manner, a domain ontology may describe domain concepts as specializations of the top-level concepts. Furthermore, the concepts in an application ontology can typically be defined by combining and refining concepts from both a domain and a task ontology; this is particularly facilitated if the domain and task ontology are founded on the same top-level concepts and thus share a common world-view. As an example, consider a top-level ontology that introduces the terminology to describe directed graphs. Based on this terminology, a task ontology could specify a graph searching method. Likewise, a domain ontology for chemical engineering could define the concept of a process flowsheets as a special form of a directed graph. An application ontology could finally combine domain knowledge and problem-solving knowledge in order to realize a search application for process flowsheets.

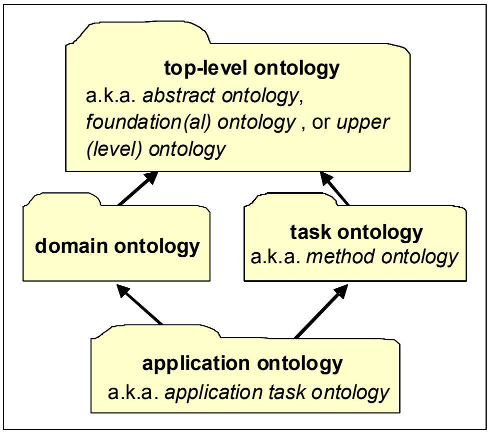
top-level ontology
a.k.a. *abstract ontology*, *foundation(al) ontology*, or *upper (level) ontology*
domain ontology
task ontology
a.k.a. *method ontology*
application ontology
a.k.a. *application task ontology*

Fig. 2.3: Ontology types and interdependencies according to Guarino (1997b); arrows indicate specialization relationships

While the above classification framework is widely accepted in principle, some points remain subject to debate: The borderline between top-level ontologies on the one hand, and domain and task ontologies on the other hand, is rather vague, as pointed out by van Heijst et al. (1997a); yet, as further argued by these authors, the distinction is intuitively meaningful and useful for building libraries of reusable ontologies. More controversial is the question whether or not it is feasible to separate domain knowledge from knowledge about problem-solving methods (cf. the discussion between van Heijst et al. 1997a, 1997b, and Guarino 1997a). At the core of the discussion is the so-called *interaction problem* (Bylander and Chandrasekaran 1988), which states the following: a method cannot be described without knowing the domain knowledge it will be applied to, and, vice versa, domain knowledge cannot be represented without knowing for what tasks or methods it will be used. Guarino (1997a), while admitting the validity of the interaction problem in principle, argues that one should nevertheless strive for a task-independent representation of domain knowledge; even though the goal cannot be fully achieved, it is quite possible to build a domain ontology that is reusable for a large number of different tasks.
As an extension to the above classification framework, some authors introduce subtypes and combinations of the four basic ontology types:

- Gómez-Pérez et al. (2004) recognize the so-called *general ontologies* (van Heijst et al. 1997a) or *common ontologies* (Mizoguchi et al. 1995) as an additional, distinct type of ontologies. According to these authors, ontologies of this type represent common-sense knowledge that is reusable across domains. However, the differentiating criterion between top-level ontologies and common ontologies remains vague – presumably, a top-level ontology contains only high-level concepts, which must be specialized in domain and task ontologies to become usable, whereas the concepts of a common ontology are directly applicable. A special type of a common ontology would be a *supertheory* – the term has been coined by Borst (1997) to denote an abstract ontology that defines a self-contained theory. Prominent examples of this category are the mereology and topology ontologies created by Borst (1997).
- Some authors (e.g., Mizoguchi et al. 1995; Gómez-Pérez et al. 2004) explicitly subdivide a task ontology in a task part and a method part; only the former part is then referred to as '*task ontology*', while the latter part is called '*method ontology*'.
- Gómez-Pérez et al. (2004) additionally introduce the type of a *domain-task ontology* which is defined as an application-independent task ontologies that is reusable in a given domain, but not across domains.
- Pâslaru-Bontaş (2007) differentiates between *application domain ontologies* and *application task ontologies*. The former refines and extends the general-purpose knowledge of a domain ontology to the requirements of a particular application, whereas the latter corresponds to a combination of

application-relevant domain and task-related knowledge, similar to the application ontologies introduced by Guarino (1997b).

– Some authors (e.g., Valente and Breuker 1996; van Heist et al. 1997a; Doerr et al. 2003) suggest an additional ontology type called *core ontology*. In the literature, there is no general agreement on what constitutes a core ontology. A core ontology, as we understand it (cf. Brandt et al. 2008a; Morbach et al. 2007; Chap. 12), constitutes the top-level part of an application ontology. More specifically, the function of a core ontology is (1) to select and retrieve the top-level concepts that are relevant for the particular application from the respective domain and task ontologies, (2) to specify how these concepts are to be used (i.e., interpreted) by the application, and (3) to introduce additional top-level concepts required by the application that cannot be retrieved from the available ontologies.

A further type of ontologies, which is not covered by the above classification framework, is the so-called *(knowledge) representation ontology*. Representation ontologies explicate the conceptualizations that underlie knowledge representation formalisms (Davis et al 1993). They are intended to be neutral with respect to world entities (Guarino and Boldrin 1993). That is, they provide a representational framework without making claims about the world (van Heijst et al. 1997a). Top-level ontologies as well as domain and task ontologies are described through the primitives provided by representation ontologies. Well-known examples of this ontology type are the Frame Ontology (Gruber 1993) or the representation ontologies for the Semantic Web languages RDFS (W3C 2000) and OWL (W3C 2002).

Finally, the notion of a *meta model*, or *meta ontology*, needs to be defined. Generally, a meta model is "a design framework, that describes the basic model elements and the relationships between the model elements as well as their semantics. This framework also defines rules for the use […] of model elements and relationships" (Ferstl and Sinz 2001, p. 86). There are two possible interpretations of the term 'meta model' which are consistent with this definition: for their differentiation, Atkinson and Kühne (2002) coined the terms *physical metalevel* and *logical metalevel*. A meta model at the physical metalevel defines the concepts and mechanisms of the modeling language and it thus equivalent to a representation ontology. By contrast, a meta model at the logical metalevel guides the development of the actual ontology by means of predefined types and patterns, which reflect modeling best practice.

Fig. 2.4 presents the extended classification framework, now including both types of meta ontologies[17]. Also, the degree of usability and reusability of the respective ontology types is shown in the figure: Compliant with the usability-reusability trade-off (cf. Sect. 1.3), the usability increases with the ontology type's degree of specialization, whereas its reusability decreases.

[17] The other ontology types introduced above are not depicted since they are merely subtypes of the ones presented in the figure.

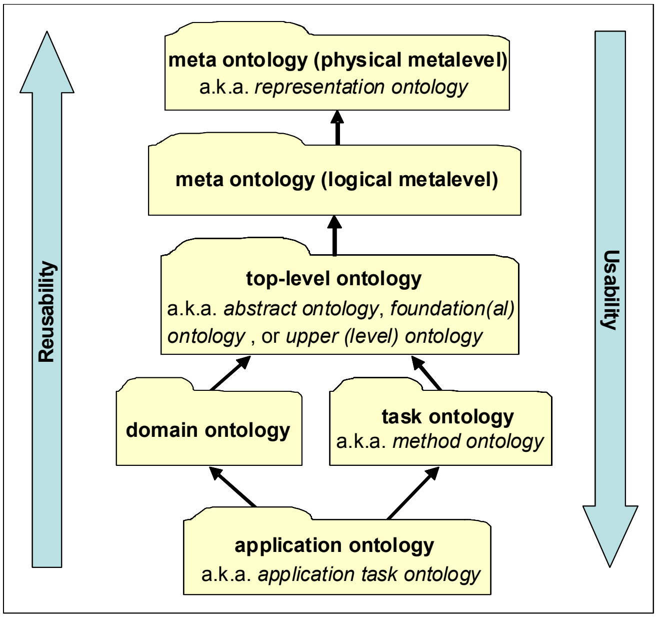
meta ontology (physical metalevel)
a.k.a. representation ontology
meta ontology (logical metalevel)
top-level ontology
a.k.a. abstract ontology, foundation(al) ontology , or upper (level) ontology
domain ontology
task ontology
a.k.a. method ontology
application ontology
a.k.a. application task ontology
Reusability
Usability

Fig. 2.4: Extended classification framework

2.7 Summary

We have contrasted the similar but different perceptions of ‘ontology’ in the areas of philosophy and computer science: In the former discipline, an ontology denotes a theory of existence, which may be formulated on any level of formality; it is created for no specific purpose but to gain insight into the respective universe of discourse. In computer science, by contrast, an ontology is created for practical use – either as a shared vocabulary for communication between interacting agents or as a library of reusable knowledge components for building intelligent systems; moreover, a computer science ontology is always formal (and thus machine-interpretable), even though the provision of an additional informal specification for human users is highly advisable.

Over the last decades, several modeling paradigms and modeling languages have been proposed for the representation of formal ontologies. We have presented the common elements and pointed out the differences of these paradigms and languages.

Different types of ontologies can be differentiated: Firstly, one needs to distinguish between full-fledged ‘heavyweight’ ontologies and ‘lightweight’ ontologies, which do not make use of axiomatic definitions. Secondly, one must distinguish truly reusable ontologies from ‘pseudo ontologies’, which are built for a single application only. Finally, an ontology may be partitioned into sub-ontologies of different types, which can be classified according to their respective functions; the most common types, ordered by increasing usability, are meta ontology, top-level ontology, domain ontology, task ontology, and application ontology.

2.8 References

Atkinson C, Kühne T (2002) The role of metamodeling in MDA. In: *Proceedings of the Workshop in Software Model Engineering (in conjunction with UML’02, Dresden, Germany).* Online available at http://www.meta model.com/wisme-2002/papers/atkinson.pdf. Accessed January 2008.

Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (2003) *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, Cambridge.

Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness D, Patel-Schneider L, Stein LA (2004) *OWL Web Ontology Language Reference*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-ref/. Accessed September 2007.

Borst WN (1997) Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Brandt SC, Fritzen O, Jarke M, List T (2008a) Goal-oriented information flow management in development processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:369–400.

Bylander T, Chandrasekaran B (1988) Generic tasks in knowledge-based reasoning: the right level of abstraction for knowledge acquisition. In: Gaines B, Boose J (eds.): *Knowledge Acquisition for Knowledge-Based Systems*. Academic Press, London:65–77.

Chandrasekaran B, Josephson JR, Benjamins VR (1999) What are ontologies, and why do we need them? *IEEE Intell. Syst.* **14** (1):20–26.

Chen PP (1976) The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems* **1** (1):9–36.

Connolly D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2001) *DAML+OIL reference description*. W3C Note, 18 December 2001. Online available at http://www.w3.org/TR/daml+oilreference. Accessed January 2008.

Davis R, Shrobe H, Szolovits P (1993) What is a knowledge representation? *AI Mag.* **14** (1):17–33.

Doerr M, Hunter J, Lagoze C (2003) Towards a core ontology for information integration. *J. Digit. Inf.* **4** (1), Article No. 169.

Fensel D, Schönegge A, Groenboom R, Wielinga BJ (1996) Specification and verification of knowledge-based systems. In: Gaines BR, Musen MA (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Ferstl OK, Sinz EJ (2001) *Grundlagen der Wirtschaftsinformatik, Bd. 1*. Oldenbourg, München.

Fowler M (1997) UML Distilled – Applying the Standard Object Modeling Language. Addison-Wesley.

Gómez-Pérez A, Fernández-López M, Corcho O (2004) *Ontological Engineering*. Springer, Berlin.

Gruber TR (1993) A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.* **5** (2):199–220.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

Guarino N (1997a) Understanding, building, and using ontologies: A commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga. *Int. J. Hum Comput Stud.* **46** (2/3):293–310.

Guarino N (1997b) Semantic matching: formal ontological distinctions for information organization, extraction, and integration. In: Pazienza MT (ed.): *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer, Berlin:139–170.

Guarino N (1998) Formal ontology and information systems. In: Guarino N (ed.): *Formal Ontology in Information Systems*. IOS Press, Amsterdam:3–15.

Guarino N, Boldrin L (1993) Ontological requirements for knowledge sharing. In: Skuce, D. (ed.): Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing.

Guarino N, Giaretta P (1995) Ontologies and knowledge bases: towards a terminological clarification. In: Mars N (ed.): Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam:25–32.

Hodges W (1983) Elementary predicate logic. In: Gabbay DM, Guenthner F (eds.): Handbook of Philosophical Logic – Vol. I: Elements of Classical Logic. Reidel, Dordrecht:1–131.

Hofweber T (2005) Logic and ontology. In: Zalta EN (ed.): The Stanford Encyclopedia of Philosophy (Winter 2005 Edition). Online available at http://plato.stanford.edu/achives/win2005/entries/logic-ontology/. Accessed January 2007.

Horrocks I, Patel-Schneider P (2004) Reducing OWL entailment to description logic satisfiability. *J. Web Sem.* **1** (5):345–357.

Horrocks I, Patel-Schneider P, Boley H, Tabet S, Grosof B, Dean M (2004) *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C Member Submission 21 May 2004. Online available at http://www.w3.org/Submission/SWRL/. Accessed December 2007.

Jarrar M, Meersman R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine* SSGRR2002.

Lassila O, McGuinness D (2001) The Role of Frame-Based Representation on the Semantic Web. Technical Report (KSL-01-02), Knowledge Systems Laboratory, Stanford University. Online available at http://www-ksl.stanford.edu/KSL_Abstracts/KSL-01-02.html. Accessed October 2007.

Lenat D, Guha RV (1990) Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project. Addison Wesley.

Lloyd JW (1987) Foundations of Logic Programming, 2nd edition. Springer, Berlin.

McGuinness DL (2002) Ontologies come of age. In: Fensel D, Hendler J, Lieberman H, Wahlster W (eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press:171–194.

Minsky M (1975) A framework for representing knowledge. In: Winston PH (ed.): *The Psychology of Computer Vision*. McGraw-Hill, New York.

Mizoguchi R, Vanwelkenhuysen J, Ikeda M (1995) Task ontologies for reuse of problem solving knowledge. In: Mars N (ed.): *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam:46–57.

Morbach J, Yang A, Marquardt W (2007): OntoCAPE – a large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20** (2):147–161.

Neches R, Fikes R, Finin T, Gruber T, Patil R, Senator T, Swartout WR (1991) Enabling technology for knowledge sharing. *AI Mag.* **12** (3):36–56.

Newell A (1982) The knowledge level. *Artif. Intel.* **18** (1):87–127.

Niles P (2001) Towards a standard upper ontology. In: Guarino N, Welty C, Smith B (eds.): Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001). ACM:2–9.

Pâslaru-Bontaş E (2007) *Contextual Approach to Ontology Reuse: Methodology, Methods and Tools for the Semantic Web.* PhD Thesis, FU Berlin.

Patel-Schneider PF, Horrocks I (2006) *OWL 1.1 Web Ontology Language Overview*. W3C Member Submission, 19 December 2006. Online available at http://www.w3.org/Submission/owl11-overview/. Accessed October 2007.

Schneider L (2003) How to build a foundational ontology: the object-centered high-level reference ontology OCHRE. In: Günter A, Kruse R, Neumann B (eds.): *KI 2003: Advances in Artificial Intelligence*. Springer, Berlin:120–134.

Smith B (1996) Mereotopology: a theory of parts and boundaries. *Data Know. Eng.* **20** (3):287–303.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Smith MK, Welty C, McGuinness DL, eds. (2004) *OWL Web Ontology Languages Guide*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-guide/. Accessed October 2007.

Sowa JF (1995) Top-level ontological categories. *Int. J. Hum Comput Stud.* **43** (5/6):669–685.

Studer S, Benjamins VR, Fensel D (1998) Knowledge engineering principles and methods. *Data Knowl. Eng.* **25** (1/2):161–197.

Uschold M, Grüninger M (1996) Ontologies: principles, methods and applications. Knowl. Eng. Rev. 11 (2):93–136.

Uschold M, King M, Moralee S, Zorgios Y (1998) The enterprise ontology. *Knowl. Eng. Rev.* **13**:31–89.

Valente A, Breuker J (1996) Towards principled core ontologies. In: Gaines BR, Mussen M (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SDRG Publications.

van Heijst G, Schreiber AT, Wielinga BJ (1997a) Using explicit ontologies in KBS development. *Int. J. Hum Comput Stud.* **46** (2/3):183–292.

van Heijst G, Schreiber AT, Wielinga BJ (1997b) Roles are not classes: a reply to Nicola Guarino. *Int. J. Hum Comput Stud.* **46** (2/3):311–318.

W3C (2000) *The RDFS representation ontology*. Web resource. Online available at http://www.w3.org/2000/01/rdf-schema. Accessed June 2008.

W3C (2002) *The OWL representation ontology*. Online available at http://www.w3.org/2002/ 07/owl. Accessed October 2007.

W3C (2006) *Extensible Markup Language (XML)*. Online available at http://www.w3.org/XML/. Accessed December 2007.

3 Overview on OntoCAPE

Having established the scientific background of ontology engineering, we now present version 2.0 of the ontology OntoCAPE. Compliant with the terminology introduced in the previous chapter, OntoCAPE can be characterized as a *formal*, *heavyweight ontology*, which is represented in the OWL modeling language. It consists of several sub-ontologies, which perform different functions: According to the classification framework introduced in Sect. 2.6, the individual sub-ontologies serve the functions of a meta ontology (at the logical metalevel), a top-level ontology, a domain ontology, as well as some application ontologies.

3.1 Overview and Structure

As for any complex system, a sound architecture is critical for an ontology (1) to facilitate its efficient construction and long-term maintenance, and (2) to enable its reuse in different application contexts. In the following, we discuss how this concern is addressed by the design of OntoCAPE.

Fig. 3.1 gives an overview on OntoCAPE. As can be observed, the ontology is organized by means of two orthogonal structuring principles, which will be discussed in the two consecutive subsections: *abstraction layering* and *modularization*.

3.1.1 Abstraction Layering

To improve the usability and reusability of an ontology, numerous authors (e.g., Chandrasekaran and Johnson 1993; Russ et al. 1999; Borst 1997; Jarrar and Meersman 2002) have proposed the idea of structuring an ontology into different *levels of abstractions*. Following their recommendation, OntoCAPE has been subdivided by means of layers (cf. Fig. 3.1), which separate general knowledge from knowledge about particular domains and applications.

The design of each layer follows the principle of "minimal ontological commitment" (Gruber 1995) (cf. Sect. 10.5), meaning that a layer holds only those ontological terms and axioms that are essential for its function; terms and axioms that are not essential for the layer's purpose are sourced out to lower layers. The topmost *Meta Layer*, is the most abstract one; it holds a meta ontology (cf. Sect. 2.6), which introduces fundamental modeling concepts and states the design guidelines for the construction of the actual ontology. Next, the *Upper Layer* of OntoCAPE defines the principles of general systems theory according to which the ontology is organized. On the subjacent *Conceptual Layer*, a conceptual model of the CAPE

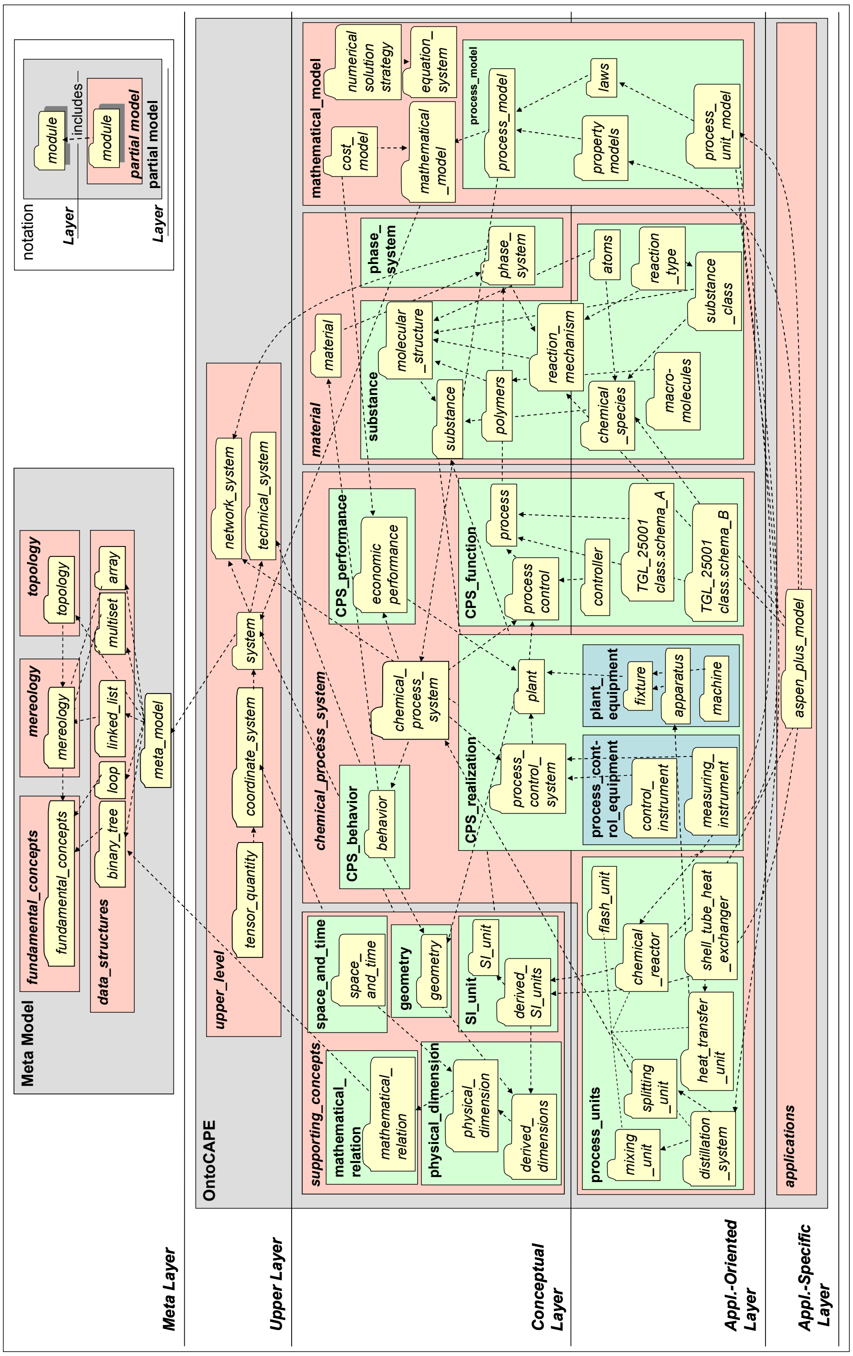

Fig. 3.1: Structure of OntoCAPE

domain is established, which covers such different areas as unit operations, equipment and machinery, materials and their thermophysical properties, chemical process behavior, modeling and simulation, and others. The two bottommost layers refine the conceptual model by adding classes and relations required for the practical application of the ontology: The *Application-Oriented Layer* generically extends the ontology towards certain application areas, whereas the *Application-Specific Layer* provides specialized classes and relations for concrete software applications.

The layered design takes the reusability-usability trade-off problem (cf. Sect. 1.3) into account: According to the trade-off problem, the general knowledge, which is located on the Upper Layer of OntoCAPE, can be reused in a variety of application contexts, but it is not immediately usable. By contrast, the knowledge located on the lower layers is ready for use, but problem-specific and thus hardly transferable to other applications. Thus, each layer contains knowledge of a specific degree of usability and reusability; traversing down the ontology, the usability of the knowledge increases, whereas its reusability decreases. Now, if (a part of) the ontology is to be reused for building some knowledge-based application, the appropriate abstraction level for knowledge reuse must be found. In practice, this means that one needs to traverse up the ontology (starting from the Application-Specific Layer) until the encountered knowledge is generic enough to fit the respective application context.

As an example, consider an intelligent CAPE tool, for the development of which a preferably large part of OntoCAPE is to be reused. The knowledge on the bottommost layer is application-specific and therefore of little value for any new tool. Yet already the above Application-Oriented Layer may contain some reusable knowledge, provided that the tool operates in an application area that is covered by OntoCAPE at all. If this is not the case, we need to move up to the Conceptual Layer; here, at the latest, some reusable knowledge can be found. Thus, for building a CAPE tool, one may reuse the ontology down to and including the Conceptual Layer at least. If, on the other hand, the tool was from a different application area than CAPE, it would still be possible to reuse knowledge from the Upper Layer and the Meta Layer.

3.1.2 Modularization

Modularization (i.e., the subdivision of the ontology into largely self-contained units), has been recommended by many authors (e.g., Gruber and Olsen 1994; Borst 1997; Bernaras et al. 1996; Visser and Cui 1998; Pinto et al. 1999; Heflin and Hendler 2000; Rector 2003; Stuckenschmidt and Klein 2003) as a means to promote the intelligibility (cf. Sect. 10.3), the adaptability (cf. Sect. 10.4), and generally the reusability of an ontology.

In OntoCAPE, modularization has been realized by partitioning the ontology into *modules* and *partial models*. Throughout the text, the identifiers of modules will be denoted in *italicized serif font*, whereas the identifiers of partial models will be denoted in **bold serif font**.

3.1.2.1 Modules

A *module* assembles a number of interrelated classes, relations, and axioms, which jointly conceptualize a particular topic (e.g., the module *plant* holds a conceptualization of chemical plants). The boundaries of a module are to be chosen such that the module can be designed, adapted, and reused to some extent independently from other parts of an ontology (Stuckenschmidt and Klein 2003). A module may include another module, meaning that if module A includes module B, the ontological definitions specified in B are valid in A and can thus be directly used (i.e., extended, refined …) in A. This allows to decompose OntoCAPE into an "inclusion lattice" (Gruber and Olsen 1994) of loosely coupled modules, as shown in Fig. 3.1.

By definition, modules have strong internal coherence but relatively loose coupling with the other parts of the ontology (Borst 1997), which facilitates their handling, thus improving the *adaptability* of the ontology (this issue is discussed in Sect. 10.4). Moreover, the modules are concise and therefore easier to comprehend than an entire ontology, hence bringing advantages with respect to *intelligibility* (cf. Sect. 10.3). Furthermore, the modular structure facilitates the selective reuse of the ontology: A user may choose to reuse only a selected part of the ontology if other parts are not relevant in the respective application context. In this case, it is relatively simple to cut the connections between the modules to be reused and the remainder of the ontology.

In the formal specification of OntoCAPE, modules are manifested through XML namespaces (Bray et al. 2006a). By convention, the concepts of a common namespace are stored in a single OWL file of the same name as the corresponding module. Inclusion is realized by means of the OWL import mechanism. Moreover, each module is (conceptually) assigned to one particular layer (cf. Fig. 3.2), thus integrating the structuring mechanisms of layering and modularization.

3.1.2.2 Partial Models

Modules that address closely related topics are grouped into a common *partial model* – for instance, the partial model **plant_equipment** clusters the thematically related modules *fixture*, *apparatus*, and *machine*.

The partial models constitute a coarse categorization of the domain. Unlike modules, partial models may be nested and may stretch across several layers. While the boundaries of the modules are chosen for practical considerations (i.e., such that the interdependencies between the individual modules are minimized), the boundaries of the partial models reflect the 'natural' thematic boundaries of the domain. In the course of ontology evolution (cf. Sect. 11.1), the partial model structure is therefore supposed to remain relatively stable, whereas the number of modules as well as their content and mutual dependencies are likely to change over time. Thus, the partial model structure provides a stable frame of orientation for the organization of the modules.

In the formal specification of OntoCAPE, the partial models are implemented as (file) directories. That way, they establish a directory structure for managing the OWL files.

3.1.2.3 Variants

As there is no unique way of modeling an area of interest, different *variants* of an ontology module may evolve. These variants represent alternative conceptualizations of the subject covered by the module. Variants will particularly arise on the application-near layers whenever the ontology is adapted to a new application because a new application usually implies a different view on the domain and consequently a different conceptualization (Noy and Klein 2004). Consider for example the module *plant* (cf. Sect. 8.3.1), which holds a conceptualization of plant equipments and their connections: A knowledge management system, as the one described in Sect. 12.2, calls for a simple, coarse-grained description of connectivity (i.e., the mere indication that equipment A is connected to equipment B is sufficient for this application). On the other hand, an ontology-based system for the integration of engineering data, as the one sketched in Sect. 12.3, would require a more precise description: Connectivity must be addressed by indicating the number and position of flanges, specification of their type and diameter, etc. Yet for the knowledge management system, these details are irrelevant and should thus be omitted. As a solution, two variants of the module *plant* can be developed, providing different conceptualizations of connectivity specifically tailored to the information demands of the respective application.

While the variants will predominantly emerge on the lower layers of the ontology, the basis for their evolvement must be established on the higher layers already, particularly on the Meta Layer and the Upper Layer. These layers define the fundamental theories on which the variants are based. Consequently, these theories must be formulated in a flexible manner, such that they allow for alternative refinements in form of different variants. In the above example, for instance, both variants of *plant* are based on a generic theory of connectivity formulated in the Meta Model (partial model **topology**, cf. Sect. 4.4). Thus, the generic theory must tolerate both the coarse-grained and the fine-grained specification of connectivity.

In the formal specification of OntoCAPE, the variants of a module are represented as separate OWL files. These files are stored in the same directory, and they have the first part of their two-part file name in common. However, this proceeding is only appropriate for a small number of variants; with an increasing number, the use of a variant management software (such as Pure::Variants; Pure Systems 2008) should be considered.

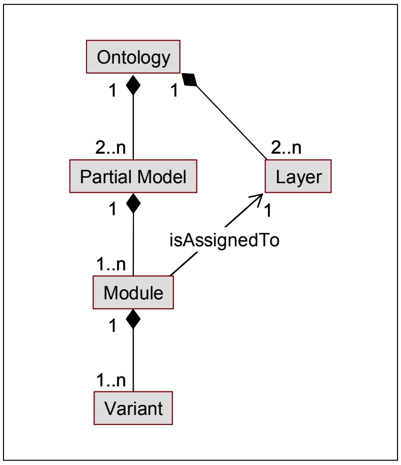

Fig. 3.2: Formal model of the structuring elements used for organizing OntoCAPE

Summarizing the above discussion, Fig. 3.2 displays a formal model of the structuring elements that have been used to organize OntoCAPE.

3.2 Representation and Dissemination

As stated in Sect. 2.6, a reusable ontology must be available both in form of an informal specification and in form of a formal specification. Accordingly, OntoCAPE 2.0 is issued in both forms of representation:

- The informal specification currently takes the form of six technical reports (Morbach et al. 2008f; 2008g; 2008h; 2008i; 2008j; Wiesner et al. 2008a), which jointly comprise about 500 pages. The important concepts introduced in these reports are summarized in the respective chapters of this book (cf. Chaps. 4-9). Within these chapters, the organization and structure of OntoCAPE are presented, and the conceptualizations of various topic areas are described in detail, often complemented by intuitive UML-like diagrams. Moreover, the major design decisions of ontology engineering are explicated, thus providing the reader with the necessary background knowledge for extending and customizing the ontology to his/her own purposes. Finally, the usage of the ontology is explained, and some sample applications are presented.

– As mentioned before, the formal specification of OntoCAPE has been realized in OWL. A short introduction to that particular modeling language can be found in Sect. 2.4. The current release of OntoCAPE consists of 62 OWL files, each of which includes one module of the ontology. In total, the implementation comprises about 500 classes, 200 relations, and 40,000 individuals[18].

According to Smith (2006), an ontology must be "open and available to be used by all potential users without any constraint, other than (1) its origin must be acknowledged and (2) it should not to be altered and subsequently redistributed except under a new name"[19]. Compliant with this demand, OntoCAPE is publicly accessible at http://www.avt.rwth-aachen.de/Ontocape. Via this webpage, both the informal and the formal specification of OntoCAPE can be accessed free of charge. OntoCAPE is distributed under the terms of the GNU General Public License (cf. GNU Project 2007)

3.3 The Meta Model

Fig. 3.3 gives an overview on the Meta Model. As can be observed, the Meta Model is partitioned into the partial models **fundamental_concepts**, **mereology**, **topology**, and **data_structures**. While both **mereology** and **topology** contain only a single module, **data_structures** comprises five: *array*, *linked_list*, *multiset*, *binary_tree*, and *loop*.
The module *meta_model* includes all these modules, thus assembling the ontological definitions of the Meta Model. The module *meta_model* is, in turn, included by the top-level module of the target ontology (shown here is the module *system*, which resides on the upper level of OntoCAPE). That way, the concepts defined in the Meta Model are available in the target ontology.
In the following, we will give a very brief overview on the structure and contents of these four partial models.

3.3.1 Fundamental Concepts

The partial model **fundamental_concepts** introduces meta root terms and their refinements. Meta root terms are the root classes and relations in the Meta Model and all other classes and relations – in the Meta Model as well as in the Onto-

[18] Virtually all of the individuals represent chemical species data.

[19] The formulation has been adopted from the distribution terms of the Open Biomedical Ontologies Foundry, as stated at http://www.obofoundry.org/crit.shtml.

CAPE ontology – can be derived from the meta root terms by specialization. Typical meta root terms are, for example, the classes object and relation class: The former subsumes all "self-standing" (Rector 2003) entities – whether physical or abstract – that exist in an application domain; the latter denotes all kinds of n-ary relations that may exist between objects.

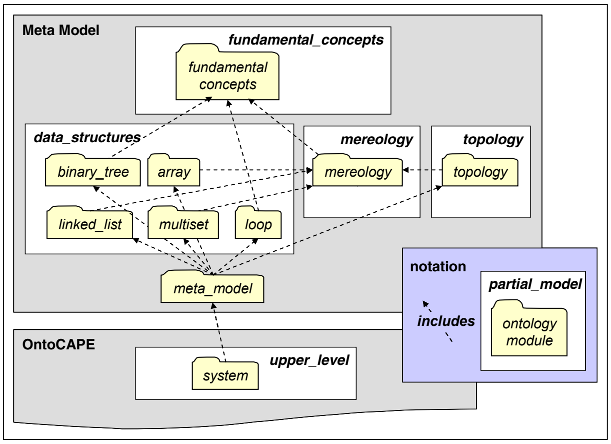

Fig. 3.3: Relations between the modules of the Meta Model and those of OntoCAPE

3.3.2 Data Structures

The partial model **data_structures** provides design patterns for the representation of the following structures, which frequently occur within ontologies.

– The module *array* establishes a pattern for representing an ordered collection of elements. The elements are ordered by an index, which specifies the position of an element within the array through a consecutive sequence of integer values. A particular element can be accessed via its respective index value. The array pattern is applied several times in OntoCAPE, e.g., for the conceptualization of vector quantities (cf. Sect. 4.5.3).

– Similar to an array, a *linked list* is a sequentially ordered collection of elements. The position of an element is defined by pointing to the next (and optionally also to the previous) element in the list. The pattern for linked lists is, for example, utilized to represent the version history of a document (cf. Sect. 4.5.4).

– A *multiset* differs from an ordinary set in that there may be multiple appearances of the same element (e.g., the multiset {a, b, b, b, c, c} has three appearances of element b). The corresponding design pattern provides a shorthand for representing such multisets; to this end, each element is assigned a multiplicity, which indicates the number of its appearances in the multiset (e.g., in the above multiset, element b is assigned a multiplicity of 3). Amongst others, this pattern is used to conceptualize reaction stoichiometry in OntoCAPE (cf. Sect. 4.5.2).

- A *binary tree* is a tree-like structure that is formed by a set of linked nodes. A node can have zero, one, or two child nodes, which are clearly identified as either the left or the right child node. In OntoCAPE, the pattern for binary trees is particularly utilized to represent mathematical equations (cf. Sect. 4.5.1).
- The design pattern *loop*[20] allows for a compact representation of repetitive structures, the elements of which differ in a systematic manner. Instead of enumerating such structures explicitly, the design pattern models only the first (and optionally also the last) element, the systematic change from one element to the next, and the total number of repetitions. A typical application of this pattern is the representation of mathematical models, which are composed of submodels of the same type (e.g., the tray-by-tray model of a distillation column; cf. Sect. 4.5.5).

3.3.3 Mereology

The partial model **mereology** establishes a theory for describing the relations between parts and wholes.
It follows common best-practice guidelines and takes up an idea from UML to distinguish between aggregation and composition. To that end, aggregation is the binary relation that exists between an aggregate (or whole) and one of its parts. A part may be part of more than one aggregate, i.e., it may be shared by several aggregates. A part can exist independently from the aggregate. Furthermore, a composition is identified as a special type of an aggregation relation, which exists between a composite object and its parts. These parts are non-shareable, i.e., they cannot be part of more than one composite object. If the composite object ceases to exist, its parts cease to exist, as well. For a more comprehensive description refer to Sect. 4.3.

[20] The name 'loop' is chosen because the syntax used to represent the loop pattern is similar to that of a 'for loop' in a programming language.

3.3.4 Topology

The partial model **topology** establishes a theory for describing topological relations between distributed entities.
The most fundamental concept of module *topology* is the relation isConnectedTo, which denotes the connectivity between objects. The relation is declared to be both symmetric and transitive. A key aspect of our topological theory is to keep mereological and topological relations strictly apart: Only the former or the latter relation can be applied between individuals. This approach, which has been adopted from Borst et al. (1997), enables the formulation of a compact but sufficient theory of mereotopology; theories that do not make this assumption require the definition of additional concepts like overlap, boundary, interior and exterior, etc., which can be avoided here. Please refer to Sect. 4.4 for a more detailed description.

3.4 The Upper Layer

The Upper Layer of OntoCAPE contains only a single partial model, called **upper_level**, which serves the function of a top-level ontology (cf. Sect. 2.6). Thus, the **upper_level** introduces a number of key concepts, which are specialized and refined on the lower layers. Moreover, it establishes the principles of *general systems theory*[21] and *systems engineering*[22], according to which the ontology is organized. The explicit representation of these principles, on the one hand, imparts an overview on the design of OntoCAPE, which helps a user to find his/her way around the ontology; on the other hand, it provides some guidance for extending or refining the ontology.
The concepts introduced by the **upper_level** are generic in the sense that they are applicable to different domains; thus, the partial model resembles the Meta Model in this respect. Yet unlike the Meta Model concepts, the concepts of the **upper_level** are intended for direct use and will be passed on to the domain-specific parts of OntoCAPE.

[21] General systems theory is an interdisciplinary field that studies the structure and properties of systems.

[22] Systems engineering can be viewed as the application of engineering techniques to the engineering of systems, as well as the application of a systems approach to engineering efforts (Thomé 1993).

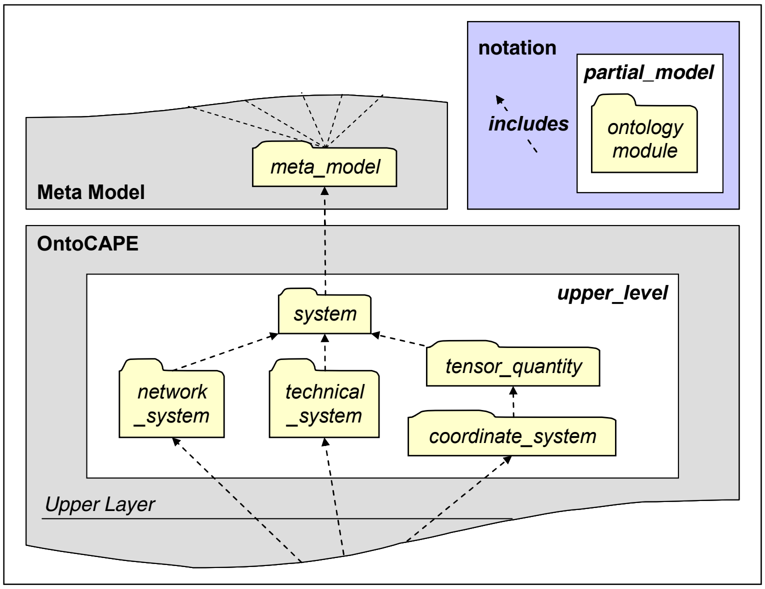

Fig. 3.4: The partial model **upper_level**

The **upper_level** partial model comprises five ontology modules (cf. Fig. 3.4). The module *system* is the most fundamental one of these. It establishes the constitutive systems-theoretical and physicochemical primitives, such as *system*, *subsystem*, *property*, *physical quantity*, *physical dimension*, etc., and specifies their interrelations. It also introduces the concept of an *aspect system* (cf. Sect. 5.1.7), which yields an abstraction of a system with respect to a particular viewpoint and thus allows partitioning a complex system into manageable parts.
As indicated in Fig. 3.4, the *system* module is located at the top of the inclusion hierarchy; it may import the ontology modules of the Meta Model, provided that such an import is desired (cf. the discussion in Chapt. 4). The remaining modules of the **upper_level** complement the *system* module:

– The ontology module *network_system* introduces a structured representation for complex systems, which is applicable to such different domains as biology, sociology, and engineering. To this end, the system is modeled as a network – that is, as a modular structure which "is determined on hierarchical ordered levels by coupling of components and linking elements" (Gilles 1998).
– The ontology module *technical_system* introduces the concept of a *technical system*, which represents a system that has been developed through an engineering design process. For a comprehensive description of a *technical system*, five designated viewpoints are of major importance (Bayer 2003): the system *requirements*, the *function* of the system, its *realization*, the

behavior of the system, and the *performance* of the system. These viewpoints are explicitly modeled as *aspect systems*.
- The module *tensor_quantity* extends the concept of a *physical quantity* (which is restricted to scalars in module *system*) to vectors and higher-order tensors.
- Finally, module *coordinate_system* introduces the concept of a *coordinate system*, which serves as a frame of reference for the observation of system properties.

For a more extensive description of the **upper_level**, refer to Chap. 5.

3.5 The Conceptual Layer

The Conceptual Layer constitutes the core of OntoCAPE. It is structured into four large partial models, which jointly conceptualize the CAPE domain.

- The partial model **material** provides an abstract description of materials and material behavior.
- The **chemical_process_system** conceptualizes all those notions that are directly related to materials processing and plant operating, such as plant equipment, process flowsheets, control systems, etc.
- The partial model **mathematical_model** defines terms required for a description of mathematical models and model building.
- Finally, **supporting_concepts** supplies auxiliary concepts, such as commonly used physical dimensions, SI units, mathematical expressions, etc. These concepts do not directly belong to the CAPE domain, but support the specification of domain concepts.

In the following, we will give a very brief overview on the structure and contents of these four partial models.

3.5.1 Supporting Concepts

The partial model **supporting_concepts** defines basic notions, such as spatial and temporal coordinate systems, geometrical concepts, mathematical relations, as well as commonly used physical dimensions and SI-units. The concepts defined in this partial model do not belong to the core of the CAPE domain, but are merely utilized by the other partial models of OntoCAPE for defining domain concepts. For that reason, **supporting_concepts** is only rudimentarily developed, as it is not the objective of OntoCAPE to conceptualize areas that are beyond the scope of the CAPE domain. For example, the partial model **mathematical_relation** does not aim at establishing a full-fledged algebraic theory, as does the EngMath ontology (Gruber and Olsen 1994; cf. Chap. 11); rather, it provides a simple but pragmatic mechanism for the representation of mathematical relations, which serves the needs of the other partial models of OntoCAPE[23].

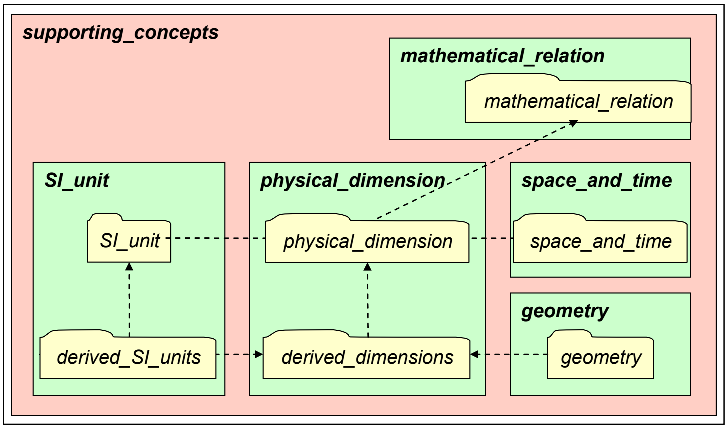

Fig. 3.5: Overview on partial model **supporting_concepts**

As depicted in Fig. 3.5, **supporting_concepts** comprises five subordinate partial models, which are **mathematical_relation**, **physical_dimension**, **SI_unit**, **space_and_time**, and **geometry**.

- Partial model **mathematical_relation** introduces concepts to represent mathematical expressions. However, it is not the objective of this module to describe mathematical models – this is the responsibility of the partial model **mathematical_model** (cf. Chap. 9). Rather, it provides auxiliary concepts, which are utilized by other ontology modules (e.g., for the definition of units).
- Partial model **physical_dimension** comprises two modules. The main module, *physical_dimension*, defines a set of base dimensions and establishes the proceedings to derive further physical dimensions from these base dimensions. It is extended by the module *derived_dimensions*, which introduces a number of frequently used derived dimensions.
- Partial model **SI_unit** comprises the modules *SI_unit* and *derived_SI_units*. The former module introduces the base units of the SI system and establishes a mechanism to derive further units from these. The latter module utilizes this mechanism to define a number of frequently used SI units.

[23] Note that this choice has been made deliberately since a full-fledged algebraic theory is not required in this context. It would only complicate matters and reduce the efficiency (cf. Sect. 10.6) of the ontology.

- Refining the concept of a *coordinate system* introduced on the Upper Layer, the partial model **space_and_time** establishes common types of spatial and temporal coordinate systems. Moreover, it provides concepts for the representation of spatial and temporal points as well as periods of time.
- Finally, the partial model **geometry** provides the concepts for describing the shapes and main dimensions of simple geometric figures.

More detailed descriptions of the above partial models can be found in Chap. 6.

3.5.2 Material

The partial model **material** provides concepts that enable an abstract description of matter. In this context, 'matter' refers to "anything that has mass and occupies space" (Gold et al 1987). The partial model considers only those characteristics of matter that are *independent of a material's concrete occurrence*[24] in time and space; complementary, the partial model **CPS_behavior** (cf. Sect. 8.6) describes the behavior of materials in the concrete setting of a chemical process.
Material comprises two partial models, called substance and phase_system (cf. Fig. 3.6).

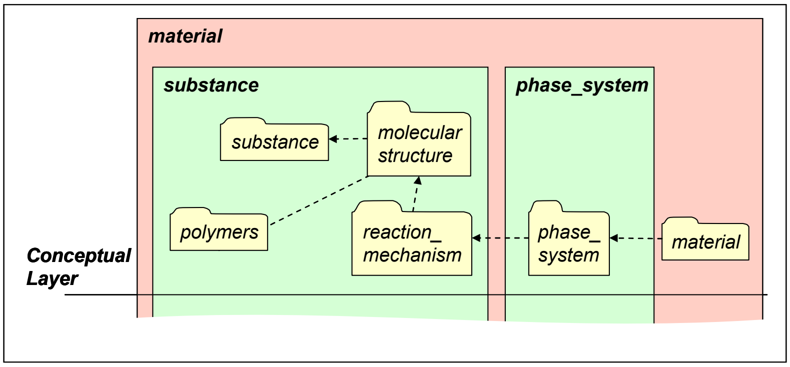

Fig. 3.6: Partial model **material** on the Conceptual Layer

On the Conceptual Layer, **substance** includes the following modules:

- *Substance* is the main module of partial model substance. It provides essential concepts for the description of pure substances and mixtures at the macroscopic scale.

[24] By 'concrete occurrence', we mean the actual spatiotemporal setting – for example, the manufacturing of some material in a chemical plant or its usage as a construction material. For details on this issue, we refer to Morbach et al. (2008j).

– The module *molecular_structure* is concerned with the characterization of pure substances at the atomic scale.
– *Polymers* completes *molecular_structure* with concepts for the description of macromolecular structures.

Finally, *reaction_mechanism* allows representing the mechanism and the stoichiometry of chemical reactions.The partial model **phase_system** comprises a single ontology module, named *phase_system*; it describes the thermodynamic behavior of *materials* subject to a certain *physical context.*
Refer to Chap. 7 for an extensive description of the partial model **material**.

3.5.3 Chemical Process System

Partial model **chemical_process_system** is concerned with the conceptualization of chemical processes and chemical plants. Its key concept is the *chemical process system*, which is a special type of a *technical system* (cf. Sect. 5.3) designed for the production of chemical compounds. The *chemical process system* is characterized from four distinct viewpoints; these viewpoints are modeled as *aspect systems* and are represented in separate partial models (cf. Fig. 3.7):

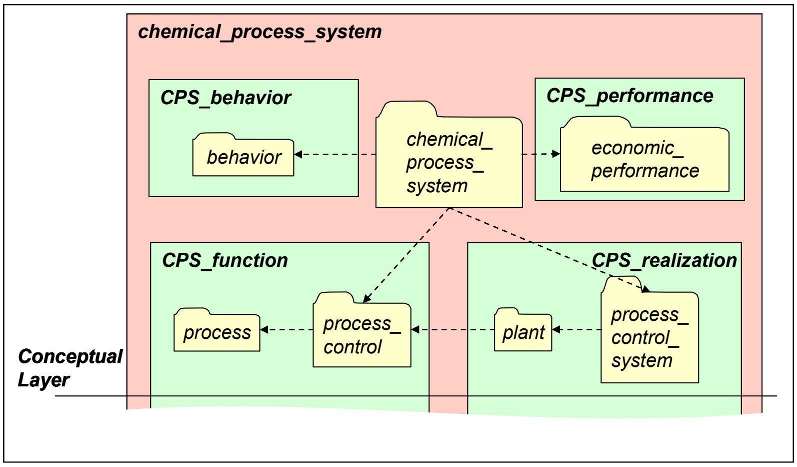

Fig. 3.7: Partial model **chemical_process_system** on the Conceptual Layer

– **CPS_function** enables a functional specification of *chemical process systems*. Its main module *process* describes chemical processes by an approach called ‘the phase model of production’ (Polke 1994; Bayer 2003). This formalism, which may be transformed into a process flow diagram or a state-task network, is suitable for describing both continuous processes and batch processes. The supplementary module *process_control* allows the specification of control strategies in terms of function blocks and control loops.

- **CPS_realization** describes the technical realization of a *chemical process system*. The main module *plant* conceptualizes the major types of processing equipment and machinery as well as the connectivity of the equipments via pipes and fittings. Complementarily, the module *process_control_sys-tem* defines the basic components required for process automation, such as measuring instruments, signal lines, and controllers.
- **CPS_behavior** describes the behavior of the *chemical process system*. The module enables both a qualitative and a quantitative-empirical characterization of chemical process behavior – the former is achieved by indicating the prevailing physicochemical phenomena, the latter by indicating the (measured or projected) *values* of the system *properties*.

Finally, the **CPS_performance** evaluates the economic performance of the *chemical process system* in terms of investment and production costs.
A more extensive description of the partial model is given in Chap. 8.

3.5.4 Mathematical Model

The partial model **mathematical_model** is concerned with the description of mathematical models; CapeML (von Wedel 2002) has been used as an important source. Fig. 3.8 gives an overview on **mathematical_model** on the Conceptual Layer. The main module, *mathematical_model*, introduces the basics concepts for mathematical modeling, including model variables and items pertaining to submodels and their connections: A *mathematical model* is conceptualized as a special type of *system*, the properties of which are reflected by its *model variables*.

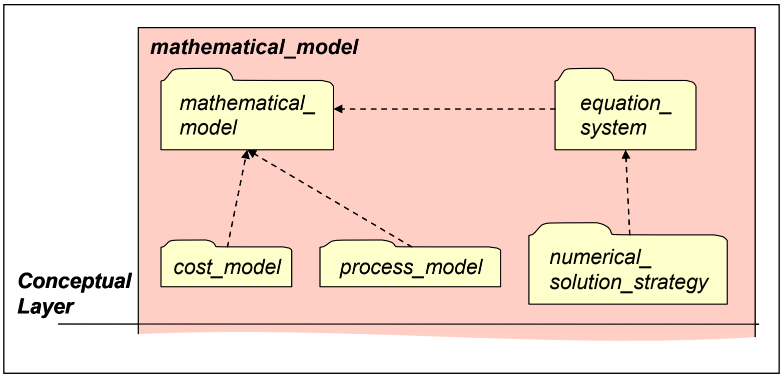

Fig. 3.8: Partial model **mathematical_model** on the Conceptual Layer

The ontology module *equation_system* further specifies the characteristics of the model equations that constitute a *mathematical model*. Based on these characteristics, an appropriate numerical solver can be selected, which is the concern of the ontology module *numerical_solution_strategy*. The modules *process_model* and *cost_model* describe two particular types of mathematical models: *process models* model the behavior of *chemical process systems* and *materials*, while *cost models* predict the costs of *chemical process systems*.
Further information about the partial model can be found in Chap. 9.

3.6 The Application Layers

The Application-Oriented Layer and the Application-Specific Layer (collectively referred to as 'application layers') extend the ontology towards concrete applications. As explained in Sect. 3.1.1, the major difference between the two application layers is that the concepts of the Application-Oriented Layer are relevant for a whole range of applications, whereas the concepts of the Application-Specific Layer are rather intended for one particular application.
Typically, the terms introduced on the application layers are instances or specializations of terms introduced on the Conceptual Layer. Accordingly, the modules located on the application layers do not open up new subject areas, but complete and refine existing partial models. These modules can thus be considered as application (domain) ontologies: As explained in Sect. 2.6, an application ontology holds application-specific, but state-independent information.
Over the lifecycle of OntoCAPE, new modules are expected to appear on the application layers with each new application encountered. For the applications realized so far (cf. Chapt. 12), the following extensions have been developed:

- Partial model **material** is extended by modules holding instance data about chemical elements, pure substances, and technical polymers. Further modules provide classification schemata for substance classes (such as alcohols or esters) and for types of chemical reactions (such as esterification or hydrohalogenation).
- Partial model **chemical process system** is refined by a number of classification schemata, which introduce special types of unit operations, of plant equipment (i.e., apparatuses, machines, and fixtures), of measuring and control instruments as well as of controller types. Furthermore, the behaviors of typical *process units*[25] are modeled in an application-oriented extension of the partial model **CPS_behavior**.

[25] The term *process unit* denotes an elementary subsystem of a *chemical process system*, such as a reactor, a heat exchanger, or a distillation column.

- Within the partial model **mathematical_model**, a number of modules are added to complement the module *process_model*. For sake of illustration, these add-ons will be discussed in more detail below.

As explained in Chap. 9, the module *process_model* enables the definition of specialized *process models*, which model the behavior of *chemical process systems* and *materials*. Typically, a *process model* is composed of *submodels*, of which *laws* and *property models* are important subtypes. A *law* constitutes the mathematical representation of a scientific law, such as the law of energy conservation. A *property model*, in turn, represents a mathematical correlation for the computation of a single physical property. Typical property models would be vapor pressure correlations or activity coefficient models.
The terms *law* and *property model* have already been defined on the Conceptual Layer in the module *process_model*. On the Application-Oriented Layer, *process_model* is extended by the ontology modules *laws* and *property_models* (cf. Fig. 3.9). The former module establishes models for a number of physical laws that are common in the context of chemical engineering. These laws represent

- the conservation of energy, mass, and momentum;
- thermal, mechanical, or phase equilibrium, as well as reaction equilibrium;
- non-equilibrium transport phenomena, such as diffusion.

Similarly, the *property_models* module defines special types of *property models*, which can be categorized into three major classes:

- *Chemical kinetics property models* specify how to calculate the rate coefficients of homogenous or heterogeneous reactions.
- *Phase interface transport property models* provide correlations for computing certain *phase interface transport properties*.
- *Thermodynamic models* indicate the correlations between certain *intensive thermodynamics state variables* and *intra-phase transport properties*.

Module *process_model* is furthermore extended by module *process_unit_models*, which defines a number of specialized *process models* for customary *process units*. Examples of such *process unit models* are a *CSTR model* or a *tray-by-tray distillation column model*. Typically, a *process unit model* includes one or several *laws* (e.g., a *tray-by-tray distillation column model* includes a *phase equilibrium law*); therefore, module *laws* is imported by *process_unit_models*.
The above described modules are all located on the Application-Oriented Layer. By contrast, the module *aspen_pus_model* constitutes a further extension of module *process_model* on the Application-Specific Layer. It has been developed for an application in knowledge management, which is presented in Sect. 12.2. Basically, the module provides concepts for the semantic annotation of simulation documents in the format of the simulation software *Aspen Plus* (AspenTech 2008). Such documents contain the specification of an *Aspen Plus model*, which is a spe-

cial type of a *process model*. The average *Aspen Plus model* includes at least one *property model* as well as a number of *process unit models*. The latter are *process unit models* of particular types, which are provided by the model library of the simulation software. To give an example, the Aspen Plus model library holds a model named *RadFrac*, which is a particular implementation of a *tray-by-tray distillation column model*. In contrast to the more general *tray-by-tray distillation column model*, the *RadFrac* model has a number of preassigned properties – such as being a closed-form model of nonlinear algebraic type.

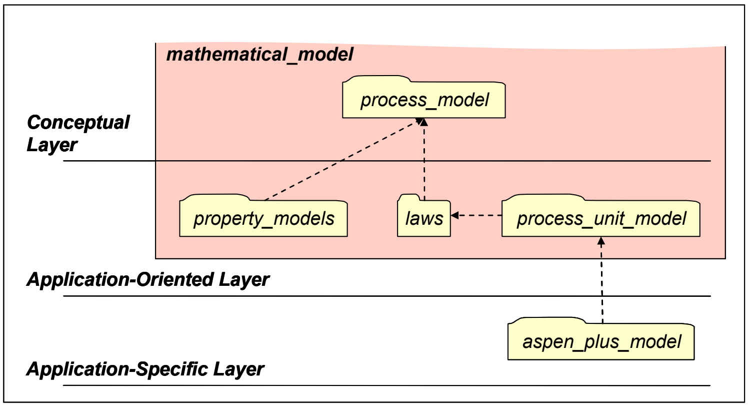

Fig. 3.9: Extension of the module *process_model* on the application layers

3.7 References

AspenTech (2008) *Aspen Plus*. Online available at http://www.aspentech.com/products/aspen-plus.cfm. Accessed June 2008.

Bayer B (2003) *Conceptual Information Modeling for Computer Aided Support of Chemical Process Design*. Fortschritt-Berichte VDI: Reihe 3, Nr. 787. VDI-Verlag, Düsseldorf.

Bernaras A, Laresgoiti I, Corera J (1996) Building and reusing ontologies for electrical network applications. In: Wahlster W (ed.): *ECAI 1996 – Proceedings of the 12th European Conference on Artificial Intelligence*. Wiley, Chichester:298–302.

Borst P, Akkermans JM, Top JL (1997) Engineering ontologies. *Int. J. Hum Comput Stud.* **46**:365–406.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Bray T, Hollander D, Layman A, Tobin R, eds. (2006a) *Namespaces in XML 1.0 (Second Edition).* W3C Recommendation, 16 August 2006. Online available at http://www.w3.org/TR/xml-names. Accessed September 2008.

Chandrasekaran B, Johnson TR (1993) Generic tasks and task structures: history, critique and new directions. In: David JM, Krivine JP, Simmons R (eds.): *Second Generation Expert Systems*. Springer, New York:232–272.

Gilles ED (1998) Network theory for chemical processes. *Chem. Eng. Technol.* **21** (8):121–132.

GNU Project (2007) The GNU General Public Licence. Online available at http://www.gnu.org/copyleft/gpl.html. Accessed December 2007.

Gold V, Loening KL, McNaught AD, Sehmi P (1987) Compendium of Chemical Terminology. Blackwell, Oxford.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/pa-pers/engmath.html. Accessed September 2007.

Heflin J, Hendler J (2000) Dynamic ontologies on the web. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000). AAAI Press, Menlo Park, CA:443–449.

Jarrar M, Meersman R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International Conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine* SSGRR2002.

Morbach J, Wiesner A, Marquardt W (2008f) *OntoCAPE 2.0 – The Meta Model.* Technical Report (LPT-2008-24), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-24.

Morbach J, Bayer B, Wiesner A, Yang A, Marquardt W (2008g) OntoCAPE 2.0 – The Upper Level. Technical Report (LPT-2008-25), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-25.

Morbach J, Yang A, Wiesner A, Marquardt W (2008h) *OntoCAPE 2.0 – Supporting Concepts*. Technical Report (LPT-2008-26), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-26.

Morbach J, Yang A, Marquardt W (2008i) *OntoCAPE 2.0 – Materials*. Technical Report (LPT-2008-27), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-27.

Morbach J, Yang A, Marquardt W (2008j) *OntoCAPE 2.0 – Mathematical Models*. Technical Report (LPT-2008-28), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-28.

Noy NF, Klein M (2004) Ontology evolution: not the same as schema evolution. *Knowl. Inform. Syst.* **6**:428–440.

Pinto HS, Gomez-Perez A, Martins JP (1999) Some issues on ontology integration. In: *Proceedings of the IJCAI'99 Workshop on Ontologies and Problem Solving Methods*.

Polke M, ed. (1994) *Process Control Engineering*. VCH, Weinheim.

Pure-Systems (2008) *Pure::Variants*. Online available at http://www.pure-systems.com. Accessed September 2008.

Rector A (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Genari J (ed.): *Knowledge Capture 2003*. ACM Press:121–128.

Russ T, Valente A, MacGregor R, Swartout W (1999) Practical experiences in trading off ontology usability and reusability. In: *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Stuckenschmidt H, Klein M (2003) Integrity and change in modular ontologies. In: Gottlob G, Walsh T (eds.): *IJCAI-03 – Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann:900–905.

Thomé B, ed. (1993) *Systems Engineering: Principles and Practice of Computer-based Systems Engineering*. John Wiley, New York.

Visser PRS, Cui Z (1998) Heterogeneous ontology structures for distributed architectures. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:112–119.

von Wedel L (2002) *CapeML – A Model Exchange Language for Chemical Process Modeling*. Technical Report (LPT-2002-16), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Wiesner A, Morbach J, Bayer B, Yang A, Marquardt W (2008a) *OntoCAPE 2.0 – Chemical Process System*. Technical Report (LPT-2008-29), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-26.

4 Meta Model

4.1 Introduction

As explained in Sect. 2.6, a meta ontology constitutes the most generic type of ontologies within the classification framework. A meta ontology is usually defined on top of some target ontology, which may be a domain ontology as well as a task ontology or even an upper-level ontology. The meta ontology establishes the fundamental modeling concepts to be used in the target ontology as well as the rules for their proper application. Thus, it explicitly represents the underlying design principles of the target ontology.
Over the lifecycle of the target ontology, a meta ontology serves different functions:

– In the development phase, the meta ontology guides the design and organization of the target ontology: It establishes general standards for ontology engineering, which serve as a sort of style guide for the development team, thus ensuring a consistent way of knowledge representation across the target ontology (cf. the discussion of the design principle of homogeneity in Sect. 10.3.2).
– When it comes to (re)using the target ontology, the meta ontology has three different benefits: Firstly, it provides guidance for extending the ontology in scope by offering templates for recurring design problems. Secondly, when the target ontology is tailored to a particular application, it ensures compliance with the overall design principles. Thirdly, by examining the rather concise meta ontology, new users can quickly familiarize themselves with the modeling style of the more complex target ontology. Consequently, the users can quickly evaluate if the target ontology is generally compatible with the requirements of the envisioned application. This is of special importance, since assessing the suitability of an ontology for a given application is one of the most time-consuming tasks in ontology reuse (Pâslaru-Bontaş 2007).

The *Meta Model*, which is defined on top of OntoCAPE, constitutes a meta ontology at the logical metalevel (cf. Sect. 2.6). Although the Meta Model has been developed specifically for OntoCAPE, it is in fact domain-independent and can thus be reused to guide the construction of other OWL-based ontologies. Currently, the Meta Model constitutes the basic framework of three further ontologies, named *Document Model* (Morbach et al. 2008), *Process Ontology* (Eggersmann et al. 2008), and *Decision Ontology* (Theißen and Marquardt 2008).

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1_4,

As shown in Fig. 4.1, it is partitioned into the partial models[26] **fundamental_concepts**, **mereology**, **topology**, and **data_structures**. While both **mereology** and **topology** contain only a single ontology module[27], **data_structures** comprises five: *array*, *linked_list*, *multiset*, *binary_tree*, and *loop*. The module *meta_model* includes[28] all these ontology modules, thus assembling the ontological definitions of the Meta Model. The module *meta_model* is, in turn, included by the top-level module of the target ontology (shown here is the module *system*, which resides on the Upper Layer of OntoCAPE). That way, the concepts defined in the Meta Model are available in the target ontology.

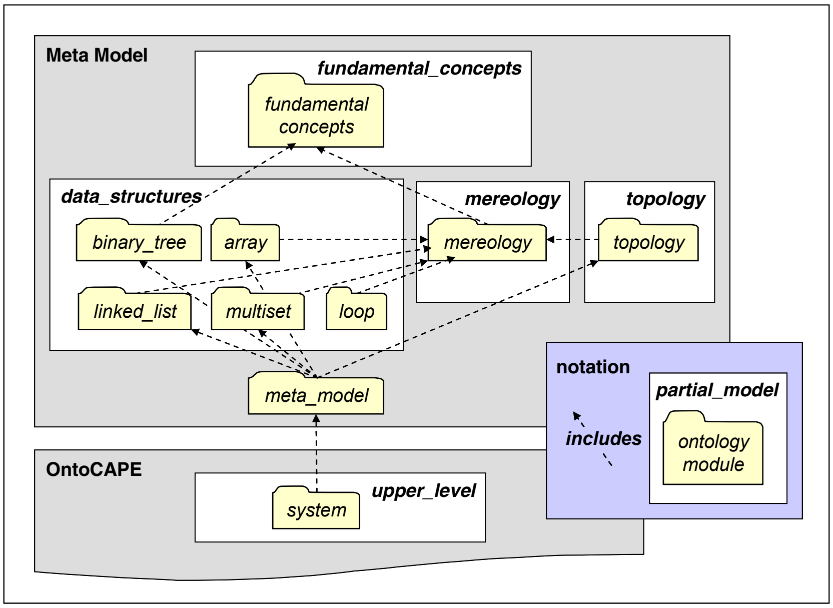

Fig. 4.1: Relations between the ontology modules of the Meta Model and those of OntoCAPE

The Meta Model is not a genuine part of the target ontology. Rather, its function is (a) to explicitly represent the underlying design principles and (b) to establish some common standards for the design and organization of the target ontology.

[26] Ontology modules assemble a number of classes that cover a common topic as well as the relations describing the interactions between the classes and the constraints defined on them.

[27] Ontology modules that address closely related topics are grouped into partial models. The partial models constitute a coarse categorization of the domain.

[28] Inclusion means that if module A includes module B, the ontological assertions provided by B are included in A. Inclusion is transitive, that is, if module B includes another module C, the ontological assertions specified in C will also be valid in A.

Thus, the Meta Model supports ontology engineering and ensures a consistent modeling style across the target ontology. These goals are achieved by means of two different mechanisms: the introduction of *fundamental concepts*, and the definition of *design patterns*.

Fundamental concepts are fundamental classes and relations from which all the root terms of the target ontology can be derived (either directly or indirectly). By linking a root term of the target ontology to a fundamental concept, its role within the ontology is characterized. That way, a user or a software program is advised how to properly treat that particular root term and the classes or relations derived from it: For instance, all classes in the target ontology that are derived from the fundamental concept '*relation class*' are auxiliary constructs for the representation of n-ary relations. Since instances of such classes do not need to be given meaningful names (cf. Noy and Rector 2006), a user or an intelligent software program can conclude that such instances can be labeled automatically, according to some standardized naming convention.

Conceptually, the linkage between the ontological terms of the Meta Model and those of the target ontology should be established by means of instantiation. However, while the OWL modeling language supports such metamodeling (i.e., instantiation across multiple levels) in principle, it is at the cost of loosing scalability and compatibility with DL reasoners (Smith et al. 2004). Therefore, it is not advisable to interlink the Meta Model and the target ontology via instantiation. Hence, the linkage between OntoCAPE and the Meta Model is currently realized via specialization.

A design pattern[29] is a template formed by a set of classes, interconnecting relations, and constraining axioms; it establishes a best-practice solution to a recurring design problem. That way, patterns encourage a consistent, uniform design throughout the target ontology. A typical example is the representation of mereologic relations (part-whole relations): A design pattern defines a standard way of modeling this relation type, which is adopted by all ontology modules of the target ontology.

It is worthwhile noting that the design patterns of the Meta Model are implementation-dependent; that is, they constitute a best-practice solution only for an ontology that is *represented in OWL* and *processed by a customary DL reasoner*. For instance, the abovementioned mereology pattern states how to implement part-whole relations in OWL such that they efficiently scale for large amounts of instance data. Yet if the part-whole relations were implemented in a different modeling language, or if the ontology was processed by a non-standard reasoner, the mereology pattern might not constitute the best possible solution.

To apply a design pattern in the target ontology, we have adopted a rather pragmatic approach that was suggested by Clark et al. (2000): The classes, relations,

[29] Design patterns are popular in software engineering (e.g., Gamma et al. 1995), where they specify general solutions for recurring problems. In ontology engineering, the term 'knowledge pattern' (Clark et al. 2000) is sometimes used instead.

and axioms that constitute the design pattern in the Meta Model are simply redefined in the target ontology. Practically, this is realized by (1) copying the axiomatic definitions of the design pattern into the target ontology and (2) renaming the non-logical symbols within these expressions (i.e., the classes and relations); additionally, the duplicated classes and relations may be linked to their respective originals in the Meta Model, but this is not mandatory (cf. the discussion in the subsequent paragraph). The advantage of this approach is its flexibility: Often, only a selected part of a theory is to be transferred (i.e., there may be symbols in the pattern that have no counterpart in the target ontology) – either because only the transferred part is needed in the target ontology, or because the omitted part is to be implemented differently from the Meta Model. For this purpose, the transfer of the design pattern via rigorous specialization (or instantiation) would not be flexible enough, as it would call for copying the entire pattern in an "all-or-nothing" fashion. By contrast, the selected approach allows for deviations and variants. Clark et al. (2000) stress that this is architecturally significant, as well, since the approach supports a better modularization of the target ontology.

While the Meta Model has proven to be highly useful during the design of the target ontology and its refinement to a knowledge base, it becomes less relevant once the refined ontology is actually used as a knowledge base of some application; in some cases it might even be harmful, as the additional, abstract concepts of the Meta Model could confuse the user. Thus, the interconnectivity between target ontology and the Meta Model should be kept at a minimum, such that the two ontologies can be separated easily if desired. Therefore, the classes and relations defined in Meta Model are not to be used directly within the target ontology; rather, they are redefined by copying the respective concepts in the target ontology, as explained above. The duplicates may subsequently be linked to the originals in the Meta Model[30]. That way, only the links to the Meta Model need to be disconnected if a stand-alone usage of the target ontology is desired. Particularly for relations, the principle of overloading[31] is often applied: that is, the relation in the target ontology receives the same name as the original relation in the Meta Model. That way, a relation with the same name can be implemented in different ontology modules, however each time possibly with a different range and domain, and thus with a different semantics.

The Meta Model is completely implemented in OWL. The ontology modules are realized through namespaces, each of which is stored in a single OWL file. The partial models are implemented as directories. A directory may contain some additional OWL files, the names of which start with the prefix "*example_*"; these files illustrate the usage of the Meta Model by means of exemplary applications.

[30] Linking a duplicate to the original through specialization has proven valuable during ontology design, since it allows checking the consistency of the duplicate against the original by means of a reasoner.

[31] The idea of overloading originates from computer science; originally, it means that multiple functions, taking different types of input, can be defined with the same name.

The reasoner RacerPro (Racer Systems 2006) has been used to validate the consistency of the Meta Model.
The remainder of this chapter is organized as follows: each ontology module is first described in natural language and by means of UML-like class diagrams, which show the main interrelations between classes and relations, including their hierarchical organization. Some application examples may be provided. Subsequently, the usage of the ontology module is explained, and some competency questions[32] are presented that characterize the functionality of the ontology module. Finally, the individual concepts (classes, relations, attributes and instances) of the respective ontology module are described in natural language. For defined classes a definition in natural language is provided to explain the nature of the class beyond the often brief formal definition.

4.2 Fundamental Concepts

The ontology module *fundamental concepts* forms the basis of the Meta Model. It introduces *meta root concepts* and their refinements. A *root concept* is a class or a relation without ancestors. Accordingly, *meta root concepts* are the root classes and relations in the Meta Model. They form the topmost layer of the concept hierarchy; all other classes and relations – in the Meta Model as well as in the target ontology – can be derived from the meta root terms by specialization. As can be seen from Fig. 4.2, three root classes are defined in the Meta Model: *object*, *relation class*, and *feature space*.
Object is a generic class that subsumes all "self-standing" (Rector 2003) entities – whether physical or abstract – that exist in an application domain. In conjunction with the *object* class, the root relation interObjectRelation is introduced, which subsumes all types of binary relations that exist between *objects*.
An *object* can be characterized by means of descriptive features. A *feature space* defines the range of values that a feature can take (Rector 2005). Three specializations of *feature space* are distinguished, which reflect different ways to define the values of a particular feature: A *value partition* describes the feature values by partitioning a class into disjoint subclasses. In contrast, a *value set* defines the values as an enumeration of individuals. While a *value set* has a fixed number of individuals, the number of individuals is not predetermined in a *non-exhaustive value set*.

[32] The formulation of competency questions forms part of the methodology for ontology engineering that was first suggested by Grüninger and Fox (1995) and later explicated in detail by Uschold and Grüninger (1996). Informal competency questions are questions in natural language that specify the requirements for the ontology to be developed, thus determining its scope. Once the ontology is implemented in a formal language, the competency questions are formalized in a machine-interpretable language such that they can be evaluated by a reasoner. By running the formal competency questions against the ontology (or rather against a set of test data instantiated from the ontology), it can be verified that the ontology complies with the specifications.

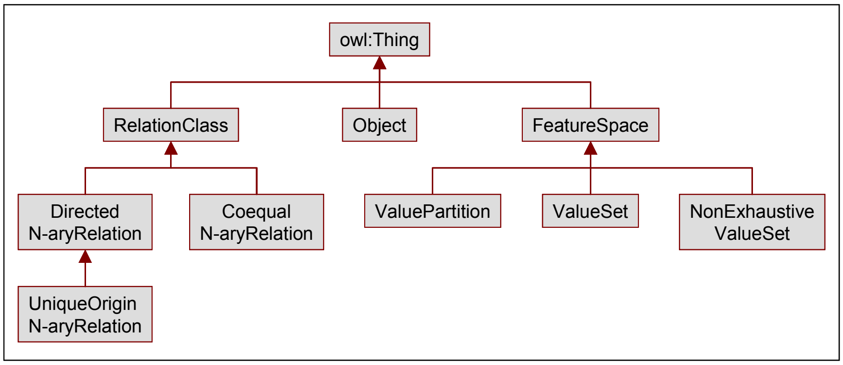

Fig. 4.2: Fundamental classes

A feature value (i.e., an instance of *feature space*) can be assigned to an *object* via the unidirectional object-featureRelation. Feature values are independent of a particular *object*; thus, a feature value may be assigned to different *objects*, as indicated in Fig. 4.3.

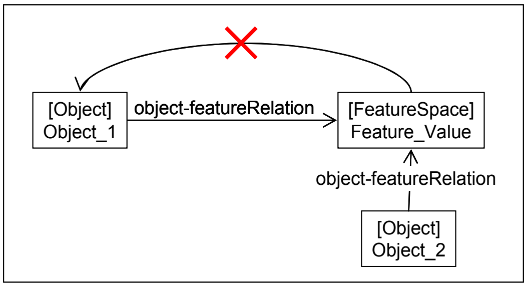

Fig. 4.3: A feature value may be assigned to different objects

Relations that refer from a feature value to an *object* are not permitted – such a relation would imply that the individual represents the feature of a particular *object*, i.e., the feature value would loose its independence. Therefore, a feature value cannot be the origin of an unidirectional object-feature relation, and there must not be any bidirectional relations between *object* and *feature space*, either.
The OWL language merely provides language primitives for *binary* relations; there is no predefined language element for an *n-ary* relation that could link three or more individuals. Also, binary relations cannot be characterized through attributes. To overcome these limitations, the concept of a *relation class* is introduced. A *relation class* may be used to

– represent n-ary relations between individuals of type *object* or *feature space*, and/or
– characterize a relation between two or more individuals by some additional attribute.

These two application cases are depicted in Fig. 4.4.

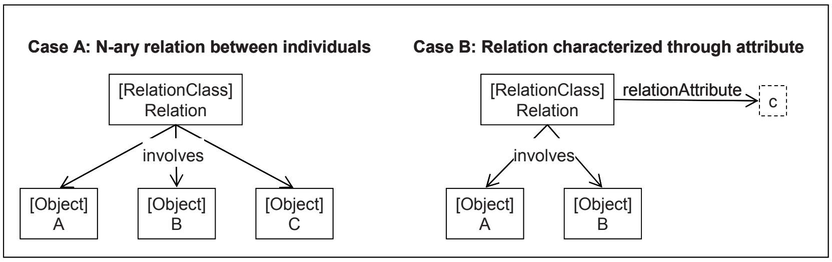

Fig. 4.4: Application cases for a *relation class*

Fig. 4.5 shows the design pattern that defines a *relation class*. A *relation class* involves at least one *object* and at least one other individual of type *object* or *feature space*. Moreover, it may be characterized by some relationAttributes. The *objects* involved in the n-ary relation can be explicitly identified via the inverse relations involvesObject and isInvovledInN-aryRelation.

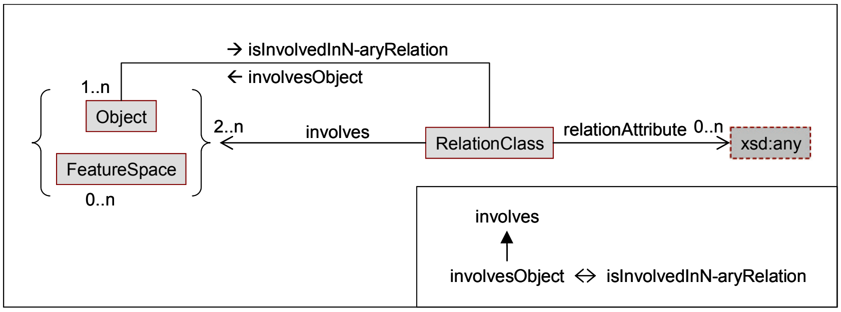

Fig. 4.5: Design pattern for a *relation class*

Two specializations of *relation class* are introduced:

– A *directed n-ary relation* describes an n-ary relation among some individuals of type *object* or *feature space* where at least one *object* is distinguished as the origin of the relation (Fig. 4.6).
– By contrast, a *coequal n-ary relation* describes an n-ary relation among some individuals where none of the individuals involved in the relation stands out as the origin of the relation.

The origin of a *directed n-ary relation* is identified by means of the inverse relations hasOrigin and isOriginOf.

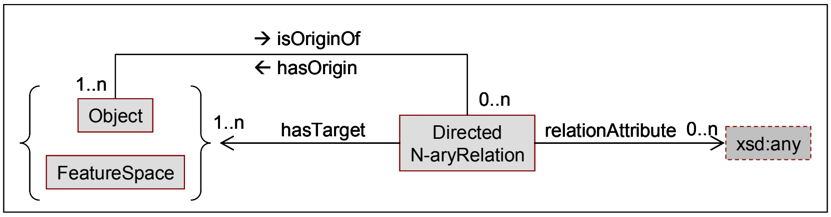

Fig. 4.6: Directed n-ary relation

The other involved individuals are denoted as targets of the n-ary relation through the hasTarget relation. The target objects can be explicitly identified via the inverse relations hasTargetObject and isTargetOf. The specialization hierarchy of these relations is displayed in Fig. 4.7.

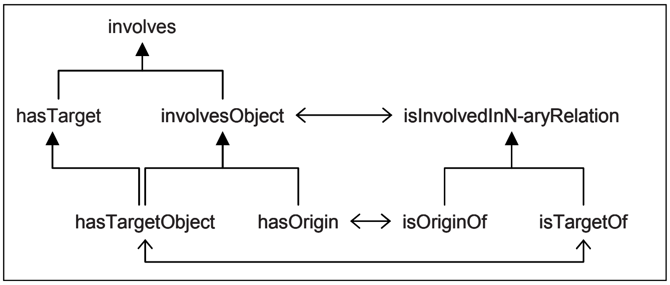

Fig. 4.7: Specialization hierarchy of the relations for the class *directed n-ary relation*

Fig. 4.8 gives an application sample of a *directed n-ary relation*. As can be seen in the figure, a *directed n-ary relation* may have more than one relation origin. The class *unique origin n-ary relation* is introduced to denote the special case of a *directed n-ary relation* which has exactly one relation origin.

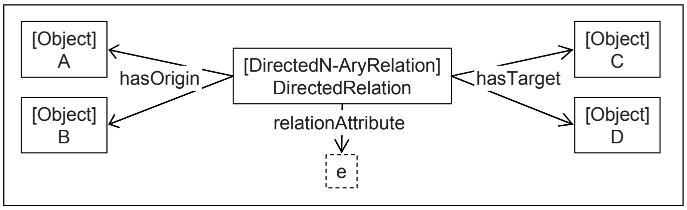

Fig. 4.8: Application sample of a *directed n-ary relation*

Practical applications require different *views* on the ontology. Comparable to a view on a relational database[33], an *ontology view* is a set of concepts (classes or

[33] A view of a relational database is a virtual or logical table that is composed of the result set of a pre-compiled query.

instance data) that is retrieved from the ontology as the result of a pre-defined query class.

To realize ontology views, Rector (2003) proposes to establish alternative axes of classification in the ontology, where each axis assembles concepts for a particular use. Generally, such axes can be implemented by means of multiple classification, as presented in Fig. 4.9 for the classification of *objects*: First, different *object types* (here: *Type_1*, *Type_2*, and *Type_3*) are introduced; then the actual *objects* (here: *A*, *B*, and *C*) are explicitly assigned to one or more of these *object types* through subclassing.

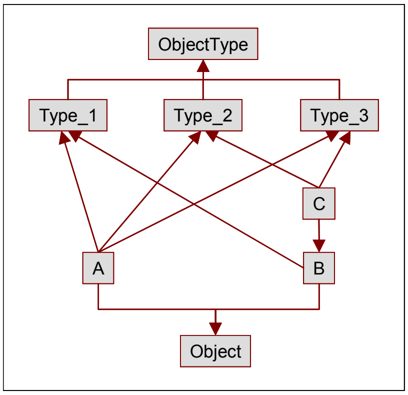

Fig. 4.9: Realization of ontology views by multiple classification

The problem with this approach is that complex *polyhierarchies* will evolve, which are hard to grasp for human users and thus difficult to manage and to maintain. Therefore, another approach is recommended here: Adopting the mechanism for *ontology normalization* suggested by Rector (2003), *objects* are explicitly classified along a single axis only. Specialization along this classification axis should preferably be based on the same (or progressively narrower) criteria, throughout. The classes introduced on this axis may be either primitive class (i.e., characterized through necessary conditions only) or defined class (i.e., characterized through necessary and sufficient conditions).

All further axes must be defined implicitly by (1) assigning *value types* to the *objects*, and (2) defining the *object types* as classes that are of a particular *value type* (cf. Fig. 4.10). That is, having an isOfType relation to a particular *value type* is a necessary and sufficient condition for an object being subsumed by a particular *object type*. Following this mechanism, a pattern evolves which is comparable to the one shown in Fig. 4.10. From this pattern, a polyhierarchy like the one presented in Fig. 4.9 can be automatically inferred by a reasoner.

Value types can either be subclasses or, as exemplarily shown in Fig. 4.10, instances of a *value type* class. Thus, *value types* can either be subclasses of *value partition* or (*non-exhaustive*) *value set*. They can again be organized in hierarchies.

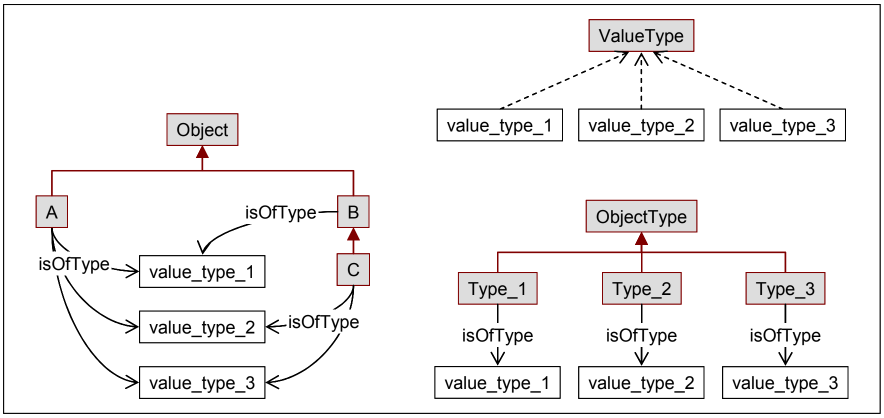

Fig. 4.10: Implicit classification through value types

4.2.1 Usage

The fundamental concepts introduced above are not intended to be used (i.e., instantiated) directly; rather, they serve the purpose of (a) organizing the derived classes and relations, and (b) characterizing their role within the ontology. By means of the latter, a user or a software program is advised how to properly treat that particular concept. For example, classes that are derived from the *relation class* are obviously auxiliary constructs for the representation of n-ary relations. Consequently, instances of such classes do not need to be given meaningful names (cf. Noy and Rector 2006). Instead, they may be labeled according to some standardized naming convention (a possible naming convention would be to use the identifier of the *relation class* and append an underscore ("_"), followed by a unique number, i.e., **<identifier of relation class>_<unique number>**). Thus, each time a class is identified as a specialization of *relation class*, the user (or an intelligent software program) can conclude that the instance should be labeled automatically, following the chosen naming convention.
Concerning the assignment of ontology views, the above classes (*object type*, *value type*, etc.) were introduced for explication only. They do not form part of the OWL implementation of the Meta Model, but should be introduced in the target ontology. Only the relation isOfType is implemented in the Meta Model. To simplify matters, it belongs to the ontology module *fundamental_concepts* (as it does not make sense to establish a new module for a single relation).
The following notation convention is recommended for target ontologies:

– Classes that represent value types should be labeled by the suffix '*VT*'.
– Classes that implement ontology views may be labeled by the suffix '*Type*'.

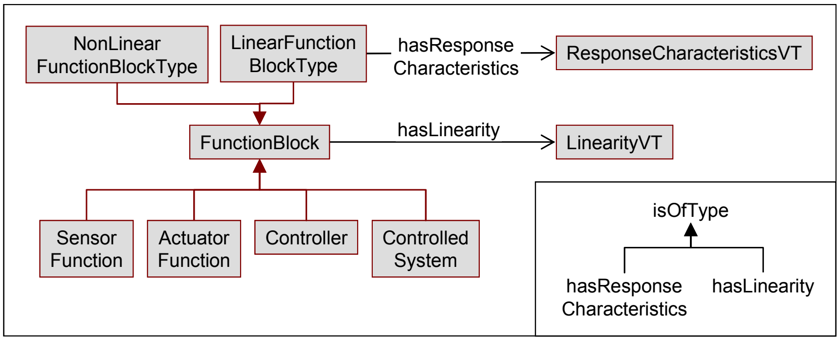

Fig. 4.11: Classification of function blocks

An application example from the area of control theory is presented in Fig. 4.11 and Fig. 4.12: A control loop is composed of different types of *function blocks*, which can be classified as *sensor function*, *actuator function*, *controller*, and *controlled system*.

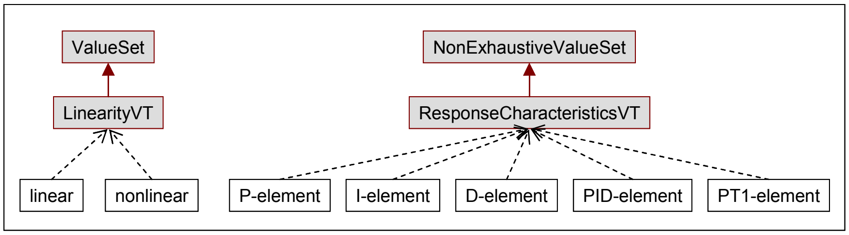

Fig. 4.12: The value types *linearity VT* and *response characteristics VT*

Two further features are of interest about a *function block*: its linearity and its response characteristics. These features are modeled through the value types *linearity VT* and *response characteristics VT*:

- *Linearity VT* is a *value set* made up of the individuals **linear** and **nonlinear**.
- *Response characteristics VT* is a *non-exhaustive value set*, which comprises the individuals **P-element**, **I-element**, **D-element**, **PID-element**, **PT1-element**, and others.

Instances of these two value types are linked to a *function block* and *linear function block* respectively, via the relations hasLinearity and hasResponseCharacteristics, which are specializations of the relation isOfType.
Based on these concepts, different ontology views on *function blocks* can be established. For example, all linear *function blocks* can be retrieved by the class *function block linear type*, which is defined as follows:

- *linear function block type* ≡ *function block* AND hasLinearity **linear**.

Similarly, all PID *controllers* could be retrieved via the class:

– *controller PID type* ≡ *controller* AND hasResponseCharacteristicsVT **PID-element**.

Further ontology views can be realized accordingly.

4.2.2 Concept Descriptions

Individual concepts of the module *fundamental_ concepts* are defined below.

Class Descriptions

Coequal n-ary relation
Coequal n-ary relation is a *relation class* that describes an n-ary relation among three or more individuals or dataype values. None of the individuals involved in the relation stands out as the origin (or owner) of the relation.

Directed n-ary relation
Directed n-ary relation is a *relation class* that describes an n-ary relation among three or more individuals or attribute values. Some of the individuals involved in the n-ary relation are distinguished from the others in that they are origins of the relation. A *relation class* that has at least one *object* as origin.

Feature space
An *object* can be characterized by means of descriptive features (qualities, characteristics). There are various ways how to model the values of such features, for example by representing the values as partitions of classes or as enumerations of individuals – see (Rector 2005) for a detailed discussion of this issue. A feature space defines the range of values that a particular feature can take. The meta root term *feature space* subsumes the different ways to define such a feature space. A *feature space* is either a *value set* or a *value partition* or a *non-exhaustive value set*.

Non-exhaustive value set
A *non-exhaustive value set* is a *feature space* that represents its possible values through individuals. These individuals, which are typically declared to be all different from each other, are instances of the *non-exhaustive value set*. Note that, in contrast to a *value set*, this class is not defined by an (exhaustive) enumeration of its instances. Thus, the number of individuals may change at run time.

Object
Object is a meta root term that subsumes all the self-standing (Rector 2003) entities – whether physical or abstract – that exist in an application domain.

Relation class
The OWL language merely provides language primitives to establish *binary* relations between two individuals or between an individual and an attribute value. To create an *n-ary* relation that links three or more individuals or attribute values, an auxiliary *relation class* needs to be introduced, which acts as an intermediate node. *Relation class* is a meta root term that subsumes the different types of n-ary relations that can be defined (Noy and Rector 2006).
Formal definition: A *relation class* is either a *directed n-ary relation* or a *coequal n-ary relation*.

Unique origin n-ary relation
A *unique origin n-ary relation* is a relation among three or more individuals or attribute values. Exactly one of the individuals involved in the *unique origin n-ary relation* is distinguished from the others in that it is the origin of the relation.

Value partition
A *value partition* is a *feature space* that represents its possible values as disjoint subclasses. These subclasses exhaustively partition the *feature space* and can in turn be further subpartitioned. It is possible to define alternative partitions of the same feature space. Further details about this particular type of feature space can be found elsewhere (Rector 2005: "Pattern 2: Values as subclasses partitioning a feature").

Value set
A *value set* is a *feature space* that represents its possible values through individuals. The individuals, which are typically declared to be all different from each other, are instances of the *value set*. The *value set* is sufficiently defined by an exhaustive enumeration of its instances.

Relation Descriptions

hasOrigin
The relation identifies the *object* that is the origin of a *directed n-ary relation*.

hasTaget
The relation hasTarget identifies the *objects* or feature values (i.e., instances of *feature space*) that are the targets of a *directed n-ary relation*.

hasTagetObject
The relation hasTargetObject identifies the *objects* that are the targets of a *directed n-ary relation*.

inter-objectRelation
The relation inter-objectRelation subsumes all types of binary relations between *objects*.

involves
The relation identifies the *objects* and feature values (i.e., instances of *feature space*) that are involved in an n-ary relation.

involvesObject
The relation identifies the *objects* involved in an n-ary relation.

isInvolvedInN-aryRelation
The relation isInvolvedInN-aryRelation denotes the relation between an *object* and a *relation class*.

isOriginOf
The relation points from an *object* that is the origin of an n-ary relation to a *directed n-ary relation*.

isOfType
The relation isOfType assigns *value types* to *objects*. Based on these characteristics, a reasoner can infer if an *object* belongs to a predefined ontology view.

isTargetOf
The relation points from an *object* that is the target of an n-ary relation to a *directed n-ary relation*.

object-featureRelation
The relation object-featureRelation denotes the relation between an *object* and its feature values (i.e., instances of *feature space*).

Attribute Descriptions

relationAttribute
The attribute relationAttribute identifies an attribute value that is an attribute of a *relation class*.

4.3 Mereology

Mereology is the theory of parthood relations (a.k.a. part-whole relations), i.e., the relations that exist between a part and the whole. There are numerous publications on this subject, e.g. by Simons (1987) or by Casati and Varzi (1999). Varzi (2006) gives an excellent introduction to the field in the Standford Encyclopedia of Philosophy. Different axiomatic systems of mereology exist, which have dissimilar properties. However, the following three axioms form the basis of any mereological theory and can thus be considered as the core principles of mereology. The axioms state the parthood relation to be

- *transitive*: an object that is a part of a part of a whole is itself a part of the whole – if object A is part of object B, and if B is part of object C, then A is part of C;
- *reflexive*: an object is part of itself – A is part of A;
- *antisymmetric*: two distinct objects cannot be part of each other – if A is part of B and A ≠ B, then B cannot be part of A.

Unlike other modeling languages such as UML (e.g., Fowler 1997), OWL does not provide any built-in primitives for part-whole relations. There are various possibilities to model such parthood relations, and the respective approaches have different effects on the usability, expressiveness, and reusability of the ontology as well as on the performance of a reasoner for classifying the ontology. Thus, a design pattern needs to be established that defines a standard way of modeling mereological relations.

The mereology design pattern suggested below follows the best-practice guidelines set out by Rector and Welty (2005) for representing part-whole relations in OWL. In addition, it takes up an idea from UML to distinguish two types of the part-whole relationship: *aggregation* and *composition*:

- Aggregation is the binary relation that exists between an *aggregate* (or whole) and one of its *parts*. A *part* may be part of more than one *aggregate*, i.e., it may be shared by several *aggregates*. A *part* can exist independently from the *aggregate*.
- Composition is a special type of an aggregation relation that exists between a *composite object* and its parts (hereafter: *part of composite object*). *Parts of composite objects* are non-shareable, i.e., they cannot be part of more than one *composite object*. If the *composite object* ceases to exist, its parts cease to exist, as well.

Mereology makes no assumptions on the nature of *aggregates* or *parts*: "They can be material bodies, events, geometric entities, or geographical regions, […] as well as numbers, sets, types, or properties" (Varzi 2006).

Thus, both *aggregates* and *parts* are defined as specializations of the generic *object* class, without imposing any further constraints on them. The two classes are not declared to be disjoint, as an *aggregate* could be at the same time a *part* of another *aggregate*. The relation between a *part* and an *aggregate* is modeled via the relation isPartOf and its inverse hasPart; it is usually depicted through a line with a white diamond-shaped arrowhead pointing towards the *aggregate* class (cf. Fig. 4.13).

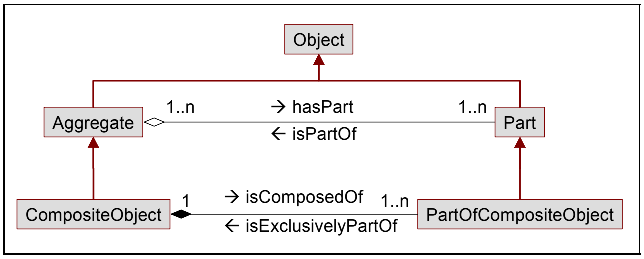

Fig. 4.13: Aggregation and composition

At present, OWL does not provide any language constructs for representing the aforementioned axiom of antisymmetry; neither can the reflexive properties of the parthood relation be properly modeled with the current version of OWL (cf. Rector and Welty 2005). The required extensions to the modeling language have been announced to be incorporated in the next release of OWL (Patel-Schneider and Horrocks 2006). Transitivity, on the other hand, can already be modeled in current OWL by declaring the relations isPartOf and hasPart to be transitive (Fig. 4.14). This enables an OWL-compatible reasoner to infer that, if *object* **A** is a part of *object* **B** and **B** is in turn a part of *object* **C**, then **A** must be a part of **C**, as well.

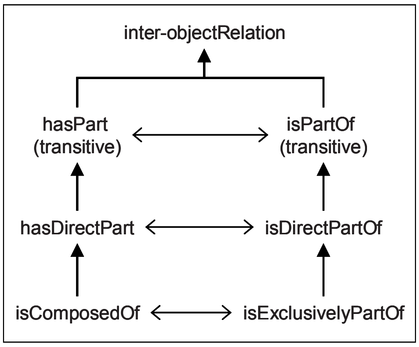

Fig. 4.14: Mereologic relations

Many applications require not a list of all parts but rather a list of the next level breakdown of parts, the so-called *direct parts* of a given entity (Rector and Welty 2005). To this end, the relation hasDirectPart is introduced as a specialization of hasPart; similarly, its inverse isDirectPartOf is declared to be a specialization of the isPartOf relation. These relations are non-transitive and link each subpart just to the next level. Declaring hasDirectPart (and isDirectPartOf) to be a specialization of a hasPart (isPartOf) has the following advantage: If *objects* are repeatedly linked via hasDirectPart (or isDirectPartOf) relation, a reasoner can still infer that a hasPart (isPartOf) relation exists between the *aggregate* and the *part* of a *part*. For example, if **A** is a direct part of **B**, and **B** is a direct part of **C**, it can be inferred that **A** is a part

of **C**. That way, an *aggregate* can be repeatedly decomposed into *parts* and sub-*parts* until the desired decomposition level is achieved.
While the declaration of direct parts seems intuitive at first sight, a possible problem pointed out by Rector and Welty (2005) is that, "the mere idea of a direct part is subjective, one may invent intermediate direct parts depending on numerous factors, or eliminate them. For example, we may choose not to represent engine as a part of cars, but rather represent all the components of engines as direct car parts. Grouping subparts into larger parts may also be subjective, a common example is a flywheel in a car, which can be viewed as an engine part or a transmission part in an ontology that includes those classes". Thus, care must be taken when applying these relations.
For some applications (cf. Chap. 12), it is advantageous to know to which decomposition level a certain *part* belongs. This requires the definition of 'real' *parts*, i.e., *parts* that have no parts of their own; alternatively, 'real' *aggregates* may be introduced. These concepts are located on the top and bottom level, respectively, of the decomposition hierarchy. An exemplary decomposition across four levels is depicted in Fig. 4.15: The class *aggregate only* is defined as an *aggregate* that is not a *part* of any *object*. *First level part* is defined as an *object* that is linked to an *aggregate only* by an isDirectPartOf relation. Similarly, *second level part* is a direct part of a *first level part*, and arbitrary higher-level parts can be defined in an analogous manner. Eventually, the class *part only* is defined as a *part* that does not have any *parts* of its own.

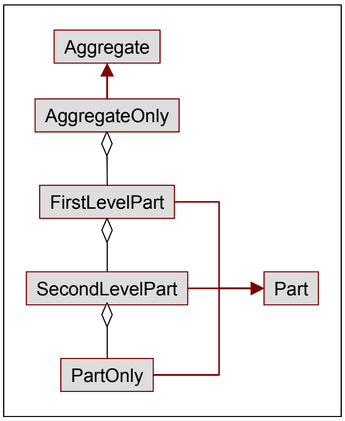

Fig. 4.15: Decomposition structure

Due to the open world assumption customarily made by DL reasoners, membership to the classes part only and aggregate only cannot be inferred, but must be declared explicitly. Once the top (or bottom) of a decomposition hierarchy has been defined that way, the membership to the intermediate decomposition levels

can be inferred automatically. Utilizing these class definitions, a reasoner should be able to assign an *object* to one of these decomposition levels[34].
To represent composition, the classes *composite object* and *part of composite object* are introduced as specializations of *aggregate* and *part*, respectively (Fig. 4.13). Moreover, the relations isComposedOf and its inverse isExclusivelyPartOf are introduced as specializations of the relations hasDirectPart and isDirectPartOf, respectively[35] (cf. Fig. 4.14; in figures, these relations are often depicted through a line with a black diamond-shaped arrowhead pointing towards the *composite object*). A cardinality restriction is imposed on the isExclusivelyPartOf relation to ensure that a *part of composite object* is part of exactly one *composite object*.
Unfortunately, the current OWL reasoners scale very badly when processing large collections of individuals connected via transitive, inverse relations. Hence, part-whole hierarchies that are connected by both hasPart and its inverse isPartOf can cause performance problems. Consequently, Rector and Welty (2005) advice to use either hasPart or isPartOf but not both in large-scale applications. Which one to choose depends very much on the occasion: isPartOf is frequently needed for query formulation, as the most common queries ask for the parts of an object (e.g., the equipment list for a particular plant). On the other hand, many class definitions require a hasPart relation – in OntoCAPE, for instance, the class *plant* is defined as the sum of its parts. Thus, as no relation can be ruled out in advance, both relations are provisionally defined in the Meta Model. Yet for large-scale applications using a reasoner, it might be necessary to abandon one of these.

4.3.1 Usage

The parthood relation is broadly applicable. According to Varzi (2006), it can be used to indicate any portion of a given entity, whatever the nature of the entity, and regardless of whether the portion is material or immaterial, whether it is connected to the remainder or disconnected, whether the part-whole relation has a spatial or a temporal character, and so on. Odell (1994) and Varzi (2006) discuss different kinds of relationships that can be considered as special types of part-whole relations, among which are the following:

– *Material constitution* describes the relation between an object and the material it is made of. This type of relation denotes the constituents of a mixture (Gin is part of Martini) as well as the construction material of a tech-

[34] Implementation advice: Unfortunately, most reasoners are currently not able to infer if an individual is a member of the class part only or aggregate only; hence, the membership to these classes must be declared explicitly for the time being. Membership to the intermediate decomposition levels can be inferred automatically.

[35] If isExclusivelyPartOf was a specialization of isPartOf, it would be impossible to state that a *part of composite object* is part of exactly one *composite object*, as there might be additional *composite objects* on higher aggregation levels.

nical artifact (a car is partly of steel). Material constitution is a special form of aggregation, as a part (i.e., the material) can exist independently of the whole. Hence, hasPart should be used to model this type of relation.
- *Membership* is also a special form of aggregation, as a member can be part of different groups and exists independently of these groups. The isPartOf relation is used to indicate a membership to a group.
- A *portion* is a part that is of the same type as the whole; for example, a slice of bread is a portion of a loaf of bread. A portion relationship is a special type of composition, as the part cannot exist on its own if the whole ceases to exist. Thus, a portion can be linked to the whole via the isExclusivelyPartOf relation.

Rector and Welty (2005) list some potential applications of a mereological ontology; among those are

- a parts inventory for a technical artifact, which requires the "explosion" of parts;
- a fault detection system for a technical device in which one progressively narrows down the functional region of the fault; or.
- a document retrieval system, in which documents are divided into subunits, such as chapters, sections, paragraphs, etc.

Typically, the functionality of such applications can be summarized by the following competency questions:

- Query for the *parts* of an *object*.
- Query for the direct *parts* of an *object*.
- Query for the *first* (*second* …) *level parts* of an *object*.
- Query for the bottom-level *parts* (*part only*) of an *object*.
- Query for all *aggregates* an *object* is part of.
- Query for the *aggregates* an *object* is directly part of.
- Query for the top-level *aggregates* (*aggregates only*) an *object* is directly part of.
- Query if a particular *object* is a *part only*.
- Query if a particular *object* is a *first* (*second* …) *level part*.
- Query if a particular *object* is an *aggregate only*.
- Query if a particular *object* is a part of an *object*.
- Query if a particular *object* is a direct part of an *object*.
- Query if an *object* has a particular *object* as a part.
- Query if an *object* has a particular *object* as a direct part.

Tests have been performed to ensure that the *mereology* ontology module complies with these competency questions: To this end, a test data set was generated, against which the reasoner RacerPro (Racer Systems 2006) evaluated the above queries. The queries were formulated partly as class definitions in OWL and partly as query expression in the nRQL (new Racer Query Language) (Haarslev et al. 2004)[36].

4.3.2 Concept Descriptions

Individual concepts of the module *mereology* are defined below.

Class Descriptions

Aggregate
An *object* that has one or more distinct parts.
Formal definition: An *object* that has some parts of type *object*.

Aggregate only
An *object* that has one or more distinct *parts* and is not part of any *object* itself.
Formal definition: An *aggregate* that is not a *part*.

Composite object
An *object* that is composed of one or more *objects*. The parts of the *composite object* are non-shareable, i.e. an *object* that is part of a *composite object* cannot be part of another *composite object*.
Formal definition: An *object* that is composed of some *objects*.

First level part
A *part* at the first level of decomposition.
Formal definition: A *part* that is a direct part of *aggregate only*.

Part
An *object* that is part of another *object*. A *part* can be part of more than one *object*.
Formal definition: An *object* that is part of an *object*.

Part of composite object
An *object* that is part of a *composite object*. The parts of the *composite object* are non-shareable, i.e. an *object* that is part of a *composite object* cannot be part of another *composite object*.
Formal definition: An *object* that is exclusively part of an *object*.

[36] Implementation advice: While the ontology module successfully passed the tests, two shortcomings of the reasoner were detected:
- The reasoner is currently not able to infer membership to a class that is defined as the com plement of another class. Consequently, membership to the classes part only and aggregate only must be declared explicitly.
- The protégé (Stanford 2008) query interface for the nRQL currently cannot handle concepts that are defined in an external (but included) ontology; thus, it is inappropriate for any modu larly structured ontology.

Part only
An *object* that is part of another *object* and has no *parts* of its own.
Formal definition: A *part* that is not an *aggregate*.

Second level part
A *part* at the second level of decomposition.
Formal definition: A *part* that is a direct part of a *first level part*.

Relation Descriptions

hasDirectPart
Parthood relation that indicates the direct *parts* of an *object*, i.e., the *parts* on the next level breakdown.

hasPart
Parthood relation that refers from an *aggregate* to its *parts*.

isComposedOf
Parthood relation that indicates the direct parts of a *composite object*. The parts of the *composite object* are non-shareable, i.e. a part cannot be part of more than one *composite object*. If the *composite object* is destroyed, all its parts are destroyed, as well.

isDirectPartOf
Parthood relation that links a *part* to the *object* on the next aggregation level.

isExclusivelyPartOf
Parthood relation that links a part to a *composite object* on the next aggregation level. The parts of the *composite object* are non-shareable, i.e. a part cannot be part of more than one *composite object*. If the *composite object* is destroyed, all its parts are destroyed, as well.

isPartOf
Parthood relation that refers from a *part* to the *aggregate*.

4.4 Topology

The ontology module *topology* defines a theory of connectedness. It provides concepts for describing topological relations between distributed entities where there exists the possibility of emergent[37] or supervenient[38] relations between items of in-

[37] Emergence is the process of complex pattern formation from more basic constituent parts or behaviors, and manifests itself as an emergent property of the relationships between those elements (Blitz 1992).

terest (Little and Rogova 2005). Examples of topological relations are the connections between geographical and/or physical entities in 2D and 3D space. Moreover, the concepts of the topology module can be used to describe the ness of abstract entities, such as the unit operations in a process flowsheet or the activities in a business process model.

According to Borst (1997), there are two different approaches to create a topological ontology. One is to extend an existing theory of mereology with topological relations. The other is to integrate mereological and topological concepts and relations into one mereo-topological theory. The approach employed in the Meta Model responds to the former and will be introduced subsequently.

The most fundamental concept of the ontology module *topology* (cf. Fig. 4.16) is the relation isConnectedTo, which denotes the connectivity between *objects*.

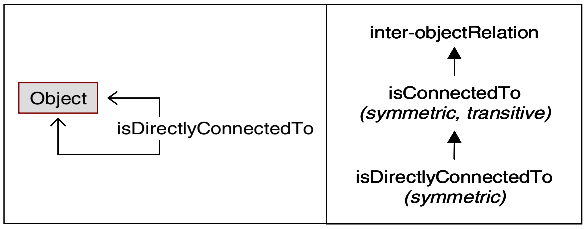

Fig. 4.16: Basic concepts of module *topology*

A first requirement for such a basic topological relation is *symmetry*: if *object* **A** is connected to *object* **B**, then **B** is connected to **A**, as well. A second requirements is *transitivity* – that is, if **A** is connected to **B**, and **B** is in turn connected to **C**, then **A** is also (indirectly) connected to **C**. As an example, consider a vessel (**A**) that is connected to a pipe (**B**), which is again connected to storage tank (**C**) – in this case, storage tank and vessel are (indirectly) connected via the pipe.

Frequently, only the *direct* connections between *objects* are of interest – in the above example, these would be the relations between **A** and **B**, and between **B** and **C**, respectively. The relation isDirectlyConnectedTo is introduced to represent direct connectivity. Similar to the definition of the hasDirectPart relation in the *mereology* module, isDirectlyConnectedTo is declared to be a non-transitive specialization of isConnectedTo. This way of defining direct connectivity enables a reasoner to infer the existence of (indirect) connections from explicitly stated direct connections: For example, if **A** is directly connected to **B**, and **B** is directly connected to **C**, it can be inferred that **A** is (indirectly) connected to **C** (cf. Fig. 4.17).

[38] A set of properties A supervenes upon another set B just in case no two things can differ with respect to A-properties without also differing with respect to their B-properties (McLaughlin and Bennett 2005).

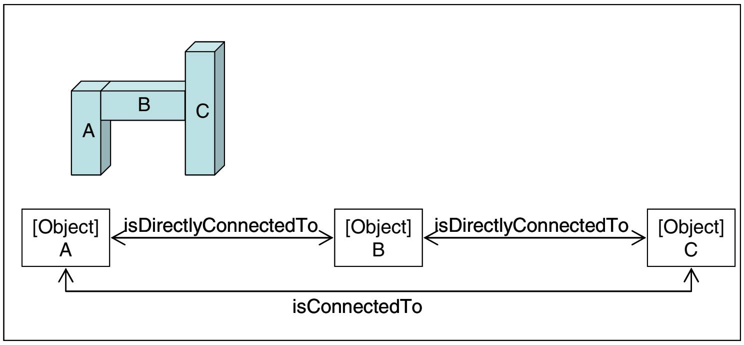

Fig. 4.17: Application example of module *topology*

4.4.1 Mereotopology

A significant aspect of this approach is that mereological and topological relations exclude each other, which prohibits topological relations (connections) between parts and wholes. Hence, a *part* that is linked to an *aggregate* via an isPartOf relation cannot refer to this *aggregate* by any topological relation. An example is given in Fig. 4.18. It shows an *aggregate* **Y** which has the distinct *parts* **a**, **b**, and **c**. A *part* cannot be directly connected to an *aggregate* (Fig. 4.18a); however, the *parts* may be directly connected to each other (Fig. 4.18b).

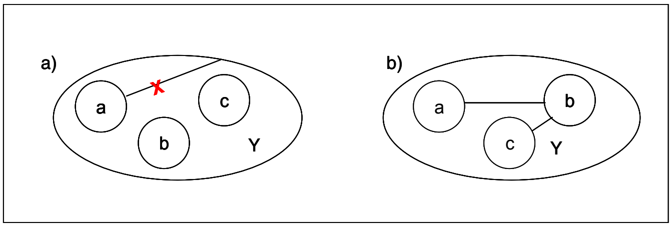

Fig. 4.18: Interdependency between mereological and topological relations

For a more concrete example, consider a cartwheel (**a**) that isPartOf a car (**Y**). Hence, an isConnectedTo relation between the cartwheel (**a**) and the car (**Y**) is prohibited (Fig. 4.18a). Yet a cartwheel isDirectlyConnectedTo an axis (**b**), an axis isDirectlyConnectedTo the car body (**c**), and for the sake of completeness the cartwheel (**a**) isConnectedTo the car body (**c**) (Fig. 4.18b).
To prevent that a direct connection between an *aggregate* and one of its *parts* is established, the following range restrictions are imposed on the isDirectlyConnectedTo relation:

- A *first* (*second* …) *level part* can only be directly connected to (connected to) a *first* (*second* …) *level part*.
- A *part only* can only be directly connected to (connected to) a *part only*.
- An *aggregate only* can only be directly connected to (connected to) an *aggregate only*.

A violation of these restrictions will be considered as an error. Hence, *objects* can only be topologically connected if they are situated at the same level of decomposition. That way, mereological and topological relations are strictly kept apart, only the former or the latter relation can be applied between individuals.

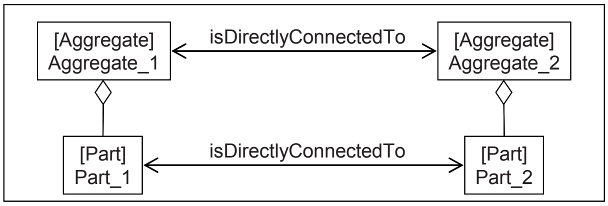

Fig. 4.19: A connection between parts implies a connection between aggregates

Another important point to make is that a connection between two parts of distinct aggregates implies a connection between these aggregates (cf. Fig. 4.19). This can be formulated as a rule: If the parts of distinct aggregates are directly connected, then these aggregates must be directly connected, as well. In contrary, if distinct aggregates are directly connected, the reasoning of connectivity of parts is by no means valid. Unfortunately, there is no means to implement such a rule in OWL; thus, the rule must currently be enforced by the user.

4.4.2 Connectors

The type and number of connections that an *object* may have can be constrained by means of *connectors*. A *connector* represents the interface through which an *object* can be connected to another. A *connector* is a *part* that is linked to an *object* via the isExclusivelyPartOf relation, and it can be connected to exactly one other *connector* via the isDirectlyConnectedTo relation (cf. left-hand side of Fig. 4.20).
Optionally, the possible connections of a *connector* can be further constrained, for instance by postulating that certain properties of two linked *connectors* need to match for a feasible connection. Take the example of two pipes that are to be connected: A connection between two pipes is feasible if the diameters of their nozzles are the same. This situation can be modeled by representing the pipes as instances of *object*, the nozzles as *connectors*, and their diameters as attributes of the respective nozzles (right-hand side of Fig. 4.20). An additional constraint permits only connections between nozzles that have the same diameter.

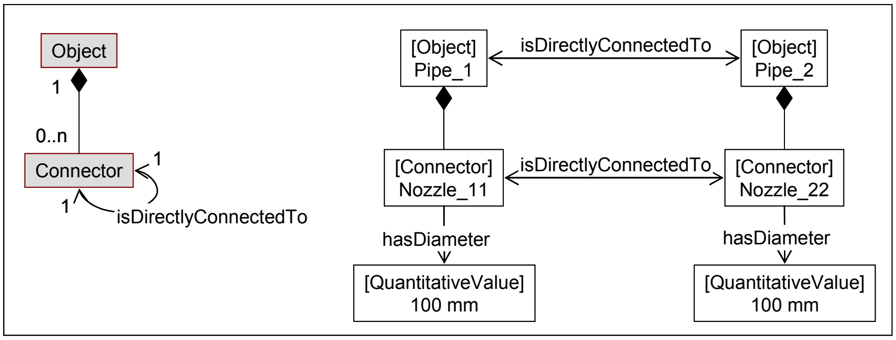

Fig. 4.20: Connecting *objects* via *connectors*

4.4.3 Representation of Graphs

An extended topology which allows for the representation of graphs is represented in Fig. 4.21; the major concepts of this approach are *nodes* and *arcs*. Basically, an *arc* cannot connect to more than two *nodes*, which excludes *arcs* that fork. A *node*, on the other hand, can be connected to several *arcs*.

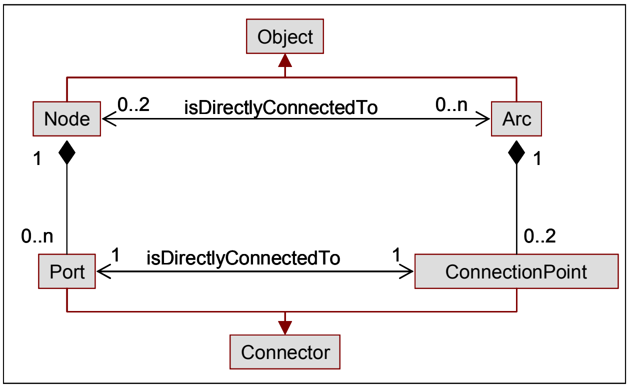

Fig. 4.21: *Nodes* and *arcs*

Optionally, a *node* may have a list of *ports*, and an *arc* may have up to two *connection points*. *Ports* and *connection points* are specializations of the *connector* class; they are linked to the corresponding *node* or *arc* via the isExclusivelyPartOf relation and can be connected to each other via the isDirectlyConnectedTo relation.

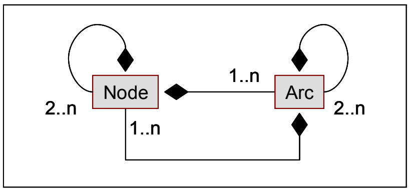

Fig. 4.22: Decomposition of *nodes* and *arcs*

Ports and *connection points* act as interfaces to *nodes* and *arcs*, respectively: like *connectors*, they carry specific characteristics that have to match if a *port* is to be connected to a *connection point*. That way, they restrict and further specify the type and number of connections that a *node* or an *arc* can have.
Another important issue is that both *nodes* and *arcs* can be decomposed into a number of sub-*nodes* and sub-*arcs*, respectively (Fig. 4.22).
When a *node* is decomposed into a number of sub-*nodes*, it is necessary for these sub-*nodes* to be connected by internal *arcs*. Similarly, when *arcs* are decomposed into sub-*arcs*, there must be internal *nodes* between the sub-*arcs*. Thus, a *node* has to be decomposed into two *nodes* and one connecting *arc*, at least; likewise, an *arc* cannot be decomposed in less than two *arcs* and one *node*[39]. The sub-*nodes* and sub-*arcs* are connected via isDirectlyConnectedTo relations (Fig. 4.23).

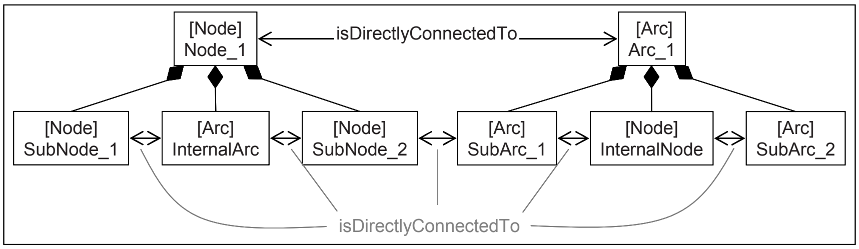

Fig. 4.23: Connections between sub-*nodes* and sub-*arcs*

A special situation arises if a *node* is decomposed while the connected *arc* is not[40]. Such a pattern occurs, for example, when a process flowsheet is hierarchically refined, as it is exemplarily shown in Fig. 4.24: Here, the *node* representing the overall process is decomposed into a reaction section and a separation section, whereas the *arcs* representing the feed and product streams are not decomposed at all. Now the question arises, which sub-*node* is connected to which *arc* (in the ex-

[39] Unfortunately, it is presently not possible to model this decomposition axiom in OWL, as the current version of OWL does not support qualified cardinality restrictions (QCR). However, QCRs will be incorporated in the upcoming OWL 1.1 (Patel-Schneider and Horrocks 2006).

[40] Of course, the same considerations apply to the opposite case, when the *arc* is decomposed while the *node* is not. To simplify matters, only the first case is discussed here.

ample, the feed stream enters the reaction section, whereas the product stream leaves the separation section).

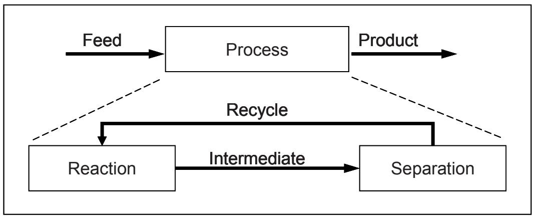

Fig. 4.24: Hierarchical refinement of a process flowsheet

A straight-forward solution is to connect the *arcs* representing the feed and product streams directly to the *nodes* representing the reaction and separation sections, as indicated in Fig. 4.25. Remember however, that topological connections are only permitted between *nodes* and *arcs* that are situated on the same level of decomposition. Therefore, this solution is only applicable if the mereological levels of the feed and product *arcs* are not fixed, that is, if the feed and product *arcs* can be assigned the same level of decomposition as the sub-*nodes* for reaction and separation. If this is not possible, an alternative solution must be chosen. Note that for the sake of clarity, the internal *arc* representing the recycle stream is neglected in the following.

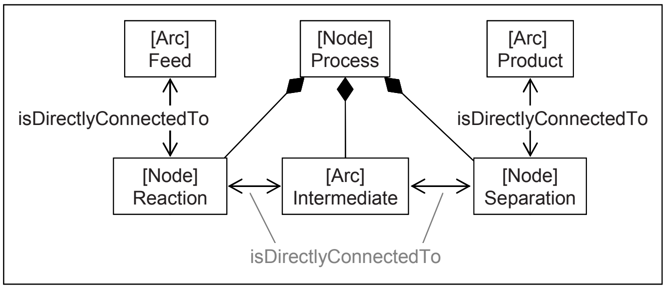

Fig. 4.25: Solution alternative 1 – the *arcs* are directly connected to the refined sub-*nodes*

If a direct connection between an *arc* and a sub-*node* is not feasible, the two may still be indirectly linked via their respective *ports* and *connection points*. Fig. 4.26 presents the corresponding pattern: *Port* and *connection point* are to be linked via an isDirectlyConnectedTo relation.

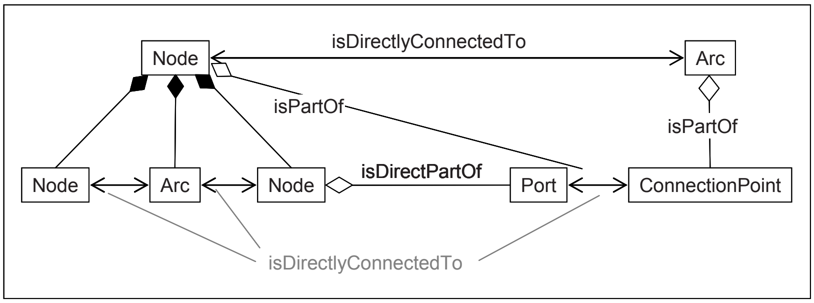

Fig. 4.26: Alternative 2 – an *arc* is indirectly linked to a sub-*node* via a *port* and a *connection point*

While the *port* may be a direct part of the sub-*node*, the *connection point* must only be an indirect part of the *arc*. The reason for this is, again, the required compatibility of the decomposition levels: If the *connection point* was a direct part of the *arc*, then *port* and *connection point* would be situated on different levels and thus could not be connected. Note that this alternative is not feasible for the refinement of the process flowsheet and thus represented in a generic way.
In case that the *node* and *arc* do not have designated *ports* and *connection points*, the above solution is not applicable. Alternatively, the *arc* may simply be duplicated; that is, a placeholder *arc* is to be introduced (cf. Fig. 4.27). The correspondence between the placeholder *arc* and the original *arc* is established via the relation sameAs.

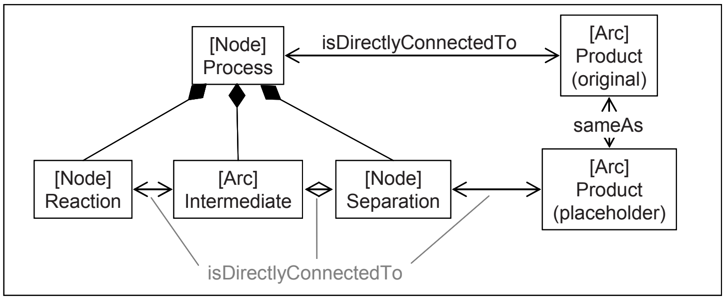

Fig. 4.27: Alternative 3 – duplication of the *arc*; correspondence is established via the sameAs relation

4.4.4 Directed Graphs

A further extension of the *topology* module allows for the representation of directed graphs: To this end, the class *directed arc* is introduced, which can be employed to indicate a directed connection between *nodes* such that one *node* is the predecessor or the successor of the other. As shown in Fig. 4.28 a *directed arc* is linked to a *node* via the relations enters and leaves, respectively.

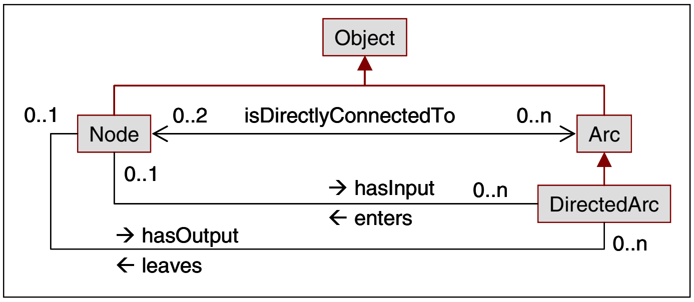

Fig. 4.28: Extended topology including directed arcs

The relation taxonomy presented in Fig. 4.12 is extended, as shown in Fig. 4.29.

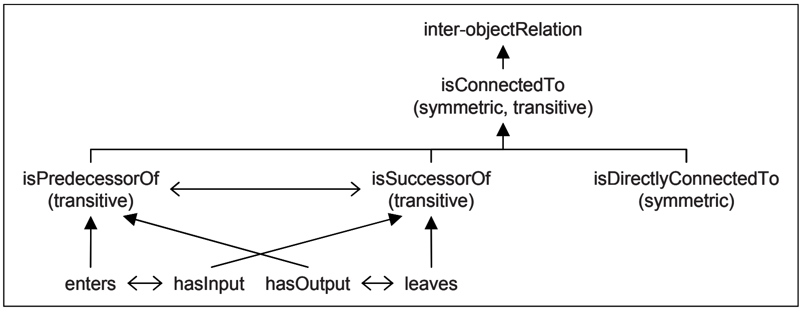
inter-objectRelation
isConnectedTo (symmetric, transitive)
isPredecessorOf (transitive)
isSuccessorOf (transitive)
isDirectlyConnectedTo (symmetric)
enters
hasInput
hasOutput
leaves

Fig. 4.29: Extended relation taxonomy

The relation enters and its inverse hasOutput are specializations of the transitive relation isPredecessorOf, which is a specialization of isConnectedTo. Similarly, the relations leaves and its inverse hasInput are specializations of isSuccessorOf, which is the inverse of isPredecessorOf. The relations hasInput and hasOutput are to be used to identify the *directed arcs* that are directly attached to a *node*. The main purpose of the supplementary relations isPredecessorOf and isSuccessorOf is to identify the *nodes* (or *directed arcs*) that precede or succeed a specific *node* (or *directed arc*) in a directed graph[41]. As mentioned in the specification of the *mereology* module, the current OWL reasoners scale badly when processing large collections of individuals connected via transitive, inverse relations. Thus, for large-scale applications, it

[41] In graph theory, a node B is considered to be the successor of node A, if a path leads from A to B.

might be necessary to abandon either the isPredecessorOf relation or the isSuccessorOf relation.

4.4.5 Usage

To illustrate the functionality of the *topology* module, several competency questions are introduced; afterwards two examples are given to demonstrate that the *topology* module complies with the competency questions.
A primary distinction is made between directed and non-directed connections. For the non-directed connections, the following competency questions are defined (the classes in parenthesis are optional):

- Query for all *objects* (*nodes*, *arcs*) that are directly connected to a specific *object.*
- Query for all *objects* (*nodes*, *arcs*) that are connected to a specific *object.*
- Query for all *arcs* that are connected to a *node* via a particular *port*.
- Query if two *objects* (*nodes*, *arcs*) are connected directly.
- Query if two *objects* (*nodes*, *arcs*) are connected (either directly or indirectly).
- Check if topological relations are wrongly defined across different levels of aggregation.

Competency questions for the directed connections may comprise the former as well as the subsequent ones:

- Query for all *directed arcs* that enter a specified *node.*
- Query for all *directed arcs* that leave a specified *node.*
- Query for all *objects* (*nodes*, *arcs*) that are predecessors of a specified *object* (*node*, *arc*).
- Query for all *objects* (*nodes*, *arcs*) that are successors of a specified *object* (*node*, *arc*).

The first example presented in Fig. 4.30 shows a directed graph consisting of four different *nodes* (**A**, **B**, **C**, **D**), which are connected by three *directed arcs* (**A→B**, **B→C**, **C→D**). The lower part of Fig. 4.30 shows how such a graph is represented through an instantiation of the above concepts. If the example is used as a test case for the *topology* module, a reasoner will give the following results (relating to individual **C**) for directed relations. Please note that for the sake of clarity, the respective class names in brackets are omitted hereafter.

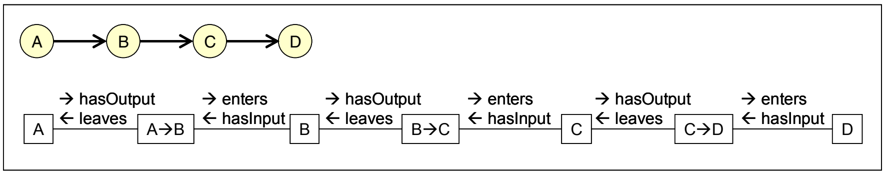

Fig. 4.30: Application example 1: acyclic directed graph

– *Objects* connected to **C**: {**A**, **A→B**, **B**, **B→C**, **C**, **C→D**, **D**}(as **C** is directly connected to **B→C**, and **B→C** is in turn connected to **C**, the reasoner infers that **C** is indirectly connected to itself).
– *Nodes* connected to **C**: {**A**, **B**, **C**, **D**}.
– *Arcs* connected to **C** : {**A→B**, **B→C**, **C→D**}.
– *Objects* preceding **C**: {**A**, **A→B**, **B**, **B→C**}.
– *Nodes* preceding **C**: {**A**, **B**}.
– *Arcs* preceding **C**: {**A→B**, **B→C**}.
– *Objects* succeeding **C**: {**C→D**, **D**}.
– *Nodes* succeeding **C**: {**D**}.
– *Arcs* succeeding **C**: {**C→D** }.
– *Arcs* entering **C**: {**B→C**}.
– *Arcs* leaving **C**: {**C→D**}.

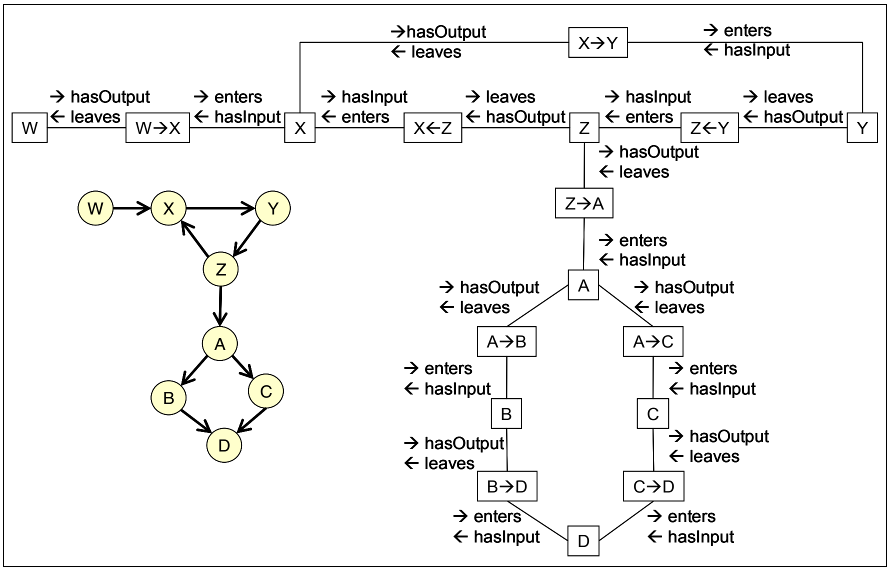

Fig. 4.31: Application example 2: directed graph with a cycle and a bifurcation

The second example presented in Fig. 4.31 shows a directed graph consisting of 8 different nodes (A, B, C, D, W, X, Y, Z) connected by directed arcs. The graph includes a cycle (X, Y, Z) as well as a bifurcation (A, B, C). Fig. 4.31 illustrates how this example is represented by an instantiation of the topological concepts. Taking this example as a test case, the following interesting results are obtained:

- *Nodes* preceding **D**: {**A**, **B**, **C**, **W**, **X**, **Y**, **Z**}.
- *Nodes* succeeding **W**: {**A**, **B**, **C**, **D**, **X**, **Y**, **Z**}.
- *Nodes* succeeding **Y**: {**A**, **B**, **C**, **D**, **X**, **Y**, **Z**} (i.e., the reasoner infers that all cycle *nodes*, including **Y**, are successors of **Y**).

4.4.6 Concept Descriptions

Individual concepts of the module *topology* are defined below.

Class Descriptions

Arc
Arc is a specialization of *object* and represents the connecting element between *nodes*.

Connection point
Connection point represents the interface through which an *arc* can be connected to the *port* of a *node*. *Connection points* may have certain attributes that further specify the type of connection. *Connection points* are *parts* of the corresponding *arc* or *directed arc*.

Connector
A *connector* represents the interface through which an *object* can be connected to another *object*. Typically, the possible connections of the *connector* are further constrained, for instance by postulating that certain properties of the connected *connectors* need to match for a feasible connection.

Directed arc
Directed arc is a specialization of *arc* and represents likewise the connecting element between *nodes*. However, the usage of *directed arc* implies the indication of a directed connection.

Node
Node is a specialization of *object* and is used to model the crucial elements (joints) which are connected by *arcs*.

Port
Ports represent the interfaces through which *nodes* are connected to *arcs*. A *port* may have certain attributes that characterize the type of connection.

Relation Descriptions

enters
The relation enters connects an incoming *directed arc* to its target *node.*

hasInput
The relation hasInput connects a *node* to an incoming *directed arc*.

hasOutput
The relation hasOutput connects a *node* to an outgoing *directed arc*.

isConnectedTo
The relation isConnectedTo represents topological connectivity between *objects.*

isDirectlyConnectedTo
The relation isDirectlyConnectedTo denotes the direct topological connectedness of two *objects.*

isSuccessorOf
The relation isSuccessorOf identifies all *nodes* and *directed arcs* that are successors of the considered one.

isPredecessorOf
The relation isPredecessorOf identifies all *nodes* and *directed arcs* that are predecessors of the considered one.

leaves
The relation leaves connects an outgoing *directed arc* to its source *node*.

sameAs
The relation denotes a correspondence between an *arc* and its placeholder in a decomposition hierarchy.

4.5 Data Structures

The partial model **data_structures** provides a design pattern for the representation of customary data structures. The below pattern provide some data structures which occur repeatedly with special relevance to the modeling of OntoCAPE. However, these data structures are fully consistent with those typically defined and applied in computer science (e.g., Black 2004). It comprises the ontology modules *binary_tree*, *array*, *linked_list*, *multiset*, and *loop*, which are presented in the following. Note that the design patterns incorporate transitive, inverse relations, which may cause performance problems (cf. Sect.3.2). Thus, for large-scale applications, it might prove necessary to abstain from implementing the inverse relations.

4.5.1 Binary Tree

A *binary tree* is a tree-like data structure that is formed by a set of linked *nodes*. A node can have zero, one, or two *child nodes*. Each child node is identified as either the *left* child or the *right* child. Fig. 4.32 shows an exemplary binary tree.

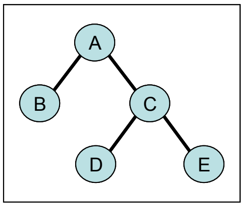

Fig. 4.32: Example of a binary tree

The topmost[42] element of the tree is called the *root node* (node A in Fig. 4.32). A node that has a child is called the child's *parent node*. Except for the root node, each node has one parent node.
Nodes that lie below a certain node (i.e., its children, grandchildren, etc.) are called the *descendents* of this node. Similarly, a node's *ancestors* are the nodes that are traversed when moving up the tree (i.e., the node's parent, grandparent, etc.). In Fig. 4.32, for example, the nodes B, C, D, and E are descendents of node A; node E has the ancestors A and C.

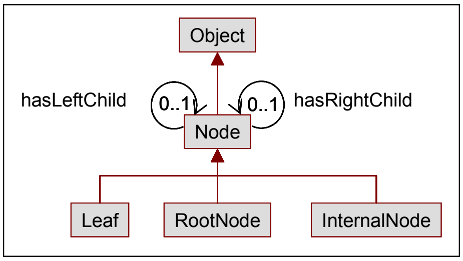

Fig. 4.33: Design pattern for the representation of binary trees

[42] By convention, binary trees are depicted top-down.

Fig. 4.33 and Fig. 4.34 illustrate how a binary tree is represented in the Meta Model. *Node* is the basic element for the construction of a tree. Three specializations of *node* are introduced:

- the *root node* is a node without a parent;
- an *internal node* is a node that has both a parent node and a child node;
- a *leaf* is a node without children.

A *node* is linked to its child *nodes* via the relations hasLeftChild and hasRightChild. The relation hasChild subsumes these two relations, as shown in Fig. 4.34.

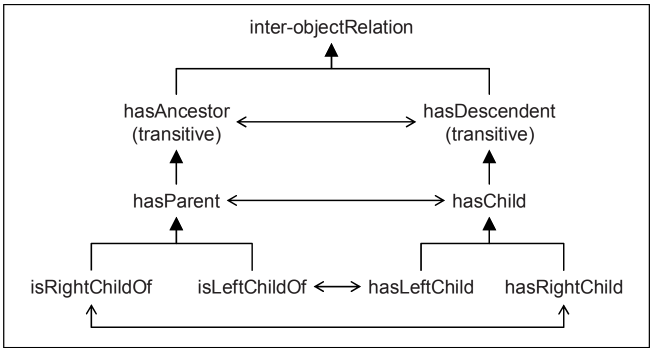

Fig. 4.34: Relations for the representation of binary trees

The relation hasParent is defined as the inverse of hasChild. It has two specializations: isLeftChildOf, which is the inverse of hasLeftChild, and isRightChildOf, which is the inverse of hasRightChild. Finally, the relation hasAncestor and its inverse hasDescendent are introduced to denote the ancestors and descendents of a particular *node*.

An application example is shown in Fig. 4.35, which uses the above concepts to represent the tree depicted in Fig. 4.32.

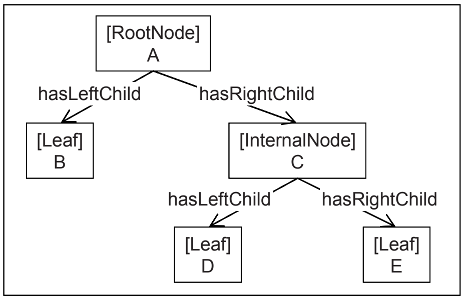

Fig. 4.35: Application example

Only the relations hasLeftChild and hasRightChild need to be explicitly defined between the *nodes*. All the other relations (i.e., the parent, ancestors, and descendents of a particular *node*) can be automatically inferred by a reasoner.
Note that a *node* may have more than one parent *node*, if that particular *node* forms part of more than one binary tree.

4.5.1.1 Usage

The design pattern *binary_tree* complies with the following competency questions:

- Query for all *nodes* of a particular tree (which is identified through its *root node*).
- Query for the *leaves* of a particular tree (which is identified through its *root node*).
- Query for the direct children of a particular *node*.
- Query for the left/right child of a particular *node*.
- Query for the descendents of a particular *node*.
- Query for the direct parent of a particular *node*.
- Query for the ancestors of a particular *node*.
- Query for the *root node* of a particular tree (which is identified through one of its *nodes*).

A possible application of the *binary_tree* pattern is the representation of mathematical expressions. The *leaves* of such an expression tree denote the operands in the expression, and the *internal nodes* denote the operators[43].

4.5.1.2 Concept Descriptions

Individual concepts of the module *binary_tree* are defined below.

Class Descriptions

Internal node
An *internal node* is a *node* that has one parent and at least one child.
Formal definition: A *node* that has both a parent *node* and a child *node*.

[43] Implementation advice: In principle, it is not necessary to explicitly declare a node to be a root node or leaf, as this can be inferred by a reasoner. However, some reasoners cannot evaluate this kind of statement (i.e., that a node has zero ancestors/descendents) – in this case, root node and leaves must be explicitly identified if required for an application. Internal nodes are automatically found by a reasoner.

Leaf
A *leaf* is a *node* without any children.
Formal definition: A *node* that has no child nodes.

Node
A *node*[44] is the basic element of a binary tree. It can be linked to up to two child *nodes*.
Formal definition: A *node* is either a *leaf* or a *root node* or an *internal node*.

Root node
A *root node* is the root element of a binary tree. All other *nodes* are descendents of the *root node*.
Formal definition: A *node* without any parent.

Relation Descriptions

hasAncestor
The ancestors of a *node* are the *nodes* that precede the current *node* in the tree (i.e., the *node*'s parent, grandparent, etc.).

hasChild
The relation hasChild points to the children of a *node*; it subsumes the relations hasLeftChild and hasRightChild.

hasDescendent
The descendents of a *node* are the *nodes* that succeed the current *node* in the tree (i.e., the *node*'s children, grandchildren, etc.).

hasLeftChild
The relation hasLeftChild links a parent *node* to its left child *node*.

hasParent
The relation hasParent denotes the parent of a *node*.

hasRightChild
The relation hasRightChild links a parent *node* to its right child *node*.

isLeftChildOf
The relation isLeftChildOf points from the left child *node* to its parent *node*.

isRightChildOf
The relation isRightChildOf points from the right child *node* to its parent *node*.

[44] Please note that this *node* is different to the one defined in topology (cf. Section 4.4.3). This is indicated by different namespace prefixes in the formal specification.

4.5.2 Multiset

A *multiset* differs from an ordinary set in that there may be multiple appearances of the same element. For example, the multiset {a, a, b, b, b, c} has two appearances of element a and three appearances of element b.

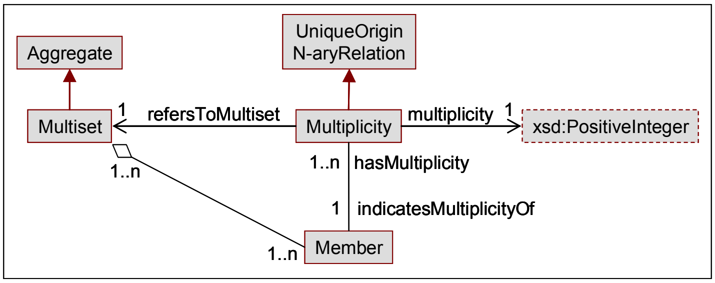

Fig. 4.36: Design pattern for the representation of multisets

In the Meta Model, a *multiset* is modeled as a special type of *aggregate* (cf. Fig. 4.36). Its elements, which are direct parts of the *multiset*, are called *members*. Each *member* has a *multiplicity* that indicates the number of its appearances within the multiset. A *member* may be a member of more than one *multiset*; in this case, the *member* must have one *multiplicity* for each of these memberships. For this reason, the *multiplicity* is modeled as a *unique origin n-ary relation* that relates the various *multiplicities* of a *member* to the respective *multisets*.

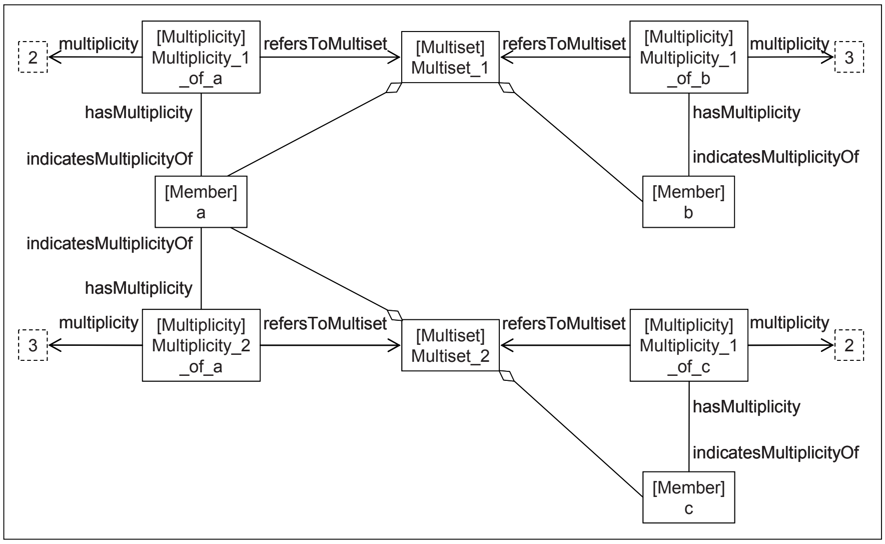

Fig. 4.37: Application example: element a is a member of both multisets

An application example is given in Fig. 4.37. It shows the ontological representation of two *multisets*:

- **Multiset 1** = {**a**, **a**, **b**, **b**, **b**}, and
- **Multiset 2** = {**a**, **a**, **a**, **c**, **c**}.

Obviously, individual **a** is a *member* of both *multisets*. **a** has a *multiplicity* of two in **Multiset 1**, and a *multiplicity* of three in **Multiset 2**. The relation refersToMultiset indicates which *multiplicity* is related to which *multiset*.

4.5.2.1 Usage

The design pattern *multiset* complies with the following competency questions:

- Query for the *members* of a particular *multiset*.
- Query for the *multiplicity* that a *member* has in a particular *multiset*.
- Query for the *multiset* of which a member is a part.

A *multiset* may be used as a shorthand to specify an *object* that is composed of alike parts (in this context, 'alike' means that the parts share certain characteristic features). Instead of specifying all these parts individually, it is sufficient to describe one representative part (*member*) and indicate its *multiplicity*. For example, a distillation column can be thermodynamically characterized by (1) describing the VLE on one tray and (2) indicating the total number of trays.

4.5.2.2 Concept Descriptions

Individual concepts of the module *multiset* are defined below.

Class Descriptions

Multiplicity
The *multiplicity* of a *member* indicates the number of its appearances in the associated *multiset*.
<u>Formal definition</u>: A *multiplicity* indicates the multiplicity of some *member*.

Multiset
A *multiset* differs from an ordinary *aggregate* in that each of its parts (*members*) has an associated *multiplicity*, which indicates the number of its appearances in the *multiset*.
<u>Formal definition</u>: A *multiset* is an *aggregate* that has at least one *member*.

Member
A *member* is a direct part of a *multiset*; it has a *multiplicity* that indicates the number of its appearances in the *multiset*.
Formal definition: A *member* is a *part* that has a *multiplicity*.

Relation Descriptions

hasMultiplicity
The relation hasMultiplicity points from a *member* to a *multiplicity* that indicates the number of its appearances in a particular *multiset*.

indicatesMultiplicityOf
The relation indicatesMultiplicityOf links a *multiplicity* to the corresponding *member*.

refersToMultiset
The relation refersToMultiset assigns a *multiplicity* to the corresponding *multiset*.

Attribute Description

multiplicity
The attribute multiplicity indicates the numerical value of a *multiplicity*.

4.5.3 Array

An *array* holds an ordered collection of objects, which are called the *elements* of the array. Similar to a *multiset*, an element can have multiple appearances in the array. The elements are ordered by an *index*, which specifies the position of an *element* within the array through a consecutive sequence of integer values. Individual elements can be accessed via their respective index values. Fig. 4.38 shows the design pattern that defines an array in the Meta Model.

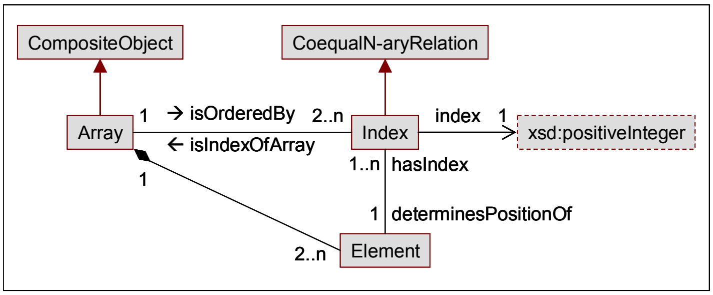

Fig. 4.38: Design pattern for the representation of arrays

An *array* is a specialization of a *composite object* which is composed of two or more *elements*. The position of an *element* within the *array* is specified by the *index*. An *index* is a *coequal n-ray relation* between an *array*, one of its *elements*, and the integer attribute value that denotes the position of the *element* in the *array*.
An application example of the *array* design pattern is given in Fig. 4.39. The *array* **A[i]** has the *elements* **x** and **y**. The *index* of **x** (**Index_of_x**) has an index value of 1, whereas the *index* of **y** (**Index_of_y**) has an index value of 2. Thus, **x** is the first *element* of **A[i]**, and **y** is the second one.

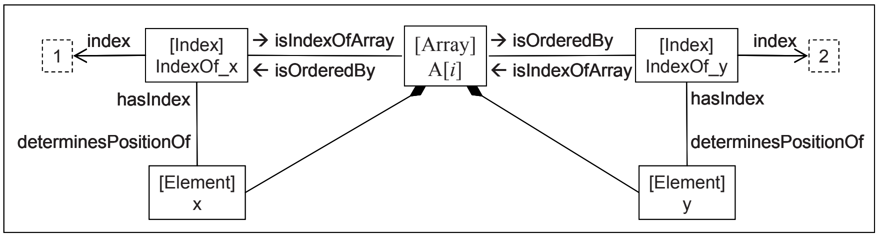

Fig. 4.39: Application example: representation of an array **A**[*i*] with elements **A**[1] = *x* and **A**[2] = *y*

4.5.3.1 Usage

The design pattern *array* complies with the following competency questions:

– Query for an *element* with a particular index value.
– Query for the *index* of a particular *element*.
– Query for all *elements* of an *array*.
– Query for the *array* to which a particular *elements* belongs.
– Query for the *array* to which a particular *index* belongs.

In the ontology OntoCAPE, the design pattern *array* is applied to model tensor quantities, such as vectors and matrices.

4.5.3.2 Concept Descriptions

Individual concepts of the module *array* are defined below.

Class Descriptions

Array
An *array* is an ordered list that is composed of two or more *elements*. The position of an *element* within the *array* is specified by the *index*.

Element
An *element* is part of an *array*. Its position within the *array* is determined by an *index*.

Index
An *index* represents the *coequal n-ary relation* between an *array*, one of its *elements*, and the integer attribute value that denotes the position of the *element* in the *array*.
Formal definition: An *index* determines the position of some *element*.

Relation Descriptions

determinesPositionOf
The one-to-one relation between an *index* and the corresponding *element*.

hasIndex
The one-to-one relation between an *element* and its *index*.

isIndexOfArray
The relation isIndexOfArray points form an *index* to the associated *array*

isOrderedBy
The relation isOrderedBy points from an *array* and to the sorting *index*.

Attribute Description

index
The attribute index indicates the numerical value of an *index*.

4.5.4 Linked List

Similar to an array, a *linked list* is an ordered collection of objects (cf. Fig. 4.40). It is formed by a sequence of *list elements*, each pointing to the next (and possibly the previous) element in the list. List elements can be inserted and removed at any point in the list. Unlike an array, a linked list does not allow random access[45].

[45] Random access is the ability to access any particular element in the list in constant time.

In the Meta Model, a *linked list* is modeled as a specialization of a *composite object* that is composed of two or more *list elements*. A *list element* points to the next as well as to the previous *list element* through the relations nextElement and previousElement, respectively. Three disjoint subclasses of *list element* are introduced:

– the *first element* of the list, which is a *list element* that does not point to a previous *list element*; it must have one next *list element*;
– the *last element* of the list, which is a *list element* that does not point to a next *list element*; it must have one previous *list element*; and
– *internal element*, which is defined as a *list element* that points to both a previous and a next *list element*.

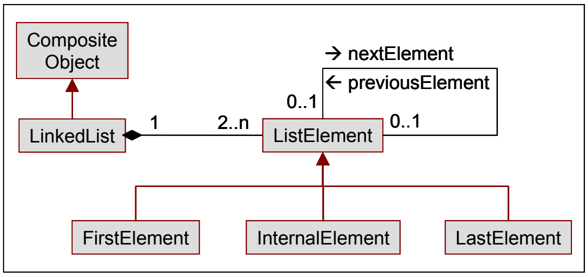

Fig. 4.40: Design pattern for the representation of a linked list

4.5.4.1 Usage

An application example is given in Fig. 4.41. It demonstrates how to represent a *linked list* with the *list elements* **x**, **y** and **z**; **x** is the *first element*, **y** is an *internal element*, and **z** is the *last element* of the *linked list*.

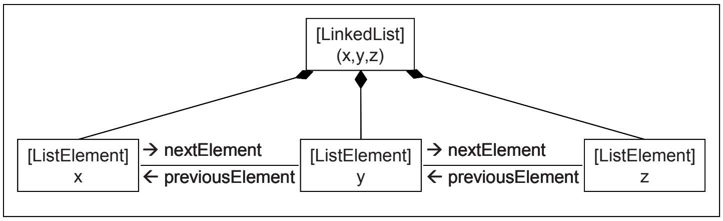

Fig. 4.41: Application example – a linked list with elements x, y, and z

The design pattern *linked list* complies with the following competency questions:

– Query for the *first element* of a particular *linked list*.
– Query for the second (third, fourth, …) *list element* of a particular *linked list*.

– Query for the *last element* of a particular *linked list*.
– Query for all *list elements.*
– Query for the *list element* succeeding a particular *list element*.
– Query for the *list element* preceding a particular *list element*.

Note that a *list element* cannot have multiple appearances in a *linked list*[46].

4.5.4.2 Concept Descriptions

Individual concepts of the module *linked list* are defined below.

Class Descriptions

First element
The first *list element* of a *linked list*.

Internal element
A *list element* that is neither the first nor the last element of a *linked list*.
Formal definition: *Internal element* is a *list element* that points to both a next and a pervious *list element*.

Last element
The last *list element* of a *linked list*.

Linked list
A *linked list* is formed by a sequence of *list elements*, each pointing to the next as well as to the previous *list element*.

List element
A *list element* is an element of a *linked list*; it may point to a next as well as to a previous *list element*.

[46] Implementation advice: In principle, it is not necessary to declare the first element and last element of a linked list explicitly, as they can be automatically found by a reasoner. This facilitates to add and remove list elements at arbitrary positions. However, some reasoners cannot evaluate the definition – in this case, the first element and last elements must be explicitly defined.
The following ontological assertions cause problems with the reasoner RacerPro and have therefore been omitted in the current version of the Meta Model:
- A list element is either a first element or an internal element or a last element.
- A linked list is composed of some first element (for some reason does the corresponding statement “A linked list is composed of some last element not cause any trouble).
- nextElement is only of type internal element oder last element (requires too much computation time).
- previousElement is only of type internal element or first element (as above).
- first element is a list element that does not have a previous list element.
- last element is a list element that does not have a next list element.

Relation Descriptions

nextElement
The relation nextElement points from a *list element* to the next *list element*.

previousElement
The relation previousElement points from a *list element* to the previous *list element*.

4.5.5 Loop

The design pattern *loop*[47] introduces a shorthand for representing structures that consist of repetitive, interlinked *objects*. This is best explained by means of an example: Consider the structure displayed in Fig. 4.42: the individuals **O1** to **O5** are sequentially connected via the relation 'o'. Furthermore, **O1** is linked to **A1**, **O2** is linked to **A2**, and so forth, via the relation 'a'. Also, each of the $\mathbf{O}_i$ is linked to individual **X** via the relation 'x'; thus, **X** represents a common feature of the $\mathbf{O}_i$. Individual **R** is linked to **O1**, and **O5** is linked to **S**; thus, **R** and **S** represent the endpoint conditions of the structure.

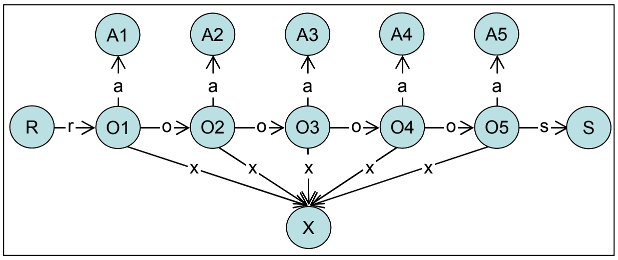

Fig. 4.42: Repetitive, interlinked structure

Instead of defining this structure explicitly, it can be represented as indicated in Fig. 4.43: First, a **Loop** is introduced, which has 5 numbersOfIteration. Several individuals are linked to the **Loop**. The relation 'statementFor_i' identifies those individuals that depend on the iterations *i*; in this example, these are the individuals **O_i** and **A_i**. This means that, for each iteration *i*, one $\mathbf{O}_i$ and one $\mathbf{A}_i$ exists.
O_i and **A_i** are connected via the relation 'a'; consequently, one 'a' is established between each occurrence of $\mathbf{O}_i$ and $\mathbf{A}_i$. The relation 'x' points from **O_i** to individual **X**, which is not linked to the **Loop**; thus, each occurrence of $\mathbf{O}_i$ points to the same **X**. For $i = 1$ and $i = 5$, the $\mathbf{O}_i$ have connections to **R** and **S**, respectively. To represent these additional connections, the individuals **O_1** and **O_N** are intro-

[47] The name 'for loop' is chosen because the syntax used to represent the loop pattern is similar to that of a 'for loop' in a programming language.

duced. **O_1** is linked to **R**, and **O_N** is linked to **S**. To indicate that **O_1** and **O_N** represent the first and last occurrence of **O**$_i$ in the Loop, they are (1) linked to **O_i** via the relation sameObject, and (2) connected to the **Loop** by means of the relations initialStatement and finalStatement.

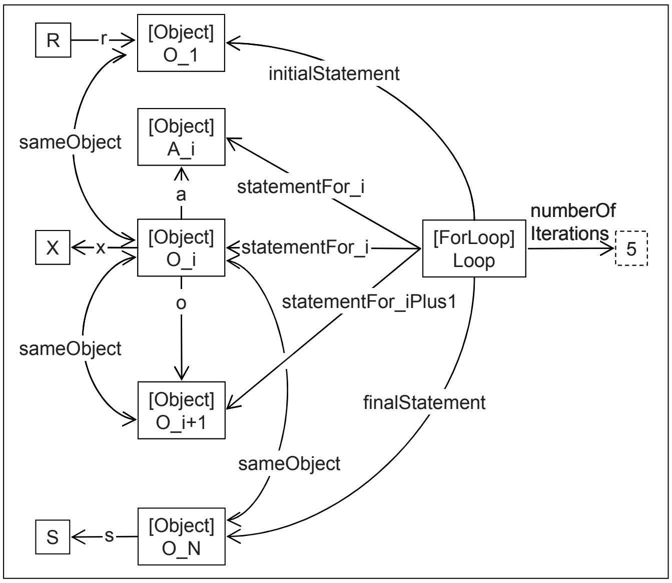

Fig. 4.43: Representation of the structure shown in Fig. 4.41 by using the *loop* design pattern

Finally the relations between the **O**$_i$ must be specified. To this end, the individual **O_i+1** is introduced, which represents the occurrence of **O**$_i$ in the iteration (i + 1). The semantics of **O_i+1** is explicitly defined by (1) establishing a sameObject relationship between **O_i** and **O_i+1**, and (2) linking **O_i+1** to the **Loop** via the relation **statementFor_iPlus1**. Lastly, the relation ‘o’ is declared between **O_i** and **O_i+1**, thus indicating that ‘o’ exists in between the different **O**$_i$.
Fig. 4.44 defines the classes and relations that are required to establish a *loop* pattern as the one exemplarily shown above. A *for loop* must have at least one statementFor_i. Additionally, a *for loop* may have an initialStatement, a finalStatement, and a statementFor_iPlus1. Any *object* that is linked to a *for loop* via one of these latter relations must have a sameObject relation to an *object* that is linked to a *for loop* via a statementFor_i relation. This is guaranteed by logical constraints imposed on the *for loop* class.

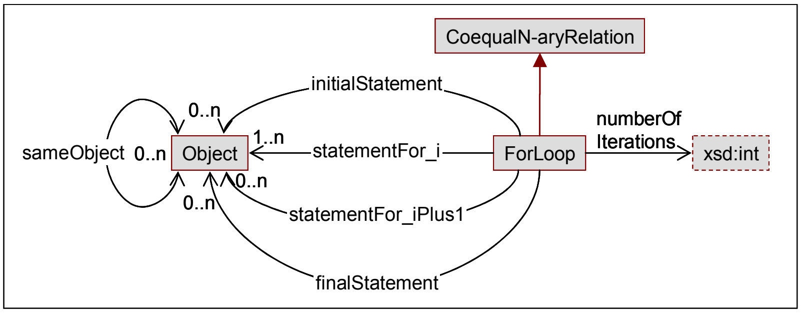

Fig. 4.44: Class diagram of the *loop* design pattern

The relations of the *loop* design pattern are given in Fig. 4.45.

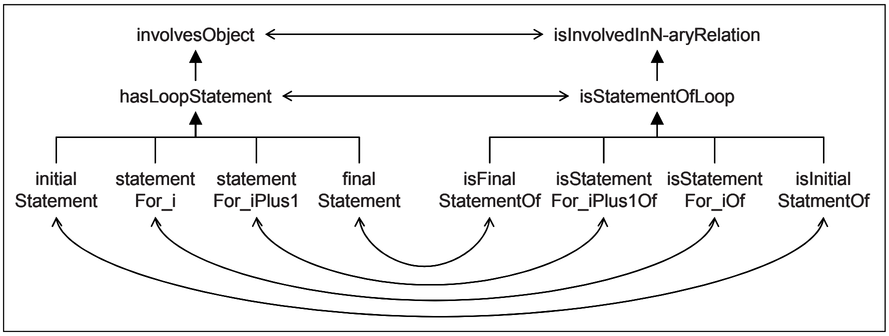

Fig. 4.45: Hierarchy of relations introduced in the *loop* ontology module

4.5.5.1 Usage

The *loop* pattern is used to represent structures that consist of repetitive, interlinked *objects*.
If the structure represented by the *for loop* is part of an *aggregate*, then all *objects* that are linked to the *for loop* should be declared as parts of the *aggregate*.

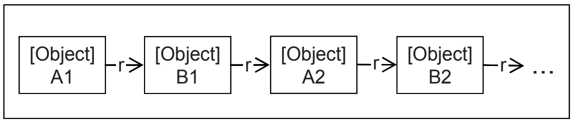

Fig. 4.46: Structure consisting of alternating **A**s and **B**s

Properties (i.e. relations or attributes) that are common to all *objects* of the structure must only be declared once, namely as properties of the individual that linked

to the *for loop* via a statementFor_i relation. Individuals that are linked to the *for loop* via a ininitalStatement, finalStatement, or statementFor_iPlus1 relation should carry only those properties that are specific for the respective iteration.
Note that the *loop* pattern also allows for the representation of structures that consist of alternating elements. An example of such a structure composed of alternating **A**s and **B**s is given in Fig. 4.46. The equivalent *loop* pattern is shown in Fig. 4.47.

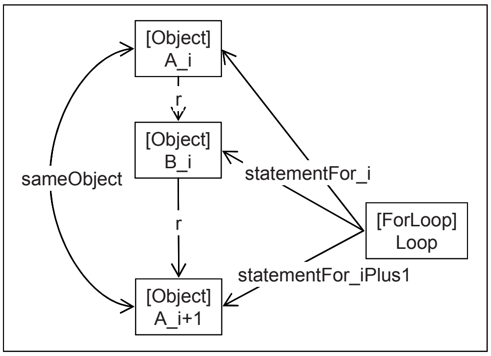

Fig. 4.47: Loop pattern representing the structure displayed in Fig. 4.46

4.5.5.2 Concept Descriptions

Individual concepts of them module *loop* are defined below.

Class Descriptions

For loop
A *for loop* is used to represent structures that consist of repetitive, interlinked *objects*.

Relation Descriptions

finalStatment
Denotes the final statement in a *for loop*.

hasLoopStatement
Subsumes the different statements of a *for loop*.

ininitalStatement
Denotes the initial statement in a *for loop*.

isFinalStatmentOf
Denotes the final statement in a *for loop*.

isIninitalStatementOf
Denotes the initial statement in a *for loop*.

isStatementFor_iOf
Denotes the *objects* that appear in each iteration of a *for loop*.

isStatementFor_iPlus1Of
Denotes the *objects* in the next iteration of a *for loop*.

isStatementOfLoop
Subsumes all the individuals that represent statements in a *for loop*.

sameObject
Identity relation between an *object* involved in a statementFor_i and an *object* that appears in an initialStatement, a finalStatement, or a statementFor_iPlus1.

statementFor_i
Denotes the *objects* that appear in each iteration of a *for loop*.

statementFor_iPlus1
Denotes the *objects* in the next iteration of a *for loop*.

Attribute Descriptions

numberOfIterations
Indicates the number of iterations of a particular *for loop*.

4.6 References

Atkinson C, Kühne T (2002) The role of metamodeling in MDA. In: *Proceedings of the Workshop in Software Model Engineering (in conjunction with UML'02, Dresden, Germany)*. Online available at http://www.meta model.com/wisme-2002/papers/atkinson.pdf. Accessed January 2008.

Biron PV, Permanente K, Malhotra A (2004) *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. Online available at http://www.w3.org/TR/xmlschema-2/. Accessed January 2007.

Black PE, ed. (2004) Data structure. In: *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology. Online available at http://www.itl.nist.gov/div897/sqg/dads/. Accessed January 2009.

Blitz D (1992) Emergent Evolution, Qualitative Novelty and the Kinds of Reality. Springer.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Casati R, Varzi A (1999) *Parts and Places: The Structures of Spatial Representation*. MIT Press.

Clark P, Thompson J, Porter B (2000) Knowledge patterns. In: Cohn A, Giunchiglia F, Selman B (eds.): *KR-2000: Proceedings of the Conference on Knowledge Representation and Reasoning*. Morgan Kaufmann:591-600.

Eggersmann M, Hai R, Kausch B, Luczak H, Marquardt W, Schlick C, Schneider N, Schneider R, Theißen M (2008) Work process models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering Design Processes: From Understanding to Substantial Support*. Springer, Berlin:126–152.

Ferstl OK, Sinz EJ (2001) *Grundlagen der Wirtschaftsinformatik*, Bd. 1. Oldenbourg, München.

Fowler M (1997) *UML Distilled- Applying the Standard Object Modeling Language*. Addison-Wesley.

Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Grüninger M, Fox MS (1995) Methodology for the design and evaluation of ontologies. In: Skuce D (ed.): *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing.*

Haarslev V, Möller R, van der Straeten R, Wessel M (2004) Extended query facilities for Racer and an application to software-engineering problems. In: *Proceedings of the 2004 International Workshop on Description Logics (DL-2004)*:148–157.

Little EG, Rogova GL (2005) *Ontology Meta-Model for building a situational picture of catastrophic events*. In: 8th International Conference on Information Fusion 2005, DOI: 10.1109/ICIF. 2005.1591935.

McLaughlin B, Bennett K (2005) *Supervenience*. Online available at http://plato.stanford.edu/entries/supervenience/. Accessed December 2006.

Morbach J, Yang A, Marquardt W (2007) OntoCAPE – a large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20** (2):147–161.

Morbach J, Hai R, Bayer B, Marquardt W (2008c) Document models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:111–125.

Noy N, Rector A, eds. (2006) *Defining N-ary Relations on the Semantic Web*. W3C Working Group Note, 12 April 2006. Online available at http://www.w3.org/TR/swbp-n-aryRelations/. Accessed November 2007.

Odell JJ (1994) Six different kinds of composition. *Journal of Object-Oriented Programming* **5**(8). Online available at http://www.conradbock.org/compkind.html. Accessed December 2006.

OWL (2002) OWL representation ontology. Online available at http://www.w3.org/2002/07/owl. Accessed October 2007.

Patel-Schneider PF, Horrocks I (2006) *OWL 1.1 Web Ontology Language Overview*. W3C Member Submission, 19 December 2006. Online available at http://www.w3.org/Submission/owl11-overview/. Accessed October 2007.

Racer Systems (2006) *What is Racer Pro?* Webpage, http://www.racer-systems.com/products/racerpro/index.phtml. Accessed December 2006.

Rector A (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Genari J (ed.): *Knowledge Capture 2003*. ACM Press:121-128.

Rector A, ed. (2005) *Representing Specified Values in OWL: "value partitions" and "value sets"*. W3C Working Group Note, 17 May 2005. Online available at http://www.w3.org/TR/swbp-specified-values. Accessed November 2007.

Rector A, Welty C, eds. (2005) *Simple part-whole relations in OWL Ontologies*. W3C Editor's Draft, 11 August 2005. Online available at http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. Accessed November 2007.

Simons P (1987) *Parts: A Study in Ontology*. Oxford University Press.

Smith MK, Welty C, McGuinness DL, eds. (2004) OWL Web Ontology Language Guide. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-guide/. Accessed October 2007.

Stanford Center for Biomedical Informatics Research (2008) The Protégé Ontology Editor and Knowledge Acquisition System. Online available at http://protege.stanford.edu/. Accessed January 2008.

Theißen M, Marquardt W (2008) Decision models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:153–168.

Varzi A (2006) Mereology. In: Zalta EN (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2006 Edition)*. Online available at http://plato.stanford.edu/archives/win2006/entries/mereology/. Accessed January 2007.

5 Upper Level

The partial model **upper_level** is located on the Upper Layer of OntoCAPE. It establishes the fundamental organizational paradigm for the ontology and states the principles governing its design and evolution. The concepts introduced by the **upper_level** partial model are generic in the sense that they are applicable to different domains; thus, the partial model resembles the *meta_model* (Chap. 4) in this respect. Yet unlike the Meta Model concepts, the concepts of the **upper_level** are intended for direct use and will be passed on to the domain-specific parts of OntoCAPE.
As for its function within the ontology, the **upper_level** serves two major purposes: Firstly, it gives a concise and comprehensive overview on OntoCAPE, thus helping a user to find his/her way around the ontology and to understand its major design principles. Secondly, it establishes a framework for the development (and later extension) of the ontology.

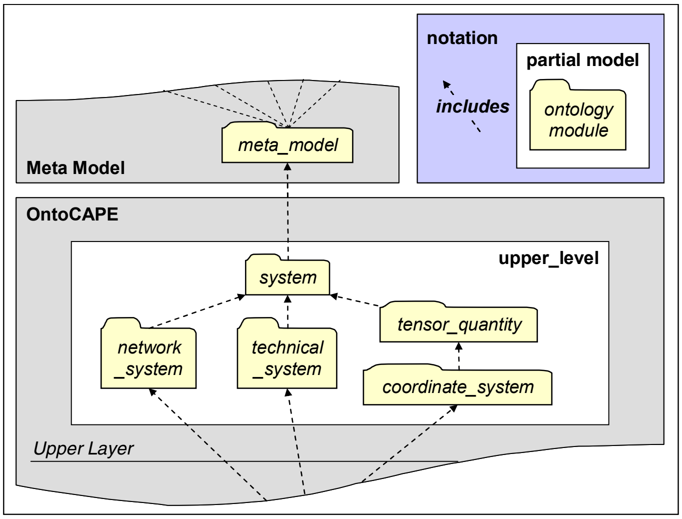

Fig. 5.1: The partial model **upper_level**

The **upper_level** partial model comprises five ontology modules (cf. Fig. 5.1). The module *system* is the most fundamental part of OntoCAPE. Consequently, it is located at the top of the "inclusion lattice" (Gruber and Olsen 1994) that constitutes the ontology. As indicated in Fig. 5.1, the *system* module may import the ontology modules of the *meta_model*, provided that such an import is desired (cf. discussion in Sect. 4.1).

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1_5,
© Springer-Verlag Berlin Heidelberg 2010

The *system* module establishes the fundamental design paradigm according to which the ontology is organized: OntoCAPE is based on *general systems theory*[48] and *systems engineering*[49], which are considered advantageous organizing principles for building large engineering ontologies (e.g., Alberts 1994; Borst 1997; Bayer and Marquardt 2004). The *system* module introduces the constitutive systems-theoretical and physicochemical primitives, such as *system*, *property*, *physical quantity*, *physical dimension*, etc., and specifies their mutual relations.
The remaining modules of the **upper_level** complement the *system* module: The modules *network_system* and *technical_system* introduce two important types of *systems* and their characteristics. The module *tensor_quantity* provides concepts for the representation of vectors and higher-order tensors, while *coordinate_system* introduces the concept of a *coordinate system*, which serves as a frame of reference for the observation of system properties.

5.1 System

5.1.1 Basic Axioms of Systems Theory

The *system* class is the central concept of the *system* module. It denotes all kinds of systems, which may be physical or abstract. The notion of a *system* is defined by the following axioms, which summarize the numerous definitions of the systems concept given in the literature (e.g., von Bertalanffy 1968; Bunge 1979; Patzak 1982; Klir 1985; Gigch 1991):

(1) A *system* interacts with, or is related to, other *systems*.
(2) The constituents of a *system* are again *systems*[50].
(3) A *system* is separable from its environment by means of a conceptual or physical boundary.
(4) A *system* has *properties* which may take different *values*.
(5) The *properties* of a *system* can be explicitly declared or inferred from the properties of its constituent subsystems.

[48] General systems theory is an interdisciplinary field that studies the structure and properties of systems (von Bertalanffy 1968).

[49] Systems engineering can be viewed as the application of engineering techniques to the engineering of systems, as well as the application of a systems approach to engineering efforts (Thomé 1993).

[50] In systems theory, there are divergent views on the nature of system constituents (e.g., Bunge 1979: “A system component may or may not be a system itself.”). Sect. 5.2.3 addresses this issue in greater detail.

The above axioms constitute the basic principles of systems theory, as it is conceptualized in OntoCAPE. They will be revisited in the following sections, which discuss the concrete realization of the systems concept.

5.1.2 Inter-System Relations

Axiom (1) states that *systems* interact with, or are related to, other *systems*. These interactions are modeled by the relation isRelatedTo, which subsumes all kinds of binary relationships[51] between *systems* (cf. Fig. 5.2). The isRelatedTo relation is symmetric to account for the fact that, if *system* **A** is related to *system* **B**, then **B** is related to **A**, as well. Moreover, the relation is declared to be transitive, such that a third *system* **C**, which is explicitly related to **B**, can be inferred to be related to **A**, as well. Additionally, the non-transitive relation isDirectlyRelatedTo is established, which subsumes all direct relations between *systems*.

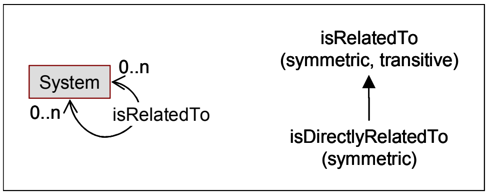

Fig. 5.2: Inter-system relations

5.1.3 Subsystems and Supersystems

For the realization of axiom (2) – the constituents of a *system* are again *systems* – the following concepts are introduced.
Firstly, the transitive relations hasSubsystem and its inverse isSubsystemOf are introduced as specializations of the isRelatedTo relation. They are derived from the aggregation relations hasPart and isPartOf introduced in the Meta Model (partial model **mereology**, cf. Sect. 4.3; their respective definitions are identical, except that their ranges and domains are restricted to *systems*.
Next, the classes *subsystem* and *supersystem* are introduced as subclasses of *system*; they correspond to the generic *parts* and *aggregates* defined in the Meta Model. A necessary and sufficient condition that a system qualifies as a *subsystem* is that the

[51] A class to represent n-ary relations between *systems* is currently not implemented in OntoCAPE.

system is linked to another *system* via an isSubsystemOf relation. Similarly, a *supersystem* is a *system* that has a hasSubsystem relation with some other *system*. In accordance with the mereological theory defined in the partial model **mereology**, a *subsystem* can have *subsystems* of its own, and a *supersystem* may be part of another *supersystem*.

The relation hasDirectSubsystem is established as a means to indicate the direct *subsystems* of a *system*; hasDirectSubsystem is a subrelation of both hasSubsystem and isDirectlyRelatedTo, and it is defined analogously to the hasDirectPart relation introduced in the Meta Model. Similarly, its inverse isDirectSubsystemOf is declared to be a specialization of the isSubsystemOf relation.

A particular *subsystem* may be part of more than one *system*[52]. To indicate a *subsystem*'s unambiguous affiliation to a *supersystem*, the relation isExclusivelySubsystemOf and its inverse isComposedOfSubsystem are to be used. These relations are subrelations of isDirectSubsystemOf and hasDirectSubsystem, respectively; they are special types of the composition relations introduced in the partial model **mereology**. *Systems* that are involved in these relations are named *exclusive subsystem* and *composite system*.

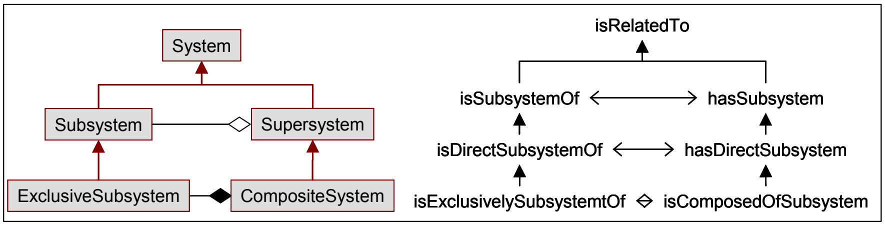

Fig. 5.3: Composition and decomposition of systems

Fig. 5.3 summarizes the classes and relations that represent the (de)composition of *systems*. In analogy to UML notation, we use a line with a white diamond-shaped arrowhead to represent the relations isSubsystemOf and isDirectSubsystemOf; a black diamond-shaped arrowhead indicates the relation isExclusivelySubsystemOf.

Unfortunately, current OWL reasoners scale badly when processing large collections of individuals connected via transitive, inverse relations (Rector and Welty 2005). Hence, the relations hasSubsystem and isSubsystemOf can cause performance problems if applied to large data sets. A possibility to avoid these problems is to employ a single, non-inverse relation, instead. To this end, the unidirectional contains relation is introduced as a replacement for hasSubsytem. Like hasSubsystem, it is a transitive relation; unlike hasSubsystem, it has no inverse counterpart.

[52] A typical example for such a case are the classes *property model* (which models, e.g., the thermodynamic behavior of materials) and *process model* (which represents the mathematical model of a chemical process). These classes, introduced in the partial model *process_model*, are special types of *systems*. A particular *property model* may be a subsystem of different *process models*.

The non-transitive relation containsDirectly is established as a specialization of contains; it is to be used analogously to the hasDirectSubsystem relation (cf. Fig. 5.4).
Aside from the performance considerations, there is another application case for the contains(Directly) relation: It is to be used when only one side of the aggregation relation is of interest, namely the indication of the constituting elements of a *supersystem*; by contrast, the inverse relation (i.e., the affiliation of a *subsystem* to a particular *supersystem*) is of little or no concern in this application case. As an example, consider the relation between the concepts *mixture* and *chemical component*[53]. For the definition of a particular *mixture*, the information about its constituent *chemical components* is essential. However, for the definition of a *chemical component*, it is irrelevant to know of which *mixtures* the *chemical component* is part of. For that reason, the constituents of a *mixture* are indicated by means of the containsDirectly relation.
Note that the contained *systems* are not classified as *subsystems*, as this information is not relevant, as explained above. Only the containing *systems* are classified as *supersystems*.

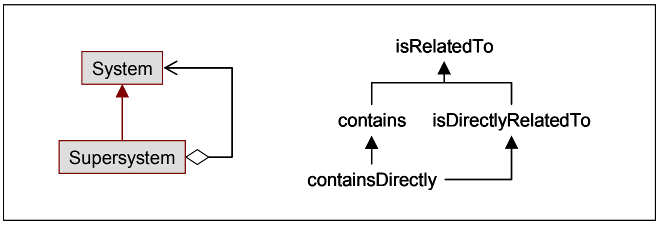

Fig. 5.4: The hasSubsystem relation may be replaced by the contains relation

As a graphical notation for the contains(Directly) relation, we use a line with a white diamond at the one end and an arrowhead at the other end. The diamond indicates the containing *system*, whereas the arrow points towards the contained *system*.
Closing the discussion on system (de)composition, it should be pointed out that some systems theorists (e.g., Bunge 1979) prefer an alternative formulation of axiom (2):

> (2*) A system consists of multiple elements, which may or may not be systems themselves.

Thus, contrary to the original formulation of the axiom, the decomposition of a system into its constituent elements is mandatory, whereas these elements being systems is optional. This alternative version of axiom (2) will be referred to as axiom (2*) hereafter.

[53] *Mixture* and *chemical component* are special types of *systems*, which are introduced in the partial model *substance* (cf. Sect. 7.2).

Fig. 5.5 shows the formal representation of axiom (2*). As can be seen, the representation of axiom (2) must be extended by one additional class (*element*) and two inverse relations (hasElement and isElementOf).

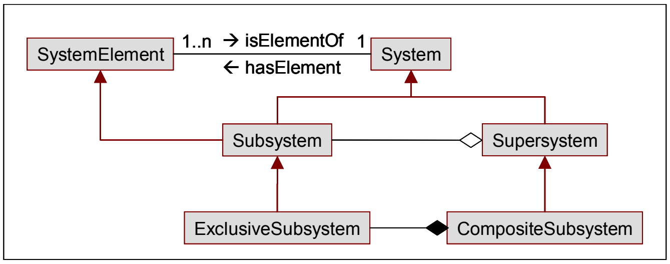

Fig. 5.5: Formal representation of axiom (2*)

There may be application cases where axiom (2*) is more advantageous than axiom (2). However, for the applications of OntoCAPE encountered so far, axiom (2) has proven to be adequate. Furthermore, since axiom (2) can be represented in a more compact way (cf. Fig. 5.3 and Fig. 5.5), it has been preferred over (2*). As demonstrated, axiom (2) can be easily converted into axiom (2*) by adding the abovementioned classes and relations to the ontology, if such an extension is required by some application.

5.1.4 Levels of Decomposition

A (sub)system is considered *elementary* if it is not further partitioned into subsystems. However, it is often impossible to decide definitively if a system is elementary or composite. It might be elementary in one context, but in a different context a further refinement of the system's description might be needed (Bayer 2003). Thus, being elementary is not a static classification.

In OntoCAPE, an *elementary system* is defined as a *subsystem* that (currently) has no *subsystems* of its own.

In an analogous manner, further (de)composition levels of systems can be established:

– A *top-level system* is a *supersystem* that is not a constituent of some other *system*.
– A *first level subsystem* is a *subsystem* that is a direct subsystem of a *top-level system*.
– A *second level subsystem* is a direct subsystem of a *first level subsystem*.
– Etc.

Due to the open world assumption, a DL reasoner cannot infer the membership to the classes *top-level system* and *elementary system* (cf. Sect. 4.3). Thus, membership must be declared explicitly. Once the top (or bottom) of a decomposition hierarchy has been defined that way, the membership to the intermediate decomposition levels can be inferred automatically.

5.1.5 Topological Connectivity of Systems

The relations isConnectedTo and isDirectlyConnectedTo are introduced to describe the topological connectedness of *systems*. They are defined and used just like the homonymic topological relations introduced in the Sect. 4.4, except that their ranges and domains are restricted to *systems* (cf. Fig. 5.6).

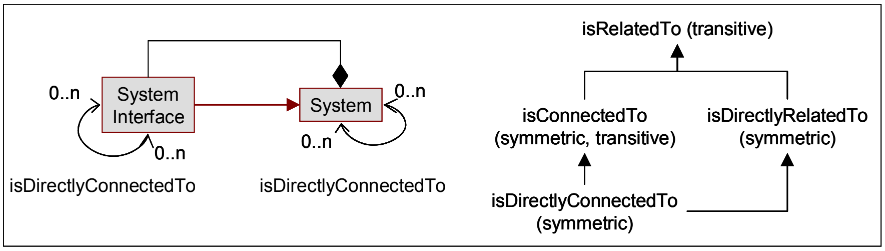

Fig. 5.6: Connectivity of systems

The relation isConnectedTo is symmetric and transitive; it summarizes all types of connections between *systems* (including indirect connectivity). The relation isDirectlyConnectedTo, a non-transitive specialization of isConnectedTo, represents direct connectivity between *systems*.
As explained in the Meta Model, mereological and topological relations exclude each other. Thus, isConnectedTo relations between a *subsystem* and its *supersystem* are prohibited. To enforce this restriction, the following range restrictions are imposed on the isDirectlyConnectedTo relation:

– A *first* (*second* ...) *level system* can only be connected to a *first* (*second* ...) *level system*.
– An *elementary system* can only be connected to an *elementary system*.
– A *top-level system* can only be connected to a *top-level system*.

Hence, connectivity is only allowed if two *systems* are on the same level of decomposition. If these restrictions are violated, the reasoner will produce an error message.
The class *system interface* represents the interfaces through which *systems* are connected to each other. The usage of this class is optional. It is derived from the meta class *connector* and should be utilized analogously.

5.1.6 Model

According to Wüsteneck (1963), a *model* is a system that is used, selected, or produced by a third system to enable the understanding of or the command over the original system, or to replace the original system. Model system and original system share certain characteristics that are of relevance to the task at hand.
Following this definition, the class *model* is introduced as a subclass of *system* (cf. Fig. 5.7). A *system* qualifies as a *model* if it models some other *system* (i.e., having a models relation to another *system* is a necessary and sufficient condition for being subsumed as a *model*). The relation isModeldBy is defined as the inverse of models.

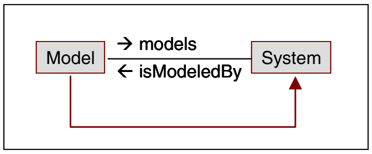

Fig. 5.7: Representation of models

Different types of *models* can be distinguished:

- *Iconic models* resemble the physical object they represent, but are simplified and/or employ a change of scale or materials. Typical examples would be an aircraft mockup used for wind tunnel testing, or a pilot plant that simulates the behavior of an industrial scale plant.
- *Symbolic models* represent the modeled system by means of some symbolic representation. Typical examples are mathematical models or information models.

Iconic models are *technical systems*, as defined in the ontology module *technical_system* (cf. Sect. 5.3). Symbolic models may be considered as *technical systems*, as well; however, this is not necessarily the case. A special class of symbolic models, *mathematical models*, is introduced in the ontology module *mathematical_model* (cf. Sect. 9.1).

5.1.7 Representation of Viewpoints

Systems are often too complex to be understood and handled as a whole. A technique for complexity reduction that is widely used in systems engineering is the adoption of a *viewpoint*[54]. A viewpoint is an abstraction that yields a specification

[54] In the literature, the viewpoint approach is also referred to as "viewing the system from a certain *perspective*" or "considering the system under a particular *aspect*".

of the whole system restricted to a particular set of concerns (IEEE 2000). Adopting a viewpoint makes certain aspects of the system 'visible' and focuses attention on them, while making other aspects 'invisible', such that issues in those aspects can be addressed separately (Barkmeyer et al. 2003).

In the following, the term *aspect system* (Patzak 1982) will be used to denote those aspects about the overall system that are relevant to a particular viewpoint. An aspect system consists of a subset of the components (elements, relationships, and constraints) of the overall system. These components constitute again a system, which is a subsystem of the overall system. Thus, an aspect system is a particular subsystem, which contains only those components of the overall system that are considered under the respective aspect.

In OntoCAPE, an *aspect system* is modeled as a subclass of an *exclusive subsystem* (cf. Fig. 5.8). The type of the respective *aspect system* can be explicitly labeled by an instance of the *aspect* class: To this end, the *aspect system* is linked to that *aspect* via the relation isConsideredUnderAspectOf. Like any *system*, an *aspect system* can be further decomposed – either into 'normal' *subsystems* or into further *aspect systems*. By means of the latter, an *aspect system* can be gradually refined.

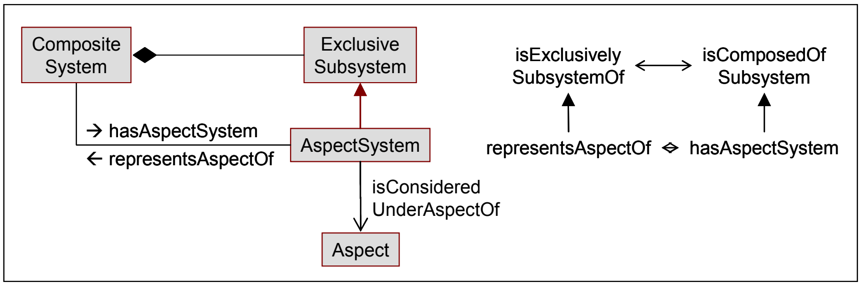

Fig. 5.8: Representation of aspect systems

The relationship between the *aspect system* and the overall (*composite*) *system* is given by the inverse relations representsAspectOf and hasAspectSystem, which are specializations of the composition relations isExclusivelySubsystemOf and isComposedOfSubsystem. These relations can be further refined to indicate the type of the *aspect system*: In the ontology module *technical_system*, for example, the class *system function* is introduced as a special type of an *aspect system* (cf. Sect. 5.1.7); a *system function* is linked to the overall *system* via the relation representsFunctionOf, which is a specialization of representsAspectOf.

Aspect systems play a key role in the organization of the OntoCAPE ontology. They are used to partition complex *systems* into manageable parts, which can be implemented in segregate ontology modules. An example is given in Fig. 5.9. Two *aspect systems*, *process* and *plant*, are shown, which represent a functional and a constitutional view on a *chemical process system* (cf. Sect. 8.1.1). Each *aspect system* is represented in its own ontology module (*process* and *plant*, respectively).

These modules are imported by the ontology module that holds the overall *system* (here, module *chemical_process_system* holding the *chemical process system*). Within their respective ontology modules, *plant* and *process* are modeled as subclasses of *system*; only in the *chemical_process_system* module, they are identified as *aspect systems*. This is achieved by linking *plant* and *process* to the *chemical process system* via the relations representsRealizationOf and representsFunctionOf, respectively, which are specializations of the representsAspectOf relation. Based on this information, a reasoner can infer that *plant* and *process* are special types of *aspect systems*.

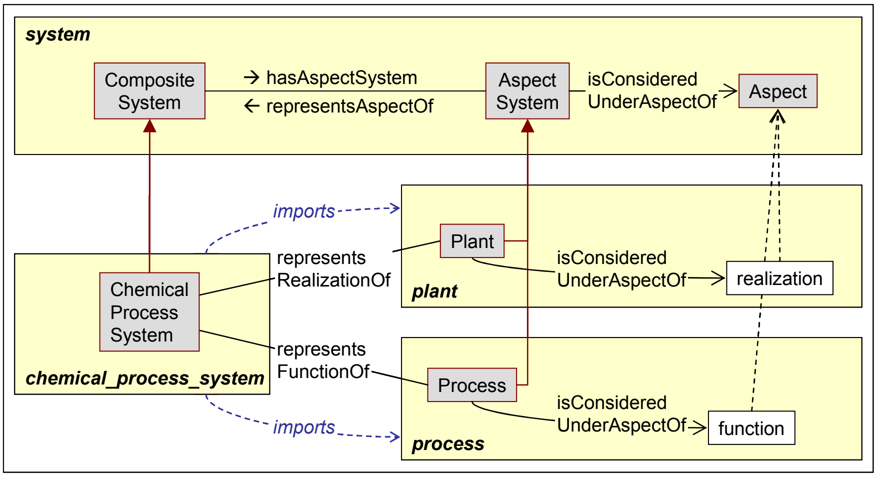

Fig. 5.9: Partitioning of a complex system into manageable parts

The above pattern is universally applied in OntoCAPE. The advantage of this pattern is that the *aspect systems* can be used and maintained independently of the overall *system*.

5.1.8 System Environment

Axiom (3) states that a system is separable from its environment by means of some conceptual boundary (which may or may not coincide with a physical system boundary). The key idea of this axiom is that the scope of a system is uniquely defined, i.e., it is clearly determinable whether a particular object forms part of the system or belongs to the system's environment. In OntoCAPE, the environment of a system can be modeled explicitly, as discussed in the following. The system boundary, on the other hand, is not represented in OntoCAPE, as it is merely an auxiliary construct to mentally demarcate the system from its environment[55].

Generally, the environment of a system includes everything that is not defined as that system (Alberts 1994). Thus, the environment of a given *system* **S** can be defined as the class of all things that are not **S**. Note that such an environment class must be individually defined for each *system*, since the environment concept is relative.

However, the above definition is too broad for practical use. Normally, one is only interested in the *immediate* environment of a *system*, as defined by Bunge (1979):

> "Our definition of the environment of a system as the set of all things coupled with components of the system makes it clear that it is the *immediate* environment, not the *total* one – i.e., the set of all the things that are not parts of the system. [...] we are interested not in the transactions of a system with the rest of the universe but only in that portion of the world that exerts a significant influence on the thing of interest."

In OntoCAPE, the immediate environment of a *system* is even further constrained to those individuals that are again *systems*[56]. Therefore, the environment of a system is defined as follows: The immediate environment of a particular *system* **S** includes all *systems* that (1) are not **S**, (2) are no subsystems of **S**, (3) are no supersystems of **S**, but (4) are directly related to **S**.

Note that the definition excludes subsystems since they form part of **S** and thus cannot be part of the environment of **S**. Supersystems are excluded since this would lead to false conclusions: It would allow a *supersystem* **SupS** of **S** to be part of the environment of **S**. On the other hand, **S** is a subsystem of **SupS** by definition. This would eventually imply that **S** is a subsystem of its environment.

In the formal specification of OntoCAPE, the class *system environment* exemplarily implements this definition for a sample *system* **S**[57].

5.1.9 Properties of Systems

Axiom (4) states that a *system* has *properties* which may take different *values*. In OntoCAPE, the *property* class represents the individual properties (traits, qualities)

[55] "The choice of the system boundary corresponds to a division of the universe of discourse into those parts included in the system under consideration and those belonging to the environment" (Marquardt 1995).

[56] As opposed to *properties*, *values*, etc.

[57] *Implementation advice*: Currently, as of 2008, the reasoner RacerPro is not able to infer the environment of a system correctly. The problem is possibly caused by the allDifferent statement for individuals, which is not evaluated properly. Nevertheless, the definition is correct in principle.

of a *system*, which distinguish the *system* from others. Typical examples would be *size, color*, or *weight*, which are modeled as subclasses of *property*.

The subclasses of *property* represent *general* properties, which exist autonomously (i.e., independent of a particular *system*). The *individual* property of a *system* is modeled by (1) instantiating the respective subclass of *property* and (2) linking that *property* instance to the *system*. For (2), the inverse relations hasProperty and isPropertyOf are to be used (cf. Fig. 5.10). As soon as the *property* instance is linked to a *system*, it represents an inherent quality of that particular *system* and thus must not be assigned to any other *system*. To ensure that a *property* instance is assigned to one *system* instance at most[58], the isPropertyOf relation is declared to be functional.

Subclasses of *property* will be introduced on the lower levels of OntoCAPE to represent properties such as *height*, *volume*, *diameter* etc. These classes can be further specialized in order to clarify the meaning of the respective *property* (e.g., refine *diameter* to *internal diameter*, *nominal diameter*, etc.). However, the refinement must not imply the affiliation to a particular *system*; for example, neither *pipe diameter* nor *vessel diameter* are valid refinements of *diameter*[59]. Instead, the affiliation to a specific *system* is modeled on the instance level by assigning a *property* instance to a *system* instance via the isPropertyOf relation.

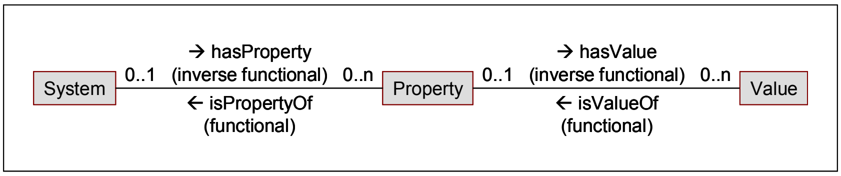

Fig. 5.10: A *system* has *properties* which may take different *values*

A *property* has certain values – for example the *property* '*color*' may take the values 'red', 'green', 'blue', etc. In OntoCAPE, the values of a *property* are represented through the *value* class, which is linked to a *property* via the isValueOf relation and its inverse hasValue, respectively. A *value* is either of qualitative nature (pertaining to *properties* like *color*, *taste*, etc.) or of quantitative nature (pertaining to *properties* like *weight*, *height*, or *temperature*). To avoid ambiguities, the isValueOf relation is declared to be functional; thus, an instance of *value* can be assigned to one *property* instance at most. A *property*, in contrast, may have multiple *values*: Take for example the *temperature* of a solid body – while the existence of this property itself is

[58] Some *properties* are not owned by a particular *system* at all (cf. Sect. 5.1.15)

[59] As an exception to this rule, one may define high-level categorizing *properties* which subsume the *properties* of a specific *system*; for instance, the class *phase system properties* subsumes the various *properties* of a *phase system*. However, these kinds of *properties* are only introduced for organizational purposes and are not to be instantiated for practical use.

invariant (a solid body will always have a temperature), the temperature *values* may change over time.

5.1.10 Backdrop

To distinguish the different *values* of a *property*, the concept of a *backdrop* (Klir 1985) is introduced. Adapting Klir's definition[60] to the terminology of OntoCAPE, a backdrop is some sort of background against which the different *values* of a *property* can be observed. Thus, a backdrop provides a frame of reference for the observation of a *property*. Space and time are typical choices of backdrops.
In OntoCAPE, the *values* of any *property* can act as a backdrop to distinguish the *values* of another *property*. The relation isObservedAgainstBackdrop maps the *values* that are to be distinguished to their respective backdrop *values*. An example is presented in Fig. 5.11: Here, the *values* of the *property* **Time** are used to distinguish the different *values* of the *property* **Temperature**, which arises in the course of an observation[61]. In this particular example, a temperature of 285 Kelvin was observed at the beginning of the observation; after 300 seconds, the temperature had cooled down to 273 Kelvin.

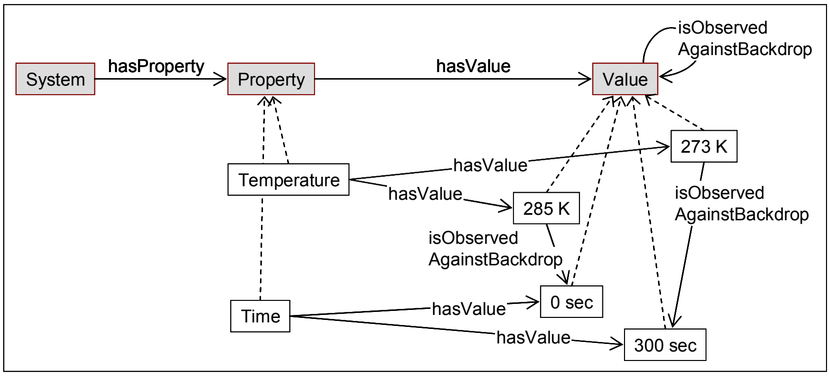

Fig. 5.11: Distinguishing the different *values* of a *property* by means of the backdrop relation

[60] Klir defines a *backdrop* as "any underlying property that is actually used to distinguish different observations of the same attribute […]. The choice of this term, which may seem peculiar, is motivated by the recognition that the distinguishing property […] is in fact some sort of background against which the attribute is observed".

[61] The properties in the example are *physical quantities* (cf. Sect. 5.1.11). Actually, the values of *physical quantities* are represented in a slightly different manner, but the representation is simplified here for the sake of clarity. The exact representation of the example is shown in Fig. 5.11.

The observed *property* and its backdrop *property* may both be owned by the same *system*; however, this is not mandatory. Often, the backdrop *property* is owned by a *coordinate system*, which is introduced in the ontology module *coordinate_system* (cf. Sect. 5.4).
Note that the backdrop concept is relative: A *physical quantity* acting as a backdrop may be observed against another backdrop quantity. Consider for instance a *physical quantity* that is observed against the space coordinate of a moving *system;* the movement of this space coordinate could in turn be measured against the space coordinate of a fixed coordinate system. Another example is given in Fig. 5.12. It extends the above example of temperature measurement (Fig. 5.11) by indicating the time and date of the observation. To this end, one defines a backdrop relation between the starting time of the observation ($t = 0$ sec) and the date-time, given by the time standard UTC (Coordinated Universal Time, cf. Sect. 6.4).

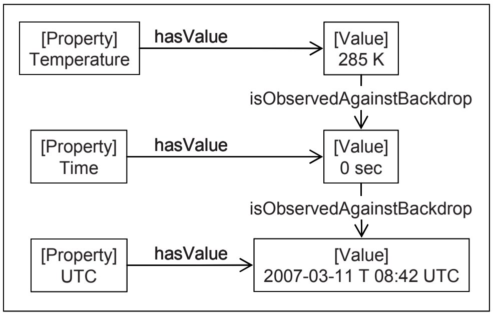

Fig. 5.12: UTC as a backdrop for the starting time of the observation

The indication of backdrop is not mandatory; it can be omitted if it is not important or if it can be recognized from the context. In particular, a backdrop is often superfluous if the *property* can take only a single value. In this case, the property is classified as a *constant property*.

5.1.11 Physical Quantity

The *International Vocabulary of Basic and General Terms in Metrology* defines a *physical quantity* (often abbreviated as a 'quantity') as a "property of a phenomenon, body, or substance, to which a magnitude can be assigned" (VIM 1993). A more extensive definition of the term is given in the *EngMath ontology* (Gruber and Olsen 1994):

> "Physical quantities come in several types, such as the mass of a body (a scalar quantity), the displacement of a point on the body (a vector quantity), […] and the stress at a particular point in a deformed body (a second order tensor quantity). […] Although we use the term "physical quantity" for this

generalized notion of quantitative measure, the definition allows for nonphysical quantities such as amounts of money or rates of inflation. However, it excludes values associated with nominal scales, such as Boolean state and part number […].”

In OntoCAPE, a *physical quantity* is a *property* that has quantifiable values (the latter are represented through the class *quantitative value*, cf. Fig. 5.13). In agreement with the definition given in the EngMath ontology, the class denotes both physical and nonphysical quantities, and it comprises scalars as well as vectors and higher-order tensors. Only *scalar quantities* are considered here; the representation of *vector quantities* and higher-order *tensor quantities* is discussed in Sect. 5.5.

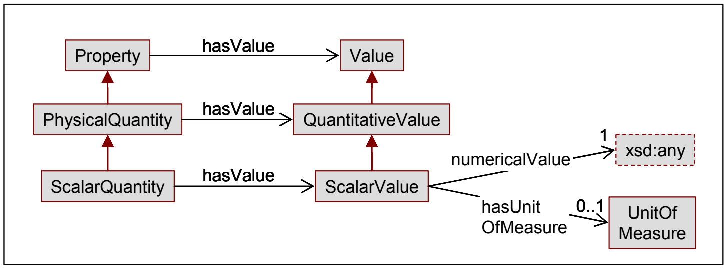

Fig. 5.13: Representing the values of *physical quantities*

Generally, the value of a *scalar quantity* consists of a number and (possibly) a unit of measure. The unit of measure is a particular example of the quantity concerned, which is used as a reference, and the number is the ratio of the value of the quantity to the unit of measure (BIPM 2006).

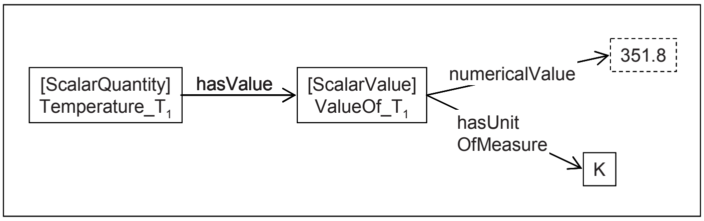

Fig. 5.14: Application example: **Temperature T1** has a value of **351.8 K**

In OntoCAPE, the values of a *scalar quantity* are represented by instances of the class *scalar value*, a subclass of *quantitative value*: The number part of a *scalar value* is expressed by the attribute numericalValue[62], and the unit of measure part is represented by an instance of the *unit of measure* class, which is connected to the *scalar value* via the relation hasUnitOfMeasure (cf. Fig. 5.13). An application exam-

[62] Ordinarily, the values of numericalValue are of type float; however, other XML Schema datatypes are also possible, such as dateTime.

ple is presented in Fig. 5.14, which shows the representation of a temperature value of 351.8 Kelvin. Figure 5.15 shows a more extensive example than Fig. 5.11; it represents the time-dependent measurement of a temperature. The *scalar quantity* **Time** acts as a backdrop to distinguish the different values of the *scalar quantity* **Temperature**.

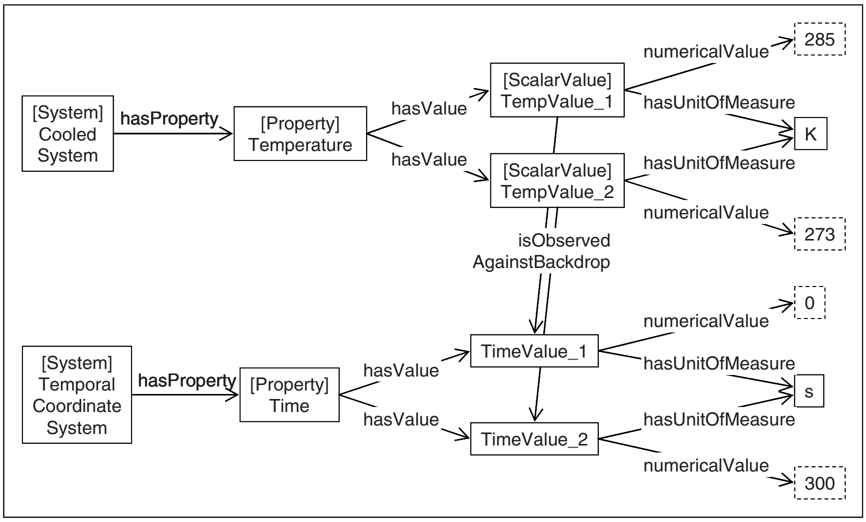

Fig. 5.15: Application example: Temperature measurement with multiple values

5.1.12 *Physical Dimension*

By convention, *physical quantities* are organized in a system of dimensions (BIPM 2006). In such systems, each *physical quantity* has exactly one associated *physical dimension*. A typical example would be the dimension of length, which can be associated with such *physical quantities* as *height*, *thickness*, or *diameter*.

In OntoCAPE, dimensions are modeled by the class *physical dimension*. A particular instance of *physical dimension* can be assigned to both a *physical quantity* and a *unit of measure* via the relation hasDimension (cf. Fig. 5.16). For instance, both the *scalar quantity* '*radius*' and the *unit of measure* '**meter**' have the dimension of **length**.

Physical dimensions serve two functions in OntoCAPE:

(1) *Physical quantities* of the same *physical dimension* share certain characteristics; for instance, their *scalar values* relate to the same set of *units of measure*. Thus, the

concept of *physical dimension* may be used to identify *physical quantities* of the same kind[63] and to differentiate those from other kinds of *physical quantities*.
(2) According to the conceptualizations stated so far, arbitrary *units of measure* can be assigned to the *scalar value* of a particular *scalar quantity*. Now, the *physical dimension* provides a means to constrain the set of possible *units of measure* for a given quantity. To this end, one needs to implement[64] the following constraint:

A *unit of measure* that is assigned to the *scalar value* of a *scalar quantity* must have the same *physical dimension* as the *scalar quantity*.

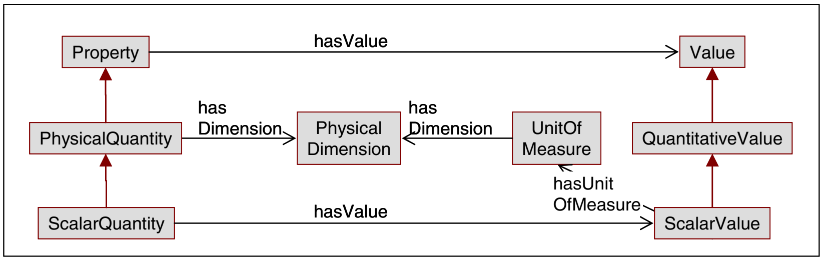

Fig. 5.16: Physical dimensions

On the basis of this constraint, the consistency of unit of measure assignment and conversion can be checked. For example, a **meter** is a valid *unit of measure* for measuring the *scalar value* of a *radius*, as both *radius* and **meter** have the dimension of **length**. Similarly, **meters** can be converted into **feet**, as both *units of measure* have the same dimension.

[63] The International Vocabulary of Basic and General Terms in Metrology (VIM 1993) defines 'quantities of the same kind' as "quantities that can be placed in order of magnitude relative to one another". While it is true that quantities of the same kind must have the same physical dimension, the opposite is not true, i.e., having the same physical quantity is a necessary, but not a sufficient condition for being of the same kind. For example, moment of force and energy are, by convention, not regarded as being of the same kind, although they have the same dimension, nor are heat capacity and entropy (VIM 1993).

[64] In principle, the constraint could be formulated in the OWL modeling language; however, such an implementation would be quite exhausting, as the constraint would have to be formulated individually for each *scalar quantity*. Alternatively, the constraint can be implemented through a single, generic rule, which applies to all quantities. Rules do not form part of current OWL, but can be formulated on top of the language. The latter approach is taken in OntoCAPE.

5.1.13 Qualitative Value

Obviously, not all *properties* are *physical quantities*. The *values* of *properties* like ‘color’ or ‘flavor’ are not (numerically) quantifiable. Instead, such *values* are represented by means of the class *qualitative value*, a subclass of *value* (cf. Fig. 5.17).

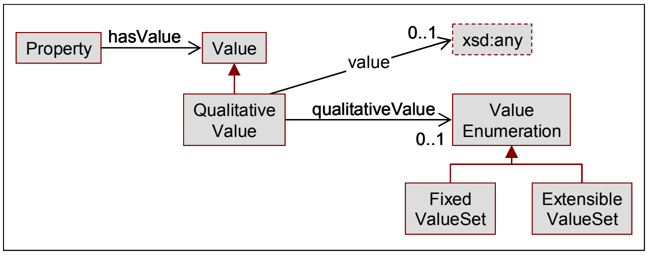

Fig. 5.17: Representation of qualitative values

The actual value of a *qualitative value* can be specified in two alternative ways: either by means of the attribute value, which accepts any string input, or by referring to an instance of the class *value enumeration* via the relation qualitativeValue. A *value enumeration* defines a (finite) set of possible values, which may be assigned to different *qualitative values*. The *value enumeration* class is derived from the meta class *feature space* and can be either a *fixed value set* or an *extensible value set*:

- A *fixed value set* is a specialization of the meta class *value set*. It is uniquely defined by an exhaustive enumeration of its instances. Thus, the number of possible values is fixed.
- An *extensible value set* is a specialization of the meta class *non-exhaustive value set*. Unlike a *fixed value set*, it is not defined by an (exhaustive) enumeration of its instances. Thus, the number of possible values may change at run time.

Like every other *value*, a *qualitative value* can be related to a backdrop *value*. Fig. 5.18 provides the example of a chameleon, whose skin color is observed against a temporal backdrop.

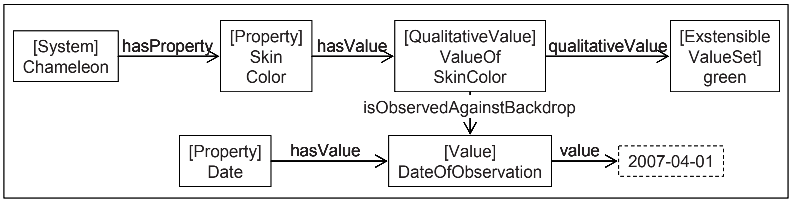
[System] Chameleon
hasProperty
[Property] Skin Color
hasValue
[QualitativeValue] ValueOf SkinColor
qualitativeValue
[Exstensible ValueSet] green
isObservedAgainstBackdrop
[Property] Date
hasValue
[Value] DateOfObservation
value
2007-04-01

Fig. 5.18: Application example of a qualitative value

At first sight, the representation of a qualitative value may seem unnecessarily complicated as it requires an instantiation of both the *qualitative value* class and the *value enumeration* class. Yet both classes are required for the complete specification of the qualitative value: While the *value enumeration* class represents the actual value, the *qualitative value* class serves the function of correlating the actual value with the corresponding backdrop value. A combination of these two functions into a single class is not possible, since the instances of *value enumeration* must not be the origin of a relation (cf. the discussion on feature values in the Meta Model). However, in cases where the specification of a backdrop is not required, the value representation can be simplified, as will be explained in the following section.

5.1.14 The hasCharacteristic Relation

Generally, the characterization of a *system* through *properties* and their *values* is fairly complex, requiring the concatenation of several concepts: First, the *property* class must be instantiated and linked to the *system* via a hasProperty relation; only then can the *value* be specified and assigned to the *property* by means of the hasValue relation. Such a 'chain of concepts' is indispensable for representing *properties* that take multiple *values*, as explained in the previous sections. However, in the case of a *constant property* having only a single *value*, the function of the *constant property* is reduced to that of a binary relation relating the *value* to the *system*. Hence, one may use a shorthand notation instead. To this end, the relation hasCharacteristic is introduced. Via this relation, the *values* of *constant properties* can be directly assigned to a *system*, thus substituting the *constant property*.

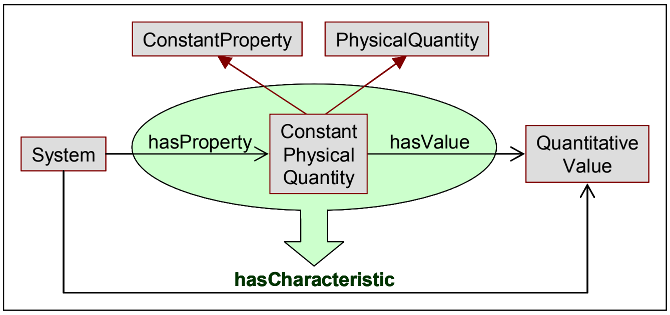

Fig. 5.19: Shorthand notation for constant *physical quantities*

Two cases must be distinguished:

- If the *constant property* is a *physical quantity*, hasCharacteristic replaces the concepts hasProperty, *physical quantity*, and hasValue (cf. Fig. 5.19).

– If the *constant property* has a *qualitative value*, the relation additionally substitutes the concepts *qualitative value* and the relation qualitativeValue, thus referring directly to the *value enumeration* (cf. Fig. 5.20).

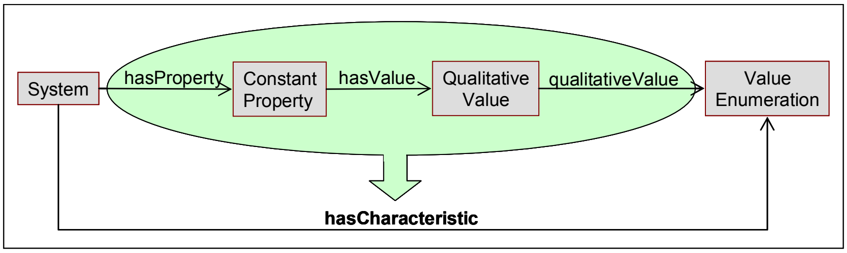

Fig. 5.20: Shorthand notation for *constant properties* with *qualitative values*

Finally some remarks on the usage of the introduced primatives:

– Just like the *property* class can be specialized to represent specific types of *properties*, the hasCharacteric relation needs to be specialized to substitute these *properties*. For instance, to replace the *property 'height'*, the relation hasHeight may be introduced as a specialization of hasCharacteristic.
– Specializations of hasCharacteristic may be utilized to implicitly define polyhierarchies of classes (cf. Sect. 4.2). In this case, the utilized relation should be declared to be a specialization of both the relation hasCharacteristic and the meta relation isOfType.
– The hasCharacterstic relation allows linking a single *value* instance to different *system* instances. This is exploited to relate the *value* of a *physical constant* to different *systems* (cf. Sect. 5.1.15).

5.1.15 Physical Constant

A *physical constant* is a special type of a *constant property* with a fixed (*scalar*) *value*. It is defined as a *physical quantity*, the *value* of which is believed to be both universal in nature and invariant over time. Examples are the elementary charge, the gravitational constant, Planck's constant, and the speed of light in the vacuum. Such specific constants are modeled as instances of the *physical constant* class.
Due to its universal nature, a *physical constant* cannot be owned by a specific *system* and thus must not be assigned to a *system* instance via the hasProperty relation. Instead, the hasCharacteristic relation is used to relate the *value* of the *physical constant* to a *system*. That way, the *physical constant* itself remains independent.

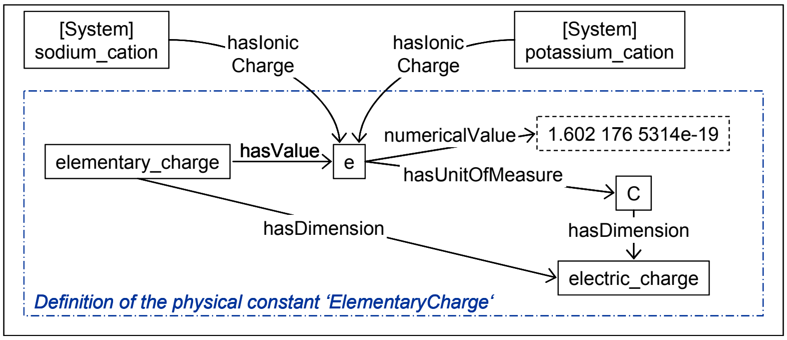

Fig. 5.21: Modeling of the elementary charge

Exemplarily, Fig. 5.21 illustrates the modeling of the **elementary_charge** as an instance of *physical constant*. The **elementary_charge** has a *physical dimension* of **electric_charge**; its *value* **e** equals 1.6021765314e-19 coulomb. By means of the relation hasIonicCharge (a specialization of hasCharacteristic), **e** can be assigned to different *systems*, such as the **sodium_cation** or the **potassium_cation**.

5.1.16 Internal and External Properties

According to axiom (5), not all the *properties* of a *system* need to be declared explicitly. Instead, they can be represented as *properties* of its constituent *subsystems*. We call those *properties* of a *system* that are explicitly assigned to the *system* the 'external *properties*' of the system. Accordingly, the 'internal *properties*' of a *system* are the external *properties* of its constituent *subsystems*.

The internal *properties* of a *system* can be inferred from the external *properties* of its *subsystems* by means of a reasoner. To this aim, one needs to define a query class, which subsumes the (external) *properties* of all *systems* that are subsystems of a given *system*. Such a query class must be individually defined for each *system* instance. An exemplary query class named '*internal properties*' has been implemented in the formal specification of this ontology module. The query class retrieves the internal *properties* of a sample *system* **S**[65].

[65] A system can have both internal and external *properties* of the same type. For example, consider a *phase system*, which is composed of two *single phases*. Both the overall *phase system* and the two *single phases* have a *property* of type *density*. However, their meanings are different: The external *property* of the *phase system* represents the (averaged) density of overall system, whereas the internal *properties* represent the densities of the constituent liquid phase and vapor phase.

5.1.17 Property Set

A *property set* constitutes an (unordered) collection of *properties*, which may be of different types. The *properties* contained in a *property set* are identified via the relation comprisesDirectly, which is a specialization of the transitive relation comprises. These relations are defined analogously to the contains(Directly) relation between *supersystems* and *subsystems*, yet with their ranges and domains restricted to *properties*. Consequently, the comprises(Directly) relation is depicted by the same symbol as the contains(Directly) relation: a white diamond with directed arrow (Fig. 5.22).

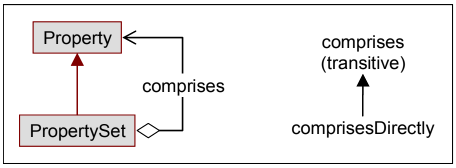

Fig. 5.22: Property set

A *property set* is itself a *property*; thus, a *property set* may comprise other *property sets*. However, a *property set* cannot have a *value* of its own.

5.1.18 Concept Descriptions

Individual concepts of the module *system* are defined below.

Class Descriptions

Aspect
An *aspect* represents a particular viewpoint of a *system*. An instance of the *aspect* class explicitly denominates that viewpoint.

Aspect system
An *aspect system* is an *exclusive subsystem* that contains those system components, relationships, and constraints that are of relevance to a particular *aspect*.
Formal definition: An *aspect system* is an *exclusive subsystem* that is considered under some *aspect*.

Composite system
A *composite system* is a *system* that is composed of other *systems*.
Formal definition: A *composite system* is composed of some *systems*.

Constant property
A *constant property* is a *property* that has exactly one *value*.

Elementary system
An *elementary system* is a *subsystem* that cannot be further partitioned into *subsystems*. Formal definition: An *elementary system* is a *subsystem* that is not a *supersystem*.

Exclusive subsystem
An *exclusive subsystem* is a direct subsystem of a *composite system*; it cannot be a direct subsystem of any other *system*.
Formal definition: An *exclusive subsystem* is exclusively a subsystem of some *system*.

Extensible value set
An *extensible value set* is a *value enumeration* which, unlike a *fixed value set*, is not defined by an (exhaustive) enumeration of its instances. Thus, the number of possible values may change at run time.

First-level subsystem
A *subsystem* at the first level of decomposition.
Formal definition: A *subsystem* that is a direct subsystem of a *top-level system*.

Fixed value set
A *fixed value set* is a *value enumeration* that is defined by an exhaustive enumeration of its instances. Thus, the number of possible values is fixed.

Internal properties
The '*internal properties*' of a *system* are the *properties* of its constituent *subsystems*. They can be specified by means of a query class and thus inferred by a reasoner. Such a query class must be defined individually for each *system* instance. The query class '*internal properties*' exemplarily demonstrates this approach for a sample *system* **S**.
Formal definition: The *internal properties* of the *system* instance **S** are equivalent to the *properties* of the *subsystems* of **S**.

Model
A *model* is a system that is used to enable the understanding of or the command over the original system, or to replace the original system. Model system and original system share certain characteristics that are of relevance to the task at hand (Wüsteneck 1963).
Formal definition: A *model* is a *system* that models some other *system*.

Physical constant
A *physical constant* is a *scalar quantity*, the *value* of which is believed to be both universal in nature and invariant over time. Examples are the elementary charge, the gravitational constant, Planck's constant, and the speed of light in the vacuum.

Physical dimension
A *physical dimension* is a characteristic associated with *physical quantities* and *units of measure* for purposes of organization or differentiation. **Mass**, **length**, and **force** are exemplary instances of *physical dimension*.

Physical quantity
A *physical quantity* is a *property* that has quantifiable *values*. The concept includes scalars as well as vectors and higher-order tensors. Moreover, it comprises both physical quantities, such as mass or velocity, and nonphysical quantities, such as amount of money or rate of inflation.
Formal definition: A *physical quantity* is a *property* that has a *physical dimension*.

Property
The *property* class represents the individual properties (traits, qualities) of a *system*, which distinguish the *system* from others. Typical examples are *size, color*, or *weight*, which are modeled as subclasses of *property*.

Property set
A *property set* constitutes an (unordered) collection of *properties*, which may be of different types.
Formal definition: A *property set* is a *property* that directly comprises some *properties*.

Qualitative value
A *qualitative value* is a *value* that is not (numerically) quantifiable.

Quantitative value
A *quantitative value* is the value of a *physical quantity*.

Scalar quantity
A *scalar quantity* is a scalar-valued *physical quantity*.

Scalar value
A *scalar value* is the value of a *scalar quantity*.

Second-level subsystem[66]
A *subsystem* at the second level of decomposition.
Formal definition: A *subsystem* that is a direct subsystem of a *first-level subsystem*.

Subsystem
A *subsystem* is a *system* that is a constituent of another *system*.
Formal definition: A *subsystem* is a *system* that refers to another *system* via the isSubsystemOf relation.

Supersystem
A *supersystem* is a *system* that has some constituent *subsystems*.
Formal definition: A *supersystem* is a *system* that refers to another *system* via the hasSubsystem relation.

System
The *system* class denotes all kinds of systems, which may be physical or abstract.

[66] This concept simply demonstrates that second, third, fourth, level subsystems can be defined in an analogues manner to the *first-level subsystem*, if required.

System environment
The *immediate environment* of a given *system* **S** consists of all *systems* that are directly related to **S**. It can be specified by means of a query class. As the environment concept is relative, such a query class must be defined individually for each *system* instance. The query class *system environment* exemplarily demonstrates the approach for sample *system* **S**.
Formal definition: The immediate environment of the *system* instance **S** includes all *systems* that (1) are not **S**, (2) are not subsystems of **S**, (3) are directly related to **S**.

System interface
The class *system interface* represents the interface through which a *system* can be connected to another *system*.

Top-level system
A *top-level system* is a *supersystem* that is not a constituent of some other *system*.
Formal definition: A *top-level system* is a *supersystem* that is not a *subsystem*.

Unit of measure
A *unit of measure* is a standard measure for the *scalar value* of *physical quantity*, which has been adopted by convention.

Value
The *value* class denotes the different values of a *property*.

Value enumeration
A *value enumeration* specifies the (finite) set of possible values of a *qualitative value*.
Formal definition: A *value enumeration* is either a *fixed value set* or an *extensible value set*.

Relation Descriptions

comprises
The relation comprises indicates the members of a *property set*.

comprisesDirectly
The relation comprisesDirectly indicates the direct members of a *property set*.

contains
The contains relation constitutes an alternative to the hasSubsystem relation. It should be used instead of hasSubsystem

- if the hasSubsystem relation causes performance problems, or
- if only one side of the aggregation relation is of interest, namely the indication of the constituting elements of a *supersystem*.

containsDirectly
The relation containsDirectly is an alternative to the hasDirectSubsystem relation. It should be used instead of hasDirectSubsystem

- if the hasDirectSubsystem relation causes performance problems, or
- if only one side of the aggregation relation is of interest, namely the indication of the direct constituents of a *supersystem*.

hasAspectSystem
The relation hasAspectSystem designates the *aspect systems* of a *system*.

hasCharacteristic
The hasCharacteristic relation constitutes a shorthand notation for the specification of a *constant property* and its *value*.

hasDimension
The relation hasDimension specifies the *physical dimension* of a *physical quantity* or a *unit of measure*.

hasDirectSubsystem
The relation hasDirectSubsystem refers from a *supersystem* to its direct *subsystem*.

hasProperty
The relation hasProperty indicates the *properties* of a *system*.

hasSubsystem
The relation hasSubsystem denotes the relation between a *supersystem* and its *subsystem*.

hasUnitOfMeasure
The relation hasUnitOfMeasure establishes the *unit of measure* of a *scalar value*.

hasValue
The hasValue relation designates the *values* of a *property*.

isBackdropOf
The isBackdropOf relation states that the *value* serves as a backdrop for the observation of some other *value*.

isComposedOfSubsystem
The relation isComposedOfSubsystem indicates the non-sharable, direct *subsystem* of a *supersystem*.

isConsideredUnderAspectOf
The relation isConsideredUnderAspectOf indicates the type of an *aspect system* by referring to an instance of the *aspect* class.

isConnectedTo
The relation isConnectedTo represents topological connectivity between *systems*.

isDirectlyConnetedTo
The relation isDirectlyConnectedTo denotes the direct topological connectedness of two *systems.*

isDirectlyRelatedTo
The relation isDirectlyRelatedTo subsumes all kinds of direct inter-system relations.

isDirectSubsystemOf
The relation isDirectSubsystemOf links a *subsystem* to its direct *supersystem*.

isExclusivelySubsystemOf
The relation isExclusivelySubsystemOf links a non-sharable *subsystem* to its direct *supersystem*.

isModeledBy
The relation isModeldBy points from a modeled *system* to its *model*.

isObservedAgainstBackdrop
The isObservedAgainstBackdrop relation maps a *value* against a backdrop *value*.

isPropertyOf
The relation isPropertyOf links a *property* instance to a *system* instance.

isRelatedTo
The relation isRelatedTo subsumes all kinds of inter-system relations.

isSubsystemOf
The relation isSubsystemOf refers from a *subsystem* to its *supersystem*.

isValueOf
The relation isValueOf assigns a *value* to a *property*.

models
The relation models links a *model* to the modeled *system*.

qualitativeValue
The relation qualitativeValue specifies the actual value of a *qualitative value*.

representsAspectOf
The relation representsAspectOf links an *aspect system* to its respective *system*.

Attribute Descriptions

numericalValue
The attribute numericalValue specifies the number part of a *quantitative value*.

value
The value attribute holds the actual value of a *qualitative value*.

5.2 Network System

The ontology module *network_system* introduces a structured representation for complex *systems*, which is applicable in such different domains as biology, sociology, and engineering. The common strategy of these disciplines is to represent the system as a *network*. In this context, a network is understood as a modular structure that "is determined on hierarchical ordered levels by coupling of components and linking elements" (Gilles 1998). Thus, the representation of network systems calls for two different mechanisms: the mereological decomposition of systems and the topological ordering of the system components.
The concepts required for the mereological decomposition of *systems* are provided by the ontology module *system*, which allows for the structuring of *systems* into *subsystems* across multiple levels of hierarchy (cf. Sect. 5.1.4). Hence, what remains to be done is to introduce concepts for the topological organization of the system components. To this aim, we adopt the design pattern for the representation of graphs that was defined in the ontology module *topology* of the Meta Model (cf. Sect. 4.4). Hence, *network system* is introduced as a specialization of *system* incorporating mereological as well as topological considerations. According to the design pattern, graphs are represented through *nodes* and connecting *arcs*, where an *arc* may or may not be directional. Additionally, *ports* and *connection points* may be used to further specify the connectivity between *nodes* and *arcs*.
Applying this design pattern to the representation of network systems, two special types of *systems*, *device* and *connection*, are introduced. Hence, a *network system* is composed of at least one *device* and one *connection* as shown in Fig. 5.23. *Device* and *connection* correspond to the meta classes *node* and *arc*, respectively, and are defined equivalently. Additionally, a *directed connection* is established as a subclass of *connection*.

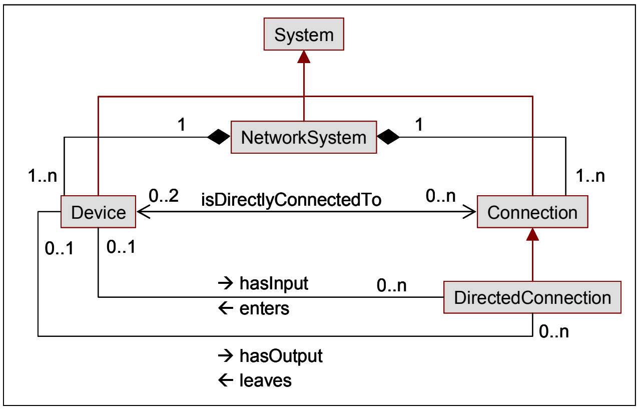

Fig. 5.23: Connectivity of *devices* and *connections*

The relation isDirectlyConnectedTo, previously established in the *system* module (cf. Sect. 5.1.5), is utilized to couple a *connection* with a *device*. For linking a *directed connection* to a *device*, the relations enters and leaves are to be used, which are defined analogously to the Meta Model (cf. Fig. 5.24).

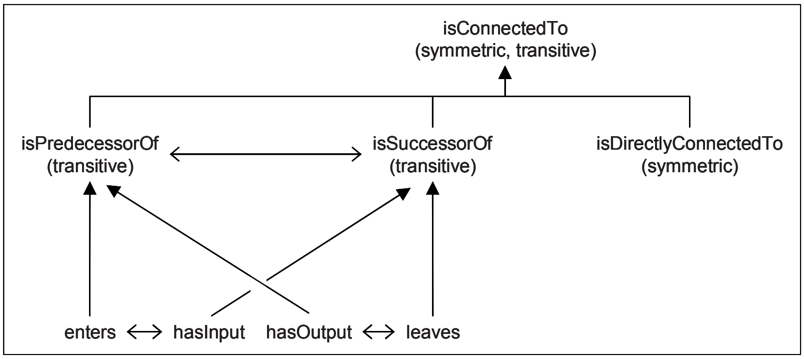

Fig. 5.24: Hierarchy of topological relations

So far, we have considered only such *connections* that are connected to exactly two *devices*. Another special case of *connection* is the single-edge *connection*, which is directly connected to only a single *device*. We denote such a class as *environment connection* because it represents the connectivity of a *network system* with its (not explicitly specified) *environment* (cf. Fig. 5.25).

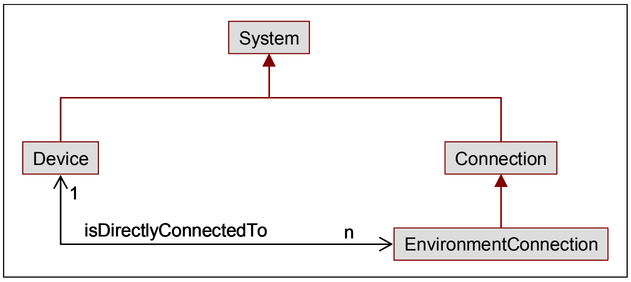

Fig. 5.25: Connectivity of *environment connection*

Ports and *connection points* are introduced as special types of *system interfaces* (Fig. 5.26). Just like in the Meta Model, *ports* and *connection points* represent the interfaces of the *devices* and *connections*. Their characteristics need to match in order to realize a valid coupling (cf. Sect. 4.4.3).

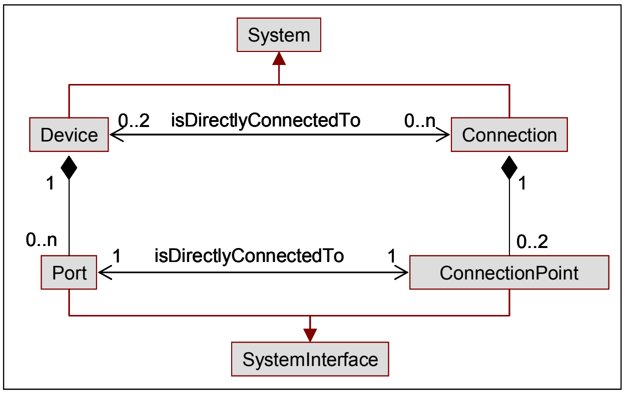

Fig. 5.26: *Ports* and *Connection points*

The decomposition of *devices* and *connections*, depicted in Fig. 5.27, is governed by the following regulations:

- *Devices* can only have direct subsystems of type *device*, *connection*, or *port*.
- *Connections* can only have direct subsystems of type *device*, *connection*, or *connection point*.
- If a *device* is decomposed into a number of sub-*devices*, then these sub-*devices* must be connected by *connections*. Thus, a *device* needs to be decomposed into two *devices* and one intermediate *connection*, at least.

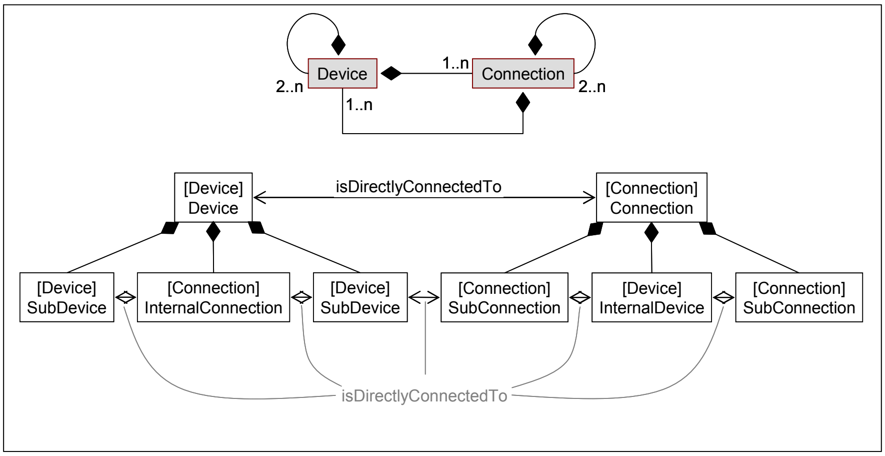

Fig. 5.27: Decomposition of devices and connections

Similarly, if a *connection* is decomposed into sub-*connections*, then there must be *devices* in between the sub-*connections*. Thus, a *connection* needs to be decomposed into two *connections* and one intermediate *device*, at least.

The aforementioned regulations are derived from the decomposition rules for *nodes* and *arcs* established in the Meta Model. For details on this issue, refer to Sect. 4.4.3. Finally, we define a *network system* as a *system* that is composed of some *devices* and *connections*.

5.2.1 Usage

A large number of real-world systems can be modeled as *network systems*: technical systems (Alberts 1994; Marquardt 1996; Marquardt et al. 2000), physico-chemical systems (e.g., Marquardt 1992a; Marquardt 1994b, Marquardt 1995; Gilles 1998), biological systems (e.g., Mangold et al. 2005), economic systems (e.g., Andresen 1999), social systems (e.g., Bunge 1979), and others. Generally, the *devices* are the crucial elements of a *network system* and hold the major functionality, while the *connections* represent the linkages between the *devices*.
To enhance the understanding for the applicability of network systems, three examples of describing real-world systems as network systems are discussed subsequently:

– Marquardt (1992a) and Gilles (1998) propose a framework for the development of mathematical models for physico-chemical systems, wherein *devices* and *connections* represent the individual model building blocks. Within the modeling framework, only the *devices* have the capability for the accumulation and/or change of extensive physical quantities, such as energy, mass, and momentum. The *connections*, on the other hand, describe the fluxes of quantities that are interchanged between the *devices*; different types of fluxes can be modeled this way – of matter (e.g., material flow through a pipe), energy (e.g., heat conduction through a wall), momentum (e.g., shock wave in a fluid medium).
– Network systems are particularly suitable for the representation of process flowsheets. For example, consider a Block Flow Diagram (BFD), which is used to specify the conceptual design of a chemical process: The individual process units (unit operations) can be considered as *devices*, and the material and energy streams that are exchanged between the units can be represented as *connections*. Another example is the Piping & Instrumentation Diagram (P&ID) applied in basic and detail engineering: Here, the apparatuses and machines are modeled as *devices*, while *connections* represent the pipes (for materials and utilities) and the power supply lines.
– In the area of control theory, the control components (controller, sensor, controlled system,…) can be modeled as *devices*, while the *connections* represent the signal lines that transmit information between the control components (Bayer et al. 2001).

5.2.2 Concept Descriptions

Individual concepts of the module *network_system* are defined below.

Class Descriptions

Connection
Connections are those elements of a *network system* that represent the linkages between the *devices*.

Connection point
A *connection point* represents the interface through which a *connection* can be connected to the *port* of a *device*. *Connection points* may have certain attributes that further specify the type of connection. *Connection points* are subsystems of the corresponding *connection* or *directed connection*, respectively.

Device
Devices are the crucial elements of a *network system*, holding the major functionality.

Directed Connection
Directed connection is a specialization of *connection* and represents likewise the connecting element between *devices*. However, the use of *directed connection* implies a directed interconnection.

Environment Connection
Environment connection is a specialization of *connection* and represents a single-edge connection to exactly one *device*. Thus, special connections like system inputs or outputs may be represented for not explicitly defined environments.

Network system
A *network system* is a *system* that is composed of *connections* and *devices*.
<u>Formal definition</u>: A *network system* is a *system* that is composed of some *connections* and some *devices*.

Port
Ports represents the interfaces through which *devices* are connected to *connections*.
<u>Formal definition</u>: A *port* may have certain attributes that characterize the type of the connection.

Relation Descriptions

enters
The relation enters interconnects an outgoing *directed connection* to its target *device*.

hasInput
The relation hasInput connects a *device* to an incoming *directed connection*.

hasOutput
The relation hasOutput connects a *device* to an outgoing *directed connection*.

isSuccessorOf
The relation isSuccessorOf identifies all *devices* and *directed connections* that are successors of the considered one.

isPredecessorOf
The relation isPredecessorOf identifies all *devices* and *directed connections* that are predecessors of the considered one.

leaves
The relation leaves connects an outgoing *directed connection* to its source *device*.

sameAs
The relation denotes a correspondence between a *connection* and its placeholder in a decomposition hierarchy.

5.3 Technical System

The ontology module *technical_system* introduces the class *technical system* as a special type of a *system* which has been developed through an (engineering) design process. The criterion to qualify as a *technical system* is "to be designed in order to fulfill some required function" (Bayer 2003). Thus, the *technical system* concept may denote all kind of technical artifacts, such as chemical plants, cars, computer systems, or infrastructure systems like a sewage water system. But also non-technical artifacts like chemical products and even non-physical artifacts, such as software programs or mathematical models, can be considered as *technical systems*.
For a comprehensive description of a *technical system*, five designated viewpoints are of major importance (Bayer 2003): the system *requirements*, the *function* of the system, its *realization*, the *behavior* of the system, and the *performance* of the system. These five viewpoints are explicitly modeled in this ontology module, as will be explained in the following sections: In Sects. 5.3.1 to 5.3.4, the precise meaning of the respective viewpoints will be clarified. In the subsequent Sect. 5.3.5, the implementation of these viewpoints as specialized *aspect systems* (cf. Sect. 5.1.7) will be described. Lastly, Sect. 5.3.6 discusses the interrelations between the different *aspect systems*.
Before going into details, it should be mentioned that the concepts provided by this module may be used to describe the 'as-is' state (i.e., the current status) of a

technical system as well as its ‘to-be’ state[67] (future state, nominal state). Yet while the concepts are usable for both the ‘as-is’ case and the ‘to-be’ case, the two cases are not explicitly distinguished within the current version of OntoCAPE. Thus, it has to be deduced from context, which of the two cases prevails.

5.3.1 Function and Requirements

The ontological representation of *function* in design is a long-standing research issue. Various definitions of the function concept have been proposed in the literature; for a review of those, see for example Baxter et al. (1994); Chandrasekaran (1994); Bilgic and Rock (1997); Chandrasekaran and Josephson (2000); Szykman et al. (2001); and Kitamura and Mizoguchi (2003).
Here, we adopt the definition of Chandrasekaran and Josephson (2000), who define function as *desired behavior*. Thus, function is an abstraction of the *actual behavior* (cf. Sect. 5.3.3) insofar as only the desired effects are considered, whereas all the unwanted and/or side-effects are ignored.
According to Chandrasekaran and Josephson (2000), two interpretations of the function concept must be distinguished for a *technical system*: function seen from an *environment-centric* viewpoint and function seen from a *device-centric* viewpoint (in this context, ‘device’ is used synonymously with *technical system*). The former viewpoint reflects the desired effect that a *technical system* exerts on its environment, yet without considering how this effect is to be achieved; the latter viewpoint additionally incorporates the principle of function of the *technical system*.
In OntoCAPE, the class *system function* represents the device-centric viewpoint, while the environment-centric viewpoint is described through the class *system requirement*; both are subclasses of *aspect system*.
The environment-centric viewpoint (*system requirements*) is more abstract than the device-centric viewpoint (*system function*): *System requirements* can be stated without knowledge of their technical realization; only the desired effect on the environment needs to be specified. The *system function*, on the other hand, specifies <u>how</u> the *technical system* fulfills the *system requirements*. Hence, the *conceptual design* of the *technical system* must be specified in terms of the underlying physico-chemical or technical principles.
As an example, consider the design of a process unit. The *system requirements* can be stated by describing the effect that the process unit shall exert on the processed materials (e.g., to separate dispersed particles from a liquid). Yet to specify the *system function*, one needs to consider the physical or technical principles based on

[67] Particularly, the concepts associated with the viewpoints of requirements and function are frequently (but not exclusively) employed to specify the ‘to-be’ state of a technical system, e.g. during its design phase.

which the desired effect is going to be achieved (e.g., decide whether the separation is realized by means of sedimentation, centrifugation, or filtration). Thus, "moving from an environment-centric functional description towards a device-centric description calls for partially solving the design problem" (Chandrasekaran and Josephson 2000).

Clearly, the main use for the concepts of *system requirements* and *system function* is to specify the 'to-be' state of a *technical system* during its design phase. Usually, the *system requirements* are formulated first, specifying the desired effect of the *technical system* on the environment. Later, at the conceptual design stage, the *system requirements* are refined into *system functions*, particularizing the principle based on which the desired effect is to be accomplished.

In addition to that, the concepts of *system requirements* and *system function* may also be used to characterize the 'as-is' state of a *technical system*. Note, however, that the semantics differ slightly, depending on whether the 'as-is' state or the 'to-be' state of the *technical system* is to be described:

- In the 'to-be' case, the *system requirements* and *system function* specify the planned desired behavior of the *technical system*, as, for example, envisioned in the early phases of the design process.
- In the 'as-is' case, the *system requirements* and *system function* provide an abstract (i.e., environment-centric or device-centric) description of the actual desired behavior.

In other words: the 'as-is' case describes the desired behavior that is effectively attainable under optimal conditions. Obviously, this may differ from the planned desired behavior reflected by the 'to-be' case. As an example, consider a chemical plant that has been designed for a nominal production capacity of 200,000 tons per year. After commissioning, however, it turns out that – due to some unforeseen problems – the actual production capacity is only 190,000 tons per year, at best. The nominal production capacity can be considered as the 'to-be' *system requirements*, whereas the actual production capacity can be considered as the 'as-is' *system requirements*.

5.3.2 Realization

The realization aspect, represented through the class *system realization*, reflects the physical (or virtual) constitution of the *technical system*. In case of a physical system, the *system realization* describes the system's physical structure, including its geometrical and mechanical properties. For example, the *system realization* of a chemical process would comprise the equipment and machinery required for materials processing; the *system realization* of a chemical product would reflect its molecular structure, crystal morphology, etc. In case of a non-physical system (such as a computer program), the *system realization* reflects the logical or abstract structure of the system; also, it may describe the (physical) implementation of the non-physical system (e.g., the model equations of a mathematical model or the source code of a computer program). Generally, the *system realization* gives a static description of the *technical system*, as opposed to the *system behavior* (cf. next section), which describes its dynamic behavior. Consequently, a *system realization* has mostly *constant properties*, which are often represented in shorthand notation via the hasCharacteristic relation (cf. Sect. 5.1.14).

A *system realization* may describe the 'as-is' state of the *technical system* as well as its 'to-be' state. In the 'as-is' case, it is comparable to a technical documentation, which reflects the current state of the *technical system*. By contrast, the 'to-be' case is comparable to a technical specification, as it is typically created in an engineering design project to specify the *technical system* that is to be built. In this context, it is important to remember that a *system realization* holds only information pertaining to the system itself; information that specify how to realize a *technical system* (e.g., assembly instructions or production planning) do not form part of the *system realization*.

Note that a *system realization* can be specified on different levels of detail and abstraction. For example, the *system realization* of a chemical plant may be stated on the information level of a P&ID (which represents the major equipment items and their main dimensions, but no geometrical details) as well as on the more detailed information level provided by isometric drawings and 3D models.

5.3.3 Behavior

The class *system behavior* describes how a *technical system* operates under certain conditions. Unlike the previously introduced *system requirements* and *system function*, which consider only the desired behavior, the *system behavior* also accounts for the unwanted behavior and the side-effects. As an example, consider chemical reactor, which is described from the viewpoint of *system behavior*: Such a description would comprise not only the main reaction (i.e. the desired behavior), but also include the undesirable side reactions.

If the *technical system* is described 'as-is', the *system behavior* reflects the behavior that can be actually observed. In the 'to-be' case, the *system behavior* concept represents the predicted behavior, which may be estimated on the basis of experiments or mathematical models.

The *system behavior* can be described both quantitatively and qualitatively. A quantitative description is provided by the *values* of its *properties*, which must be distinguished by means of a suitable backdrop *property*, usually a *temporal coordinate*[68] (cf. Sect. 6.4). This agrees well with the literature on dynamic systems (e.g., Föllinger 1982), where the behavior of a system is often defined as the change of

[68] Of course, other choices of backdrop *properties* are also possible.

its states over time. According to Bayer et al. (2001), the *values* of one distinct *property* and their related (temporal) backdrop *values* can be considered as a *state variables* of the *technical system*. The *state* of a *technical system* is given by the totality of all state variables at one particular point in time. Thus, a state can be considered as a temporal snapshot of the *system behavior*, and the *system behavior* can be described by the sequence of its states over time.

A qualitative description of the *system behavior* can be obtained by indicating the system's characteristic *phenomena*. In this context, a *phenomenon* denotes a typical mode of behavior exhibited by the system. The specification of a phenomenon implies (1) the existence of certain *properties* associated with that particular mode of behavior, and (2) that the *values* of these *properties* follow a designated pattern. To give an example: the indication of the *physicochemical phenomenon* of **laminar** flow (cf. Sect. 8.6.1.6) implies that (1) the *properties* '*velocity*' (or '*mass flow*'), '*viscosity*', and '*density*' are of relevance for describing the *system behavior*, and (2) that the *values* of these *properties* must comply with the laws of laminar flow[69]. Thus, through the specification of the prevailing *phenomena*, the state of the *technical system* can be qualitatively defined[70].

5.3.4 Performance

The *system performance* is concerned with the evaluation and benchmarking of the *technical system*. The concept itself represents a performance measure for the evaluation. Different performance measures are possible, depending on the chosen evaluation criterion. Typical criteria would be safety, reliability, ecological performance, and economic performance; a typical performance measure for the latter would be costs. The *system performance* can represent the predicted performance ('to-be' case) as well as the performance that is actually measured ('as-is' case).

Note that a *system performance* may evaluate only a particular aspect of the *technical system*: For example, construction costs measure the economic performance of a *system realization*, operating costs denote the economic performance of a *system behavior*, and a ranking of conceptual design alternatives corresponds to the performance evaluation of some *system function*.

[69] Note that the mathematical formulation of the laws of laminar flow can be specified through concepts from the partial model **mathematical_model** (cf. Chap. 9).

[70] Even for the specification of the quantitative behavior, it is advantageous to specify the *phenomena* first; afterwards, one may query the ontology for a list of relevant *properties* and physical *laws* associated with these *phenomena*.

5.3.5 Implementation of the Technical System in OntoCAPE

In OntoCAPE, the viewpoints of *system requirements*, *system function*, *system behavior*, *system realization* and *system performance* are modeled as subclasses of *aspect system*. Each *aspect system* is assigned an instance of the *aspect* class, which explicitly typifies the nature of the respective *aspect system*: For example, the *system function* is assigned the *aspect* of **function** (cf. Fig. 5.28).

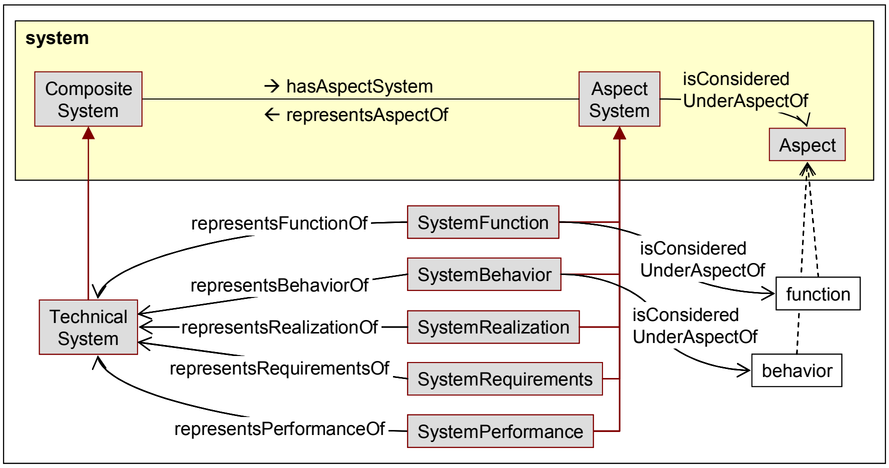

Fig. 5.28: The five major aspects of a technical system

The relationships between the *technical system* and its *aspect systems* are established via specializations of the relations hasAspectSystem and representsAspectOf, as indicated in Fig. 5.28 and Fig. 5.29.

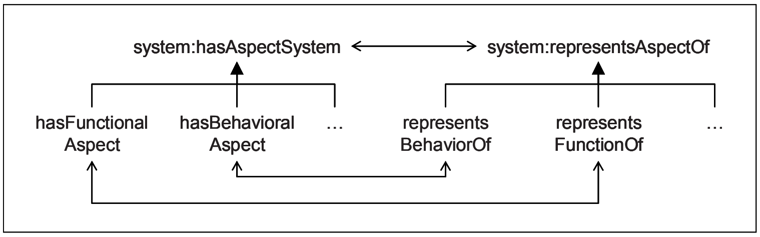

Fig. 5.29: Refinement of the hasAspectSystem relation

As explained above, the *system behavior* can be qualitatively described by indicating the relevant phenomena. This is modeled through the class *phenomenon*, which is assigned to a *system behavior* via the relation hasPhenomenon (Fig. 5.30).
The occurrence of a particular *phenomenon* exerts an influence on certain *properties*: For example, if the *phenomenon* of **laminar flow** is present, it will influence the

properties '*velocity*' and/or '*mass flow*'; the *phenomenon* of **chemical equilibrium** has an influence on the *concentrations*, etc. These kinds of interdependencies can be modeled by means of the relation isInfluencedBy, which explicitly designates those *properties* that are influenced by a particular *phenomenon*.

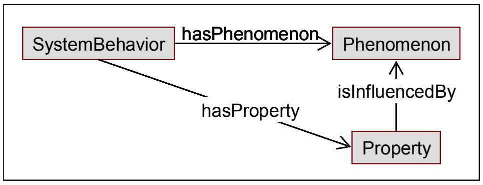

Fig. 5.30: Qualitative description of system behavior

5.3.6 Relations between Aspect Systems

Manifold relations and dependencies exist between the *aspect systems* of *technical system*. The type and the number of relations vary, depending on the respective application context. For example, the following relationships will arise in the course of a design project:

- In conceptual design, the *system requirements* are transformed into *system functions*.
- Later, the *system function* is detailed into the *system realization* at the stage of basic design.
- The *system realization* sets boundary conditions that constrain the possible *system behavior*.

Depending on the target application, an ontological model of these relations can turn very complex. For example, Kitamura and Mizoguchi (2003) present a fairly large ontology designated solely for modeling the interrelations between *system requirements* and *system functions*. According to the authors, this level of detail is required to provide adequate support for an intelligent design environment.

So far, such applications have not been the focus of OntoCAPE; consequently, the inter-aspect relations are presently not modeled in detail. Fig. 5.31 presents some generic binary relations, which may be used to navigate between aspect systems; additional ones may be introduced if required.

Generally, the inter-aspect relations displayed in Fig. 5.31 are specializations of the isRelatedTo relation.

- *System requirements* and *system function* can be linked via the relations fulfills and its inverse isAchievedThrough, thus stating that a conceptual design solution fulfills a particular requirement.

– The relation realizes and its inverse isRealizedBy indicate that a particular *system realization* is able to implement some *system function*.
– The relations constrains and isConstrainedBy denote the restrictions on the *system behavior*, which are imposed by a *system realization*.

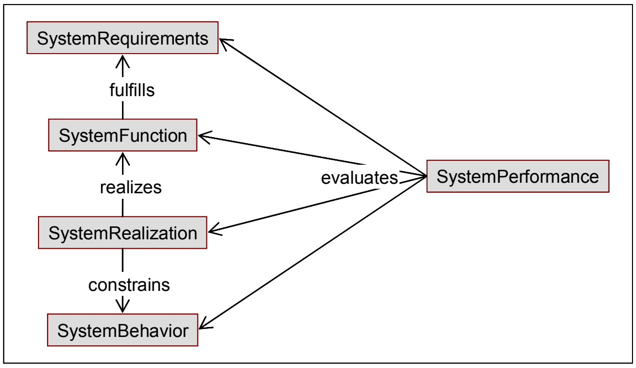

Fig. 5.31: Exemplary relation applied between *aspect systems*

Finally, the relation evaluates refers from a *system performance* to the *aspect system* the performance of which is measured; its inverse hasPerformanceMeasure points from the evaluated *aspect system* to the performance measure.

5.3.7 Concept Descriptions

Individual concepts of the module *technical_system* are defined below.

Class Descriptions

Phenomenon
A *phenomenon* denotes a typical mode of behavior exhibited by a *technical system*, thus providing a qualitative description of a recurring *system behavior*.

System behavior
The *system behavior* describes how a *technical system* operates under certain conditions; this description can be of a qualitative or quantitative nature.
Formal definition: A *system behavior* represents the behavioral aspect of a *technical system*.

System function
A *system function* describes the desired behavior of a *technical system* from a device-centric perspective (cf. Chandrasekaran and Josephson 2000). To indicate the *system function* of a *technical system*, the conceptual design of the *technical system* must be specified in terms of the underlying physicochemical and/or technical principles.
Formal definition: A *system function* represents the functional aspect of a *technical system.*

System Performance
The *system performance* concept constitutes a performance measure for the evaluation and benchmarking of *technical systems*. Different performance measures are possible, depending on the chosen evaluation criterion. Typical criteria would be safety, reliability, ecological performance, and economic performance.
Formal definition: A *system performance* represents the performance aspect of a *technical system.*

System realization
The *system realization* represents the physical (or virtual) constitution of the *technical system.* In case of a physical system, the *system realization* describes the system's physical structure, including its geometrical and mechanical properties. In case of a non-physical system, the *system realization* reflects the logical or abstract structure of the system; moreover, it may describe the (physical) implementation of the non-physical system.
Formal definition: A *system realization* represents the realization aspect of a *technical system.*

System requirements
The *system requirements* specify the desired behavior of a *technical system* from an environment-centric perspective (cf. Chandrasekaran and Josephson 2000). From the perspective of *systems requirements*, the *technical system* is viewed as a black box: Its structure and the underling physical and technical principles are not considered; only the effect on the environment is specified.
Formal definition: The *system requirements* represent the requirements aspect of a *technical system.*

Technical system
A *technical system* is a *system* which has been developed in an engineering design process. The criterion to qualify as a *technical system* is "to be designed in order to fulfill some required function" (Bayer 2003). Thus, the *technical system* concept may denote all kinds of technical artifacts, such as chemical plants, cars, computer systems, or infrastructure systems like a sewage water system. But also non-technical artifacts like chemical products, and even non-physical artifacts, such as software programs or mathematical models, can be considered as *technical systems.*

Relation Descriptions

constrains
The constrains relation indicates that a *system realization* imposes constraints on the *system behavior*.

evaluates
The relation evaluates refers from a performance measure to the *aspect system* the performance of which is evaluated.

fulfills
The fulfills relation states that a *system function* fulfills a particular *system requirement*.

hasBehavioralAspect
The relation points to the behavioral aspect of a *technical system*.

hasFunctionalAspect
The relation points to the functional aspect of a *technical system*.

hasPerformanceMeasure
The relation hasPerformanceMeasure points from an *aspect system*, the performance of which is evaluated, to the performance measure.

hasPerformanceAspect
The relation points to the performance aspect of a *technical system*.

hasPhenomenon
The relation hasPhenomenon assigns a *phenomenon* to a *system behavior*.

hasRealizationAspect
The relation points to the realization aspect of a *technical system*.

hasRequirementsAspect
The relation points to the requirements aspect of a *technical system*.

isInfluencedBy
The relation isInfluencedBy indicates which *properties* are influenced by a particular *phenomenon*.

isAchievedThrough
The relation isAchievedThrough states that a *system requirement* can be achieved by means of a some *system function*.

isConstrainedBy
The isConstrainedBy relation states that the *system behavior* is limited by the constraints imposed by the *system realization*.

isRealizedBy
The relation isRealizedBy states that a *system function* is implemented by some *system realization*.

realizes
The relation realizes states that a *system realization* implements a particular *system function*.

representsBehaviorOf
The relation refers from a *system behavior* to the overall *technical system*.

representsFunctionOf
The relation refers from a *system function* to the overall *technical system*.

representsPerformanceOf
The relation refers from a *system performance* to the overall *technical system*.

representsRealizationOf
The relation refers from a *system realization* to the overall *technical system*.

representsRequirementsOf
The relation refers from the *system requirements* to the overall *technical system*.

5.4 Coordinate System

The ontology module *coordinate_system* is a supplement to the *system* module. Fig. 5.32 gives an overview on the concepts established by *coordinate_system*. In particular, it introduces the concept of a *coordinate system*, a special type of *system* that provides a frame of reference for the observation of *properties* owned by other *systems*.

The *properties* of a *coordinate system* are called *coordinates*. A *coordinate* is defined as a *scalar quantity*, the values of which (i) serve as a backdrop for some *values* and (ii) cannot be observed against some further backdrop. Hence, as a *coordinate* cannot have a backdrop of its own, it constitutes an ‘absolute’ or ‘final’ backdrop for the observation of *properties*; it thus breaks the loop caused by the relativity of the backdrop concept (cf. the discussion in Sect. 5.1.10).

Each *coordinate* refers to one *coordinate system axis*, which further qualifies the *coordinate*. For example, a spatial coordinate may refer to the x-axis of a spatial coordinate system, thus clarifying its spatial orientation. The *coordinate system axis* itself is not further specified through ontological concepts; consequently, its characteristics – e.g., its orientation relative to some spatial objects not described by OntoCAPE – must be defined outside the ontology.

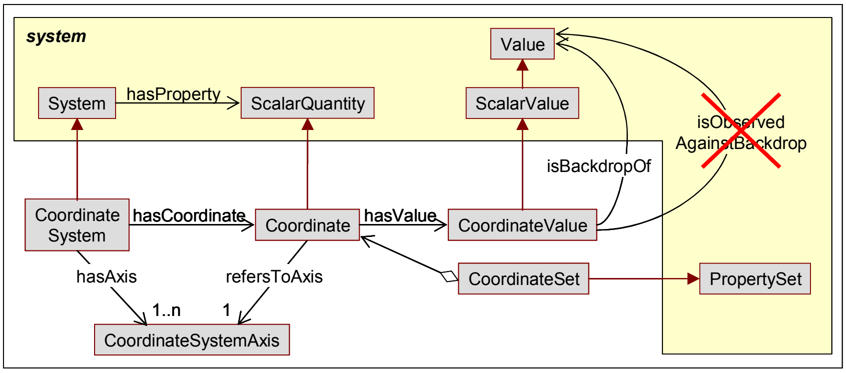

Fig. 5.32: Coordinate system

Detailed concept definitions are given below. The usage of the concepts is explained in Sect. 6.4 as part of the documentation of ontology module *space_and_time*

5.4.1 Concept Descriptions

Individual concepts of the module *coordinate_system* are defined below.

Class Descriptions

Coordinate
A *coordinate* is a property of a *coordinate system*. The values of a *coordinate* provide an ‘absolute’ or ‘final’ backdrop for the observation of some *properties*.

Coordinate set
A *coordinate set* groups some *coordinates* which logically belong together.
Formal definition: A *coordinate set* is a *property set* that comprises only *coordinates*.

Coordinate system
A *coordinate system* constitutes a frame of reference for the observation of *properties* owned by other *systems*. A *coordinate system* is a *system* that has some *coordinates* as properties.

Coordinate system axis
A *coordinate system axis* represents an axis of a *coordinate system*.

Coordinate value
A *coordinate value* serves as a backdrop for some *values*, yet it cannot have a backdrop of its own.
Formal definition: A *coordinate value* is a *scalar value* which is the value of a *coordinate*.

Relation Descriptions

hasAxis
The relation hasAxis identifies the *coordinate system axes* that belong to a particular *coordinate system*.

hasCoordinate
The relation hasCoordinate indicates the *coordinates* of a *coordinate system*.

refersToAxis
By means of the relation refersToAxis, a *coordinate* can be further specified. For example, a spatial coordinate may refer to the x-axis of a spatial coordinate system, thus clarifying its spatial orientation.

5.5 Tensor Quantity

As explained in Sect. 5.1.11, *physical quantities* include not only scalars but also vectors (e.g., velocity vector) and higher-order tensors (e.g., the dyadic stress tensor). The ontology module *tensor_quantity* provides the necessary concepts to define such *tensor quantities*.
A *tensor quantity* is a *physical quantity* that is assigned a tensor order. A *tensor quantity* of order *k* can be defined by induction:

- A *tensor quantity* of order 0 is a *scalar quantity*.
- A *tensor quantity* of rank k is given by an n-tuple, the elements of which are again *tensor quantities* of order (k-1).

Thus, a *tensor quantity* of arbitrary order can be recursively decomposed into *tensor quantities* of lower order, ultimately obtaining *scalar quantities*.
The above definition is implemented in OWL as follows. The order of the *tensor quantity* is denoted by the attribute hasTensorOrder. For the modeling of the tuple structure, we apply the design pattern for an array introduced in the Meta Model (cf. Sect. 4.5.3). This leads to the structure displayed on the left-hand side of Fig. 5.33.
A *tensor quantity* has elements of type *physical quantity*, which may again be *tensor quantities* of a lower order (note that the rank reduction of the tensor elements cannot be enforced in the OWL language, but must be accomplished manually). The

order of the tensor elements is established through the *index* class: Each tensor element is assigned an *index* with unique integer value (given by the index attribute) via the determinesPositionOf relation; the *indices* refer to the *tensor quantity* via the isOrderedBy relation (cf. Sect. 5.5 for details).

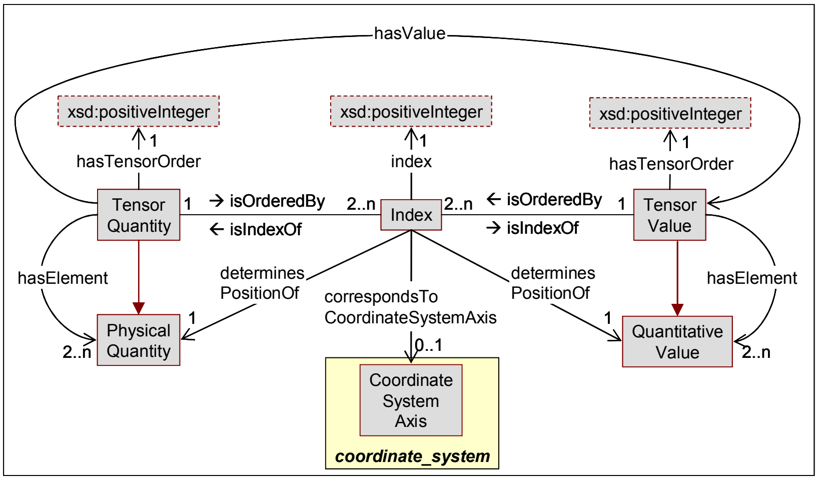

Fig. 5.33: Tensor quantity and tensor value

The *value* of a *tensor quantity* must again be a tensor of the same order as the *tensor quantity*. To this end, the class *tensor value* is introduced. A *tensor value* is defined analogously to a *tensor quantity*, as can be seen on the right-hand side of Fig. 5.33. Thus, each *tensor value* can be ultimately decomposed into *scalar values*. Like all *physical quantities*, a *tensor quantity* is assigned a *physical dimension*, which must be the same *physical dimension* as that of its tensor elements[71]. Thus, unlike the concept of a *property set*, a *tensor quantity* comprises only *physical quantities* of the same type.

Two special types of *tensor quantities* are exemplarily introduced below: the *vector quantity* and the *matrix quantity*.

A *vector quantity* is a *tensor quantity* that has a tensor order of 1. It is composed of *vector elements*, subclasses of *scalar quantity*, which by default refer to an *index* via the hasIndex relation. A *vector quantity* has *vector values*, which are defined analogously to *vector quantities*. A *vector value* is composed of scalar *vector element values*; these are specialized *scalar values* referring to an *index*. Fig. 5.34 summarizes the above concept definitions.

A *matrix quantity* is a *tensor quantity* of rank 2, the elements of which are *vector quantities*. As these vectors constitute the columns of the *matrix quantity*, they are specif-

[71] Note that this axiom cannot be expressed in the OWL language; consequently, it must be enforced by the user.

ically designated as *column vector quantities*, and each *column vector quantity* is assigned a *column index*. By contrast, the *vector elements* of the *column vector quantity* are ordered by a *row index*.

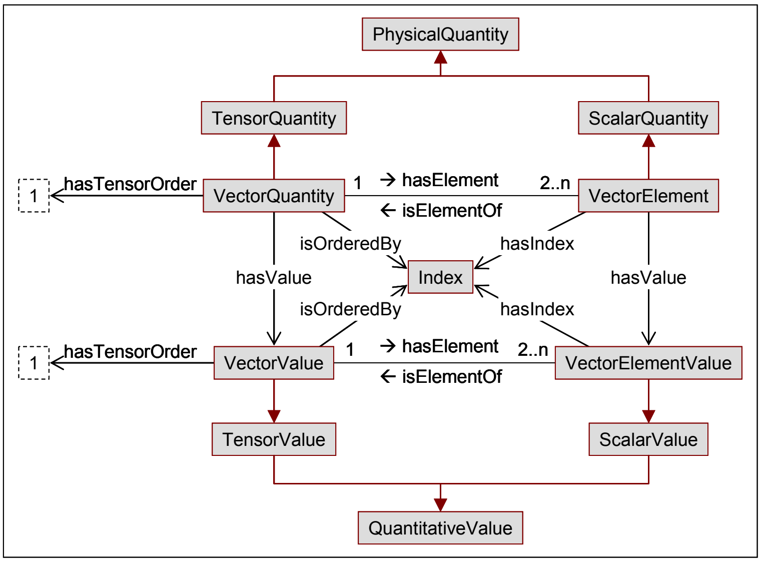

Fig. 5.34: Interrelations between *vector quantity*, *vector element*, *vector value*, and *vector element value*

The definitions of these concepts are summarized by Fig. 5.35; Fig. 5.36 illustrates their usage.

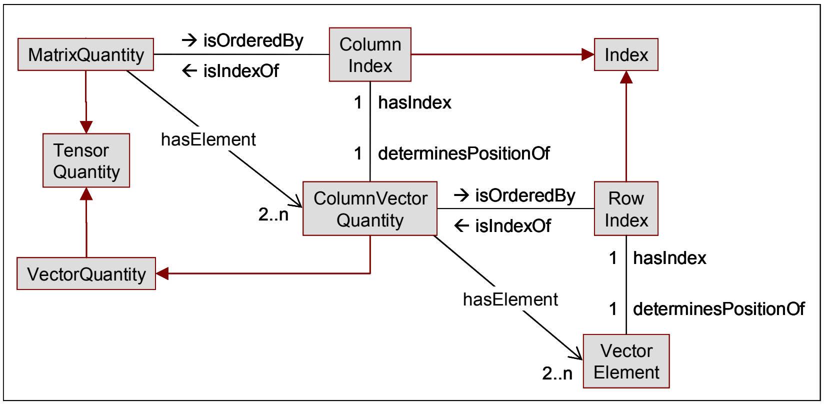

Fig. 5.35: Definition of the *matrix quantity* concept

The value of a *matrix quantity* is designated as a *matrix value* (not shown in Fig. 5.35 for the sake of clarity). Analogously to the above definitions, a *matrix value* is composed of *column vector values*, again ordered by a *column index*; the elements of the *column vector value* are *vector values*, which are ordered by a *row index*.

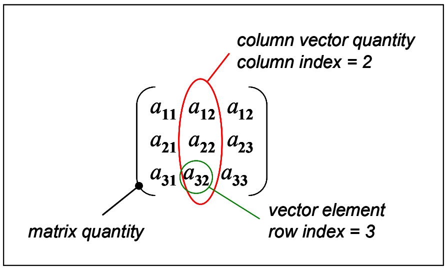

Fig. 5.36: Usage of the *matrix quantity* concept

Concluding the above discussion, Fig. 5.37 gives an application example. It shows a two-dimensional stress tensor (i.e., *matrix quantity*), consisting of the *scalar quantities* σ_x, τ_{xy}, τ_{yx}, and σ_y, and its associated *matrix value*. Note that only the second columns of *matrix quantity* and *matrix value* are elaborately modeled. For the sake of clarity, the respective class names in brackets are omitted.

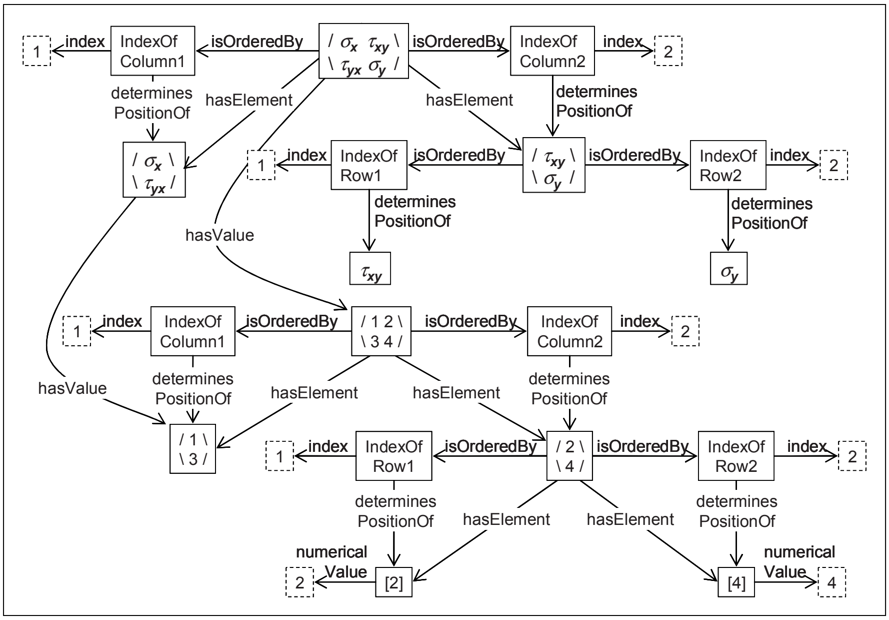

Fig. 5.37: Application example of a *matrix quantity* and its *matrix value*

The definitions introduced so far conceptualize a tensor as a mere data structure, thereby ignoring its geometrical properties. Yet the complete specification of a tensor requires a statement of direction or orientation (Gruber and Olsen 1994). The tensor orientation can be indicated by assigning a spatial dimension to each element of a tensor; concretely, this is realized by referring from a *vector element* to the concept of a *coordinate system axis* (cf. Sect. 6.4) via the relation hasOrientation. Note that a *vector element* may refer to a *cartesian coordinate system axis* or a *curvilinear coordinate system axis* (cf. definitions in Sect. 6.4). The latter enables the definition of rotation vectors to represent *physical quantities* like torque or angular momentum.

The reference to a *coordinate system axis* (cf. Fig. 5.38) is of special importance, since we have defined the tensor as the recursive composition of its scalar elements. Yet while a tensor (as a whole) is independent of any chosen frame of reference, the decomposition of the tensor into its scalar elements depends on the particular choice of the reference frame. Thus, for a complete definition of a tensor in terms of its constituent elements, the respective reference *coordinate system* must be specified. If such specification is omitted, the following will be assumed by default: The tensor elements refer to a positive Cartesian coordinate system, where the *vector element* with an index value of 1 refers to the *x*-axis, and the *vector element* with an index value of 2 refers to the *y*-axis, etc.

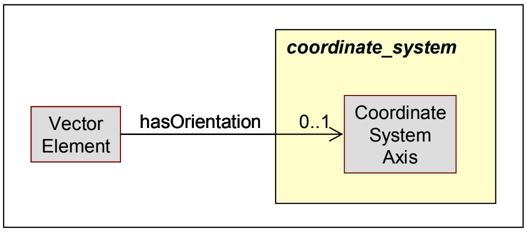

Fig. 5.38: Specifying the orientation of a tensor by referring to a *coordinate system axis*

5.5.1 Concept Descriptions

Individual concepts of the module *tensor_quantity* are defined below.

Class Descriptions

Column index
A *column index* denotes the position of a column vector within a matrix.

Column vector quantity
A *column vector quantity* represents a column vector of a *matrix quantity*.
Formal definition: A *column vector quantity* is a *vector quantity* that is an element of a *matrix quantity*.

Column vector value
A *column vector value* represents a column vector of a *matrix value*.
Formal definition: A *column vector value* is a *vector value* that is an element of a *matrix value*.

Index
An *index* represents the n-ary relation between a tensor, one of its elements, and the index attribute that denotes the position of the tensor element.

Matrix quantity
A *matrix quantity* is a second order *tensor quantity*.

Matrix value
A *matrix value* is a second order *tensor value*.

Row index
A *row index* denotes the position of a scalar element within a column vector.

Tensor quantity
A *tensor quantity* is a non-scalar *physical quantity*, such as a velocity vector or a stress tensor.

Tensor value
A *tensor value* is non-scalar *quantitative value* of a *tensor quantity*.

Vector element
Formal definition: A *vector element* is a *scalar quantity* that is the element of a *vector quantity*.

Vector element value
Formal definition: A *vector element value* is a *scalar value* that is the element of a *vector value*.

Vector quantity
A *vector quantity* is a first order *tensor quantity*.

Vector value
A *vector value* is a first order *tensor value*.

Relation Descriptions

determinesPositionOf
The relation determinesPositionOf refers from an *index* to the associated tensor element.

hasElement
The relation hasElement identifies the elements of a tensor.

hasIndex
The relation hasIndex refers from a tensor element to its *index*.

isElementOf
The relation isElementOf denotes the affiliation of a tensor element to a tensor.

isIndexOf
The relation isIndexOf points from an *index* to the associated tensor.

isOrderedBy
The relation isOrderedBy identifies the *index* of a tensor.

hasOrientation
The relation hasOrientation specifies the orientation of a tensor element by referring to the corresponding *coordinate system axis*.

Attribute Descriptions

hasTensorOrder
The attribute denotes the order of a tensor. Scalars are of order 0, vectors of order 1.

index
The attribute indicates the numerical value of an *index*.

5.6 References

Alberts LK (1994) YMIR: a sharable ontology for the formal representation of engineering design knowledge. In: Gero JS, Tyugu E (eds.): *Formal Design Methods for CAD*. Elsevier, New York:3–32.

Andresen T (1999) The macroeconomy as a network of money-flow transfer functions. *Model. Ident. Contr.* **19** (4):207-223.

Barkmeyer EJ, Feeney AB, Denno P, Flater DW, Libes DE, Steves MP, Wallace EK (2003) *Concepts for Automating Systems Integration*. Technical Report (NISTIR 6928), National Institute of Standards and Technology (NIST), Gaithersburg, MD.

Baxter JE, Juster NP, de Pennington A (1994) A functional data model for assemblies used to verify product design specifications. *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* **208** (B4):235-244.

Bayer B (2003) *Conceptual Information Modeling for Computer Aided Support of Chemical Process Design*. Fortschritt-Berichte VDI: Reihe 3, Nr. 787. VDI-Verlag, Düsseldorf.

Bayer B, Krobb C, Marquardt W (2001) *A Data Model for Design Data in Chemical Engineering - Information Models*, Technical Report LPT-2001-15, Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bertalanffy L (1968). General System Theory: Foundations, Development, Applications, Braziller, New York.

Bilgic T, Rock D (1997) Product data management systems: State-of-the-art and the future. In: *Proceedings of the 1997 ASME Design Engineering Technical Conferences,* Sacramento, CA.

BIPM (2006) *The International System of Units (SI)*, 8th edition. SI brochure, published by the International Committee for Weights and Measures (Bureau International des Poids et Measures, BIPM). Online available at http://www.bipm.fr/en/si/si_brochure/. Accessed 26 September 2007.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, PhD Thesis, Centre of Telematics and Information Technology, University of Twente.

Bunge M (1979) *Treatise on Basic Philosophy, Volume 4. Ontology II: A World of Systems.* Reidel, Dordrecht.

Chandrasekaran B (1994) Functional representation and causal processes. In: Yovits MC (ed.): *Advances in Computers*. Academic Press, New York.

Chandrasekaran B, Josephson JR (2000) Function in device representation. *J. Eng. Comput.* **16** (3/4):162-177.

Föllinger O (1982) Einführung in die Zustandsbeschreibung dynamischer Systeme. Oldenbourg, München.

Gigch JP (1991) System Design Modeling and Metamodeling. Springer, New York.

Gilles ED (1998) Network theory for chemical processes. *Chem. Eng. Technol.* **21** (8):121–132.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

IEEE (2000) *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*. IEEE Standard 1471-2000, Institute for Electrical and Electronics Engineering, New York.

Kitamura Y, Mizoguchi R (2003) Ontology-based description of functional design knowledge and its use in a functional way server. *Expert Syst. Appl.* **24** (2):153-166.

Klir GJ (1985) *Architecture of Systems Problem Solving*. Plenum Press, New York.

Mangold M, Angeles-Palacios O, Ginkel M, Kremling A, Waschler R, Kienle A, Gilles ED (2005) Computer-aided modeling of chemical and biological systems: methods, tools and applications. *Ind. Eng. Chem. Res.* **44**:2579–2591.

Marquardt W (1992a). An object-oriented representation of structured process models. *Comput. Chem. Eng.* **16**:329–336.

Marquardt W (1994b) Computer-aided generation of chemical engineering process models. *Int. Chem. Eng.* **34**:28–46.

Marquardt W (1995) Towards a Process Modeling Methodology. In: R. Berber: *Methods of Model-Based Control*. NATO-ASI E, Applied Sciences, **293**, Kluwer, Dordrecht:3-41.

Marquardt W (1996) Trends in computer-aided process modeling. *Comput. Chem. Eng.* **20** (6/7):591–609.

Marquardt W, von Wedel L, Bayer B (2000) Perspectives on lifecycle process modeling. In: Malone MF, Trainham JA, Carnahan B (eds.): *Foundations of Computer–Aided Process Design*, AIChE:192–214.

Morbach J, Yang A, Marquardt W (2007) OntoCAPE – A large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20** (2):147-161.

Patzak G (1982) *Systemtechnik – Planung komplexer innovativer Systeme.* Springer, Berlin.

Rector A, Welty C, eds. (2005) *Simple part-whole relations in OWL Ontologies*. W3C Editor's Draft, 11 August 2005. Online available at http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. Accessed November 2007.

Szykman S, Sriram RD, Regli WC (2001) The role of knowledge in next-generation product development systems. *J. Comput. Inf. Sci. Eng.* **1** (1):3–11.

Thomé B, ed. (1993) *Systems Engineering: Principles and Practice of Computer - based Systems Engineering*. John Wiley, New York.

VIM (1993) *International Vocabulary of Basic and General Terms in Metrology (VIM), 2nd Edition*. Jointly prepared by ISO, IEC, BIPM, IFCC, IUPAC, IUPAP and OIML. Published by ISO, Geneva, as ISO Guide **99**:1993.

6 Supporting Concepts

The partial model **supporting_concepts** provides fundamental notions, such as spatial and temporal coordinate systems, geometrical concepts, mathematical relations, as well as commonly used physical dimensions and SI-units. The concepts defined in this partial model do not belong to the core of the CAPE domain, but are merely utilized by the other partial models of OntoCAPE for defining and completing domain concepts. For that reason, **supporting_concepts** is only rudimentarily developed, as it is not the objective of OntoCAPE to conceptualize areas that are beyond the scope of the CAPE domain. For example, the partial model **mathematical_relation** does not attempt to establish a full-fledged algebraic theory, as does the EngMath ontology (Gruber and Olsen 1994); rather, it provides a simple but pragmatic mechanism for the representation of mathematical relations, which serves the current needs of the other partial models of OntoCAPE. In the future, the ontology modules of **supporting_concepts** could be replaced by generic ontologies developed and tested by others in a more systematic manner.

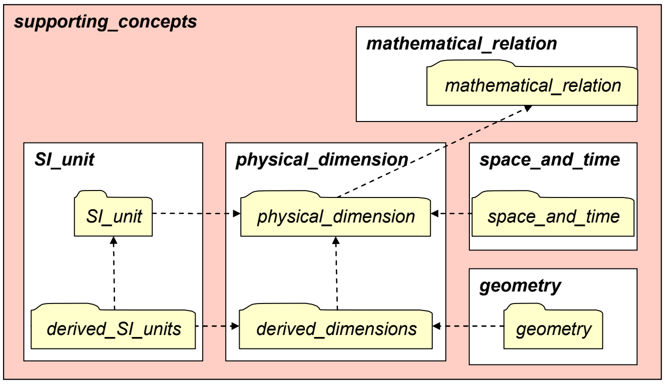

Fig. 6.1: Overview on partial model **supporting_concepts**

As depicted in Fig. 6.1, **supporting_concepts** comprises five subordinate partial models, which are **mathematical_relation**, **physical_dimension**, **SI_unit**, **space_and_time**, and **geometry**.

6.1 Mathematical Relation

The ontology module *mathematical_relation* introduces concepts to represent mathematical expressions. However, it is not the objective of this module to describe mathematical models. Rather, this module provides auxiliary concepts, which are utilized by other ontology modules (e.g., for the definition of units, cf. Sect. 6.3.1).

Mathematical relations and expressions are represented by means of binary trees[72]. The nodes of the tree represent the operands and the operators, and the structure of the tree specifies the order of evaluation. For example, the relation $a + b = 2$ can be represented through the binary tree depicted in Fig. 6.2.

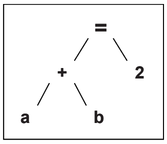

Fig. 6.2: Tree representation of the equation $a + b = 2$

The leaves of the tree represent the *operands* (here: a, b, 2), the internal nodes represents the *functional operators* (here: +), and the root node denotes either a *relational operator* (here: =) or another functional operator.
The ontological representation of binary trees is adopted from the design pattern *binary tree*, as defined in the Meta Model (cf. Sect. 4.5.1): A *node* may have a left child and a right child; special *nodes* are *root node*, *leaf*, and *internal node.*

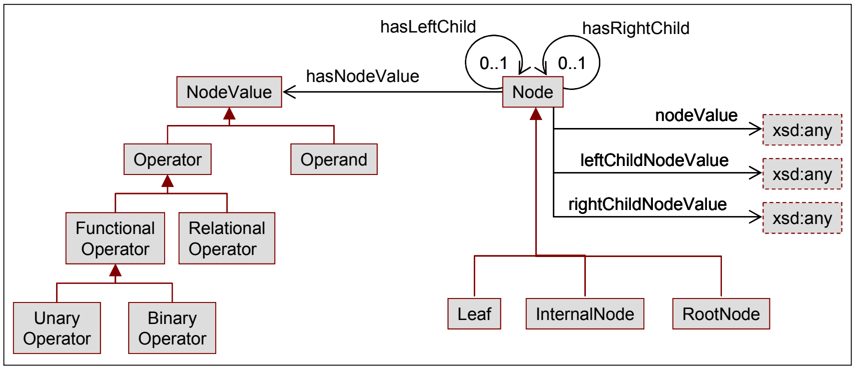

Fig. 6.3: Class diagram of the ontology module *mathematical relation*

Additionally, the class *node value* is introduced, which assigns a *node* to a mathematical concept (cf. Fig. 6.3). A node value is either an *operand* or an *operator.* The *operand* class represents the variables, parameters, or numbers of a mathematical expression. An *operator* is either a *relational operator* or a *functional operator.* A *relational operator* represents mathematical relations, such as equality or greater than. A *functional operator* denotes mathematical operations, such as addition, exponentiation, or logarithm. Two types of *functional operators* are distinguished: A *binary*

[72] Currently, it is applied to algebraic expressions only. However, the approach can probabely be extended to complex PDE systems.

operator, which takes two arguments, and a *unary operator*, taking only a single argument[73].
Depending on the type of *node*, some restrictions are imposed on the type of its *node value*:

- A *root node* has only *operators* as *node values*.
- An *internal node* has only *functional operators* as node values.
- A *leaf* has only *operands* as node values.
- If a *unary operator* is used, the corresponding *node* has exactly one child.

If a *binary operator* is used, the corresponding *node* has exactly two children.
It is not compulsory that the operands of a mathematical expression are represented as instances of the class *operand*; instead, they may be represented as values of the attribute nodeValue. An example is presented in Fig. 6.4, which shows the OntoCAPE representation of the equation $a + b = 2$. The variables a and b are values of type 'string', the number 2 is of type 'float' (both 'string' and 'float' are built-in XML schema datatypes; cf. Biron et al. 2004).

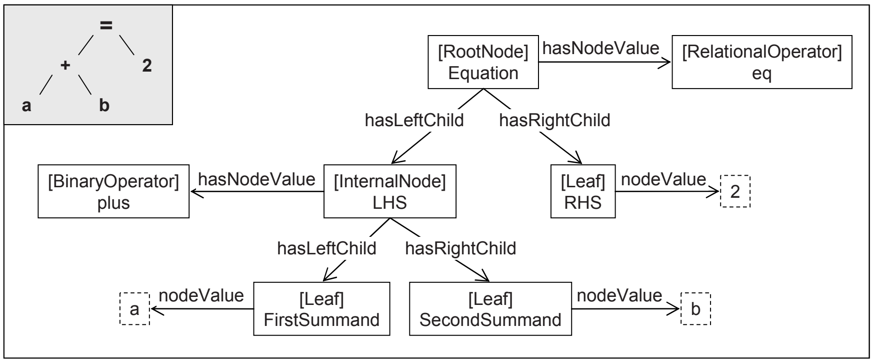

Fig. 6.4: OntoCAPE representation of $a + b = 2$; operands are represented through attribute values

Moreover, a shorthand notation may be used to substitute child *nodes* that apply the nodeValue attribute. These *nodes* must be *leaves* that have no other function but to carry the value of the attribute nodeValue. In this case, they may be replaced by the attributes leftChildNodeValue and rightChildNodeValue, which are applied instead of the relations hasLeftChild and hasRightChild. Fig. 6.5 shows again the representation of the equation $a + b = 2$, this time in shorthand notation. Compared to Fig. 6.5, the child *nodes* **RHS**, **FirstSummand**, and **SecondSummand** have been pruned, and their respective node values are represented through the attributes leftChildNodeValue and rightChildNodeValue.

[73] In this case, the corresponding node has only a single child node, which represents the operand.

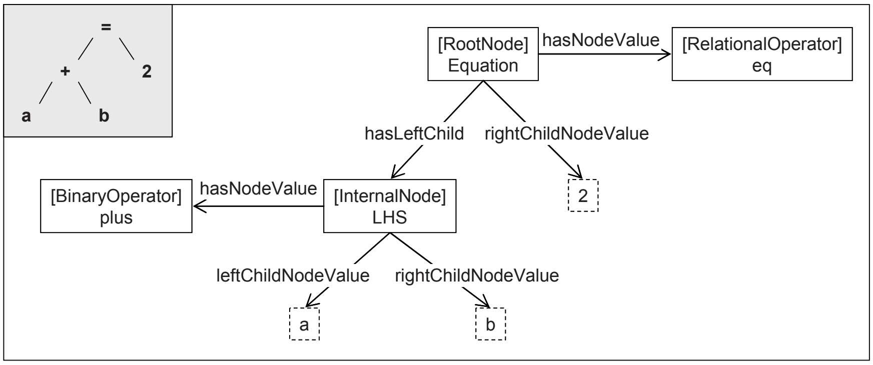

Fig. 6.5: Shorthand representation of the equation *a + b = 2*

6.1.1 Usage

In the module *physical_dimension*, the concepts of *mathematical_relation* are utilized to establish mathematical relations between physical dimensions: (e.g., *velocity* = *length / time*). Similarly, in the ontology module *SI_unit*, mathematical relations can be established between different units (e.g., *Newton* = *kilogram* * *meter / second*^2). In these applications, a single *operand* is frequently involved in multiple mathematical expressions. Take for instance the linear equation system

$$a + b = 2 \qquad (1)$$
$$a - b = 0 \qquad (2)$$

Here, the *operand* **a** appears both in equation (1) and in equation (2); the same is true for *operand* **b**. There are two possible representation patterns for such a situation:

Representation pattern 1 is shown in Fig. 6.6. Here, separate *nodes* (*leaves*) are created for each occurrence of an *operand*; for instance, the *leaf* **Var1_Eq1** represents the occurrence of the *operand* **a** in equation (1), **Var1_Eq2** represents the occurrence of **a** in equation (2).

Representation pattern 1 is rather straight-forward. However, it may lead to a significant overhead, as a new *node* must be created for each occurrence of an *operand*. This can be avoided by representation pattern 2, which is exemplarily presented in Fig. 6.7. Here, only one *node* (*leaf*) is created for each *operand*; this *node* forms part of multiple trees. For instance, **Var1** is a *leaf* of both the tree that represents equation (1) and of the tree that represents equation (2).

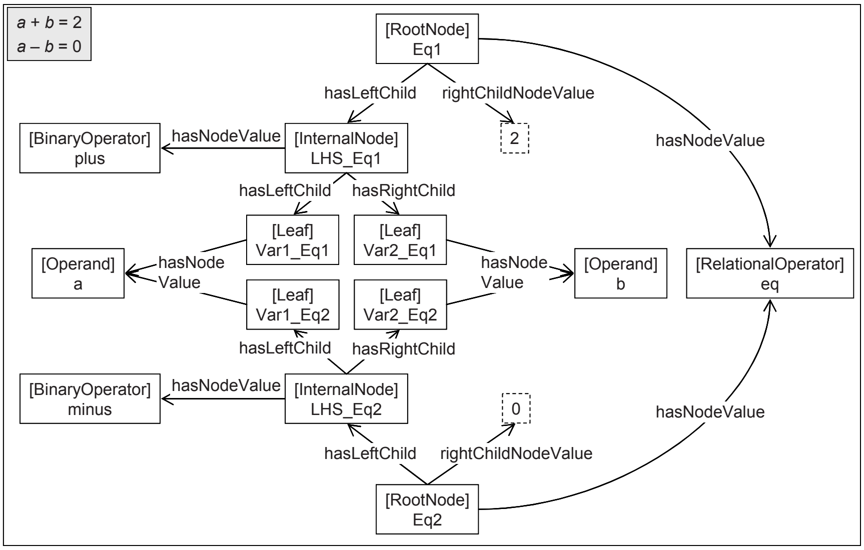

Fig. 6.6: Representation pattern 1

Pattern 2 reduces the representation effort, which is a considerable relief when modeling large equation systems or nested definition expressions, such as those in the ontology module *SI_unit*. On the other hand, pattern 2 is not as straightforward as representation pattern 1. The user must choose which one he/she prefers.

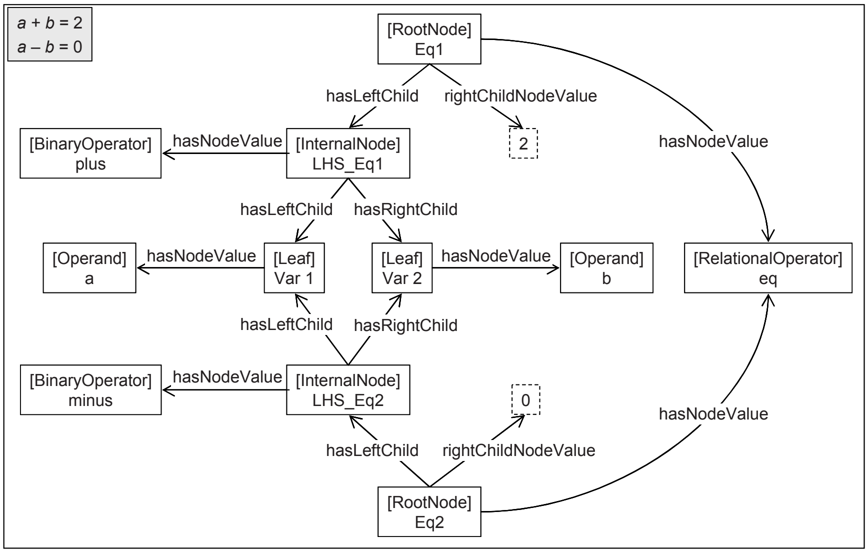

Fig. 6.7: Representation pattern 2

Note that representation pattern 1 can be enforced by adding the constraint that each *node* must have only a single parent.

6.1.2 Concept Descriptions

Individual concepts of the module *mathematica_ relations* are defined below. The labels and definitions of operator instances, e.g. plus, minus etc., have been adopted from the Mathematical Markup Language, MathML (Carlisle et al. 2003). For their description in OntoCAPE, we refer to Morbach et al. (2008h).

Class Descriptions

Binary operator
A *binary operator* denotes a binary operation between two expressions. Typical binary operations are addition, subtraction, multiplication, division, and exponentiation.

Functional operator
A *functional operator* denotes a mathematical function.
Formal definition: A *functional operator* is either a *unary operator* or a *binary operator*.

Internal node
An *internal node* is a *node* that has at least one parent and at least one child. The child may be represented either as a *node* or through the attribute leftChildNodeValue or rightChildNodeValue.
Formal definition: A *node* that has a parent *node* as well as a child *node*.

Leaf
A *leaf* is a *node* without any children.
Formal definition: A *node* that has neither a child *node*, nor a leftChildNodeValue, nor a rightChildNodeValue.

Node
A *node* is the basic element of a binary tree. It can be linked to up to two child *nodes*.
Formal definition: A *node* is either a *leaf* or a *root node* or an *internal node*.

Node value
A *node value* represents a component part of mathematical expression. It can be either an *operator* or an *operand*.
Formal definition: A *node value* is either an *operand* or an *operator*.

Operand
An *operand* is one of the inputs of a *functional operator*.

Operator
Formal definition: An *operator* is either a *relational operator* or a *functional operator*.

Relational operator
A *relational operator* denotes a mathematical relation, such as equality or greater than, between two expressions.

Root node
A *root node* is the root element of a binary tree. All other *nodes* are descendents of the *root node*.
Formal definition: A *node* without any parent *node*.

Unary operator
A *unary operator* denotes a mathematical operation which takes a single argument. Typical binary operations are squaring, root extraction, or factorial.

Relation Descriptions

hasAncestor
The ancestors of a *node* are the *nodes* that precede the current *node* in the tree (i.e., the *node*'s parent, grandparent, etc.).

hasChild
The relation hasChild points to the children of a *node*; it subsumes the relations hasLeftChild and hasRightChild.

hasDescendent
The descendents of a *node* are the *nodes* that succeed the current *node* in the tree (i.e., the *node*'s children, grandchildren, etc.).

hasLeftChild
The relation hasLeftChild links a parent *node* to its left child *node*.

hasNodeValue
The relation hasNodeValue links a *node value* to a *node*.

hasParent
The relation hasParent denotes the parent of a *node*.

hasRightChild
The relation hasRightChild links a parent *node* to its right child *node*.

isLeftChildOf
The relation isLeftChildOf points from the left child *node* to its parent *node*.

isRightChildOf
The relation isRightChildOf points from the right child *node* to its parent *node*.

Attribute Descriptions

leftChildNodeValue
The attribute leftChildNodeValue can be used as a shorthand to substitute a left child *node*, the node value of which is represented through the attribute nodeValue.

rightChildNodeValue
The attribute leftChildNodeValue can be used as a shorthand to substitute a left child *node*, the node value of which is represented through the attribute nodeValue.

nodeValue
The attribute nodeValue indicates an operand (usually a number) in a mathematical expression.

6.2 Physical Dimension (Partial Model)

The partial model **physical_dimension** comprises two ontology modules (cf. Fig. 6.8). The main module, *physical_dimension*, defines a set of base dimensions and establishes the proceedings to derive further physical dimensions from these base dimensions. It is extended by the ontology module *derived_dimensions*, which introduces a number of frequently used derived dimensions.

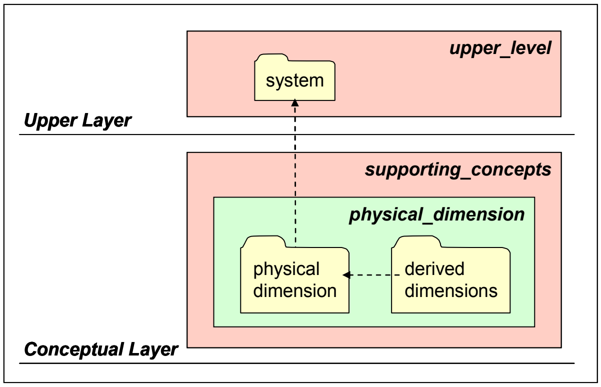

Fig. 6.8: Overview on partial model **physical_dimension**

6.2.1 Physical Dimension (Ontology Module)

The concept of a *physical dimension* has been introduced in the *system* module. Basically, it serves as a characteristic that can be associated with, *physical quantities*, and *units of measure* for the purpose of classification or differentiation.
The relations between *physical dimensions* can be described through mathematical equations, such as

Joule = Newton * meter.

Multiplication and exponentiation to a real power are the permissible operations on physical dimensions. Exploiting this property, the majority of *physical dimensions* can be mathematically derived from a small set of dimensions that we call *fundamental* or *base dimensions* by means of multiplication and exponentiation operations; *physical dimensions* that can be defined in terms of the base dimensions are called *composite* or *derived dimensions*. For example, the derived dimension 'velocity' can be defined as the product of the base dimension 'length' and the base dimension 'time' raised to the power of minus one.
There is no intrinsic property of a dimension that makes it fundamental (Gruber and Olsen 1994); hence, the choice of the base dimensions is a matter of convention. While the definition of the *physical dimension* concept in the *system* module still allows for arbitrary conventions (or 'systems of dimensions', as they will be called hereafter), the *physical_dimension* module establishes the SI system of dimensions, which comprises the base dimensions of *length*, *time*, *thermodynamic temperature*, *mass*, a*mount of substance*, *electric current*, and *luminous intensity* (BIPM 2006).
Fig. 6.9 presents the major concepts of the *physical_dimension* module. The class *physical dimension*, introduced in the ontology module *system*, is exhaustively partitioned into three subclasses:

– The class *base dimension* is defined as the exhaustive enumeration of the individuals **length**, **time**, **thermodynamic_temperature**, **mass**, **amount_of_substance**, **electric_current_strength**, and **luminous_ intensity**.
– The class *supplementary dimensions* subsumes further fundamental dimensions that do not form part of the SI system of units and are therefore not classified as *base dimensions*. Currently, the class holds two individuals: **amount_of_money** characterizes monetary *physical quantities* and *units of measure*. For the characterization of dimensionless *physical quantities*, the concept of an **identity_dimension** is introduced. Prominent examples of dimensionless *physical quantities* are dimensionless numbers like the Reynolds number, or counting quantities like the partition function in statistical thermodynamics or the number of trays in a distillation column. According to Gruber and Olsen (1994), the **identity_dimension** represents the identity element for multiplication over physical dimensions. That means that the product of the **identity_dimension** and any other *physical dimension* is that other *physical dimension*.

– All other *physical dimensions* are *derived dimensions*. These can be subdivided by further subclasses that, for example, group physical dimensions by fields of science. A possible classification is provided by the ontology module *derived_dimensions*: (cf. Sect. 6.2.2).

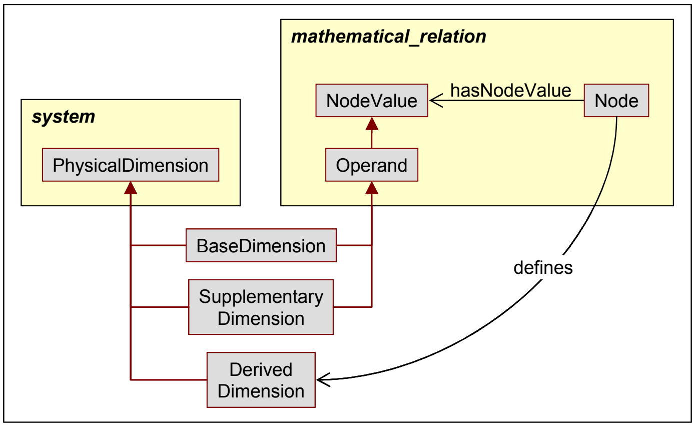

Fig. 6.9: Class diagram of the ontology module *physical_dimension*

The mathematical definition of a *derived dimension*[74] can be specified using the concepts defined in ontology module *mathematical_relation*: A *derived dimension* is defined by a *node*, which represents the root of a definition tree; the leaves of the definition tree have either *base dimensions* or *supplementary dimensions* as *node values*. To this end, *base dimension* and *supplementary dimension* are declared as subclasses of *operand*; that way, their instances may appear as operands in the definition equation. An example is given in Fig. 6.10.

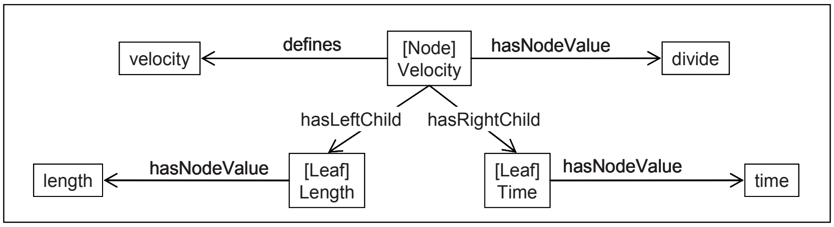

Fig. 6.10: Definition of the *derived dimension* '**velocity**'

[74] Implementation advice: Unfortunately, the definition of derived dimensions through the above mechanism scales badly with current reasoners. Therefore, such definitions should be omitted in order to enhance the reasoning performance, unless they are definitely required by the respective application.

Velocity is declared as an instance of *derived dimension*; it is defined by **Velocity** *node*, which is in turn defined by its two child *nodes*, **Length** and **Time**, which are concatenated by the *binary operator* **divide**. **Length** and **Time** have the *base dimensions* **length** and **time**, respectively, as node values.

6.2.1.1 Concept Descriptions

Individual concepts of the module *physical_dimension* are defined below. For the exhaustive enumeration of *base dimension* individuals, we refer to Morbach et al (2008h). Alternatively, Chertov (1997) provides comprehensive definitions of such individuals.

Class Descriptions

Base dimension
Most *physical dimensions* can be mathematically derived from a small set of dimensions that we call *base dimensions*. Such a set of *base dimensions* is chosen by convention. In OntoCAPE, we adopt the base dimensions of the SI system of units (BIPM 2006), which are *length*, *time*, *thermodynamic temperature*, *mass*, a*mount of substance*, *electric current*, and *luminous intensity*.
Formal definition: A *base dimension* is one of the following individuals: **amount_of_substance**, **electric_current**, **length**, **luminous_intensity**, **mass**, **thermodynamic_temperature**, or **time**.

Derived dimension
A *derived dimension* is a *physical dimension* that can be defined as a product of powers of the *base dimensions*. For example, the *derived dimension* **velocity** can be defined as the ratio of the base dimensions **length** and **time**.

Supplementary dimension
This class subsumes fundamental dimensions that do not form part of the SI system of units and are therefore not classified under the *base dimension* class.

Relation Descriptions

defines
The relation defines links a *note* to a *derived dimension*, which represents the right hand side of a definition equation for the *derived dimension*.

6.2.2 Derived Dimensions

The ontology module *derived_dimensions* provides a number of frequently used *derived dimensions*. These predefined *derived dimensions* (cf. Fig. 6.11) are classified into categories which have been suggested by Chertov (1997):

- The class *space and time* subsumes those *physical dimensions* that can be derived from the *base dimensions* **length** and **time**.
- *Periodic phenomena* assembles *derived dimensions* with a periodic character, such as **frequency** or **period**.
- *Mechanics* subsumes *derived dimensions* that are relevant for the field of mechanics.
- Similarly, the instances of the class *thermodynamics* are of relevance in the area of thermodynamics and transport phenomena.
- Finally, *electricity and magnetism* is intended to subsume *derived dimensions* that are connected with the phenomena of electricity or magnetism.

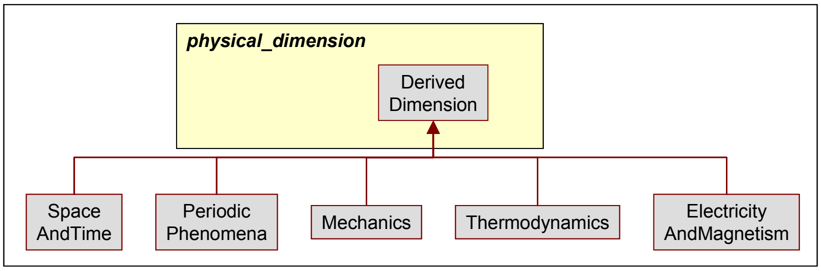

Fig. 6.11: Classification of *derived dimensions*, according to Chertov (1997)

6.2.2.1 Concept Descriptions

Individual concepts of the module *derived_dimensions* are defined below. For a subset of corresponding individuals, we refer to Morbach et al. (2008h). Alternatively, Chertov (1997) provides comprehensive definitions of such individuals.

Class Descriptions

Electricity and magnetism
The class *electricity and magnetism* subsumes *derived dimensions* that are connected with the phenomena of electricity or magnetism.

Thermodynamics
The class *thermodynamics* subsumes *derived dimensions* that are of relevance in the area of heat transfer and thermodynamics.

Mechanics
The class *mechanics* subsumes *derived dimensions* that are of relevance for the field of mechanics.

Periodic phenomena
The class *periodic phenomena* subsumes *derived dimensions* with a periodic character, such as **frequency** or **period**.

Space and time
The class *space and time* subsumes the *physical dimensions* that can be derived from the *base dimensions* **length** and **time**.

6.3 SI Unit (Partial Model)

The partial model **SI_unit** comprises the ontology modules *SI_unit* and *derived_SI_units* (cf. Fig. 6.12). The former module introduces the set of base units of the SI system, and it establishes a mechanism to derive further units from these. The latter module defines a number of frequently used derived SI units by applying this mechanism.

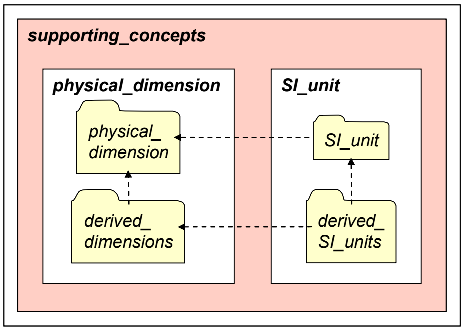

Fig. 6.12: Overview on the partial model **SI_unit**

6.3.1 SI Unit (Ontology Module)

The ontology module *SI_unit* establishes the International System of Units, which is also known as the SI system of units (cf. BIPM 2006). According to Gruber and

Olsen (1994), a *system of units* is "a class of units defined by composition from a base set of units, such that every instance of the class is 'standard' unit for a physical dimension and every physical dimension has an associated unit."
Some units are designated as *fundamental units* or *base units*, meaning that all other units can be derived from them.

base dimension	**base unit**
lenght	meter
mass	kilogram
time	second
electric current	ampere
thermodynamic temperature	kelvin
amount of substance	mole
Luminous intensity	candela

Fig. 6.13: The base dimensions and the corresponding base units of the SI system

There is no intrinsic property that makes a unit fundamental; rather, a system of units defines a set of orthogonal base dimensions and assigns a base unit to each of them. Fig. 6.13 shows a listing of the base dimensions and the corresponding base units of the SI system.
Fig. 6.14 shows how the SI system of units is implemented in OntoCAPE.

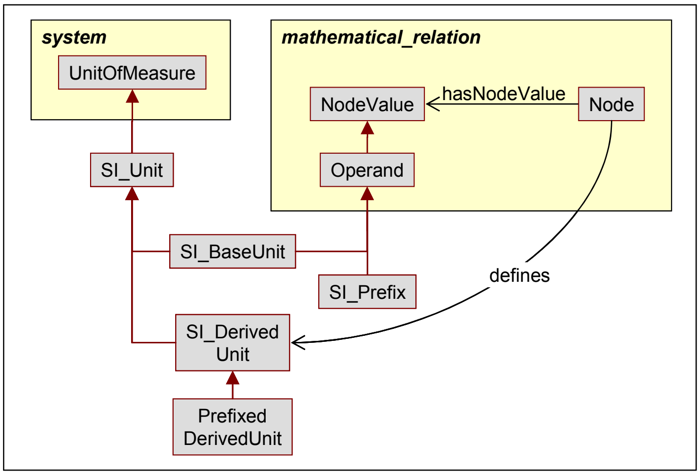

Fig. 6.14: Overview on the ontology module *SI_unit*

The *unit of measure* class has already been established in the ontology module *system*. Now, an *SI unit* is introduced as a special type of *unit of measure* that complies with the SI system; formally, an *SI unit* is defined as the exhaustive enumeration of its subclasses *SI base unit* and *SI derived unit*.

- The class *SI base unit* subsumes the seven base units of the SI system, namely **A** (ampere), **cd** (candela), **K** (kelvin), **kg** (kilogram), **m** (meter), **mol** (mole), and **s** (second).
- A *derived SI unit* is a *SI unit* that can be derived from one or several of the *SI base units* by means of multiplication and exponentiation operations.

Additionally, the classes *SI prefix* and *SI derived unit* are introduced: A *SI prefix* represents a decimal power by which a *SI unit* is multiplied; that way, one obtains a *prefixed derived unit*, which is a multiple or submultiple of the original *unit of measure*. So far, the following 20 prefixes have been approved by the General Conference on Weights and Measures: yotta, zetta, exa, peta, tera, giga, mega, kilo, hecto, deca, deci, centi, milli, micro, nano, pico, femto, atto, zepto, yocto (cf. BIPM 2006).The definition equation for a particular *derived SI unit* can be indicated through the same mechanism that has already been used for the definition of *derived dimensions* (cf. Sect. 6.2.2): A *SI derived unit* is defined by a *node* (cf. ontology module *mathematical_relation*, Sect. 6.1), which represents the root of a definition tree; the leaves of the definition tree have *SI base units* or *SI prefixes* as *node values*. To this end, *SI base unit* and *SI prefix* are declared to be subclasses of *operand*; that way, their instances may appear as operands in the definition equation. An example is given in Fig. 6.15: The *SI derived unit* **m_per_s** is defined by the *node* **Meter_Per_ Second**. In turn, **Meter_Per_Second** is further specified through its two child *nodes*, **Meter** and **Second**, which are concatenated by the *binary operator* **divide**. **Meter** and **Second** have the *SI base units* **m** and **s**, respectively, as node values.

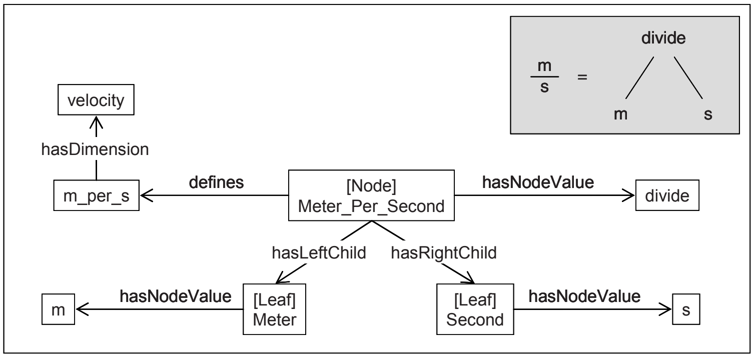

Fig. 6.15: Definition of the *derived SI unit* '**m_per_s**'

A more complex example, the definition tree for a Newton (**N**), is shown in Fig. 6.16. Note that these derivation as shown in Fig. 6.15 and Fig. 6.16 are defined in *derived_units*.

As explained before, the suggested mechanism for the indication of definition equations scales badly with current reasoners. Therefore, one should abstain from such definitions, unless they are definitely required by the respective application.

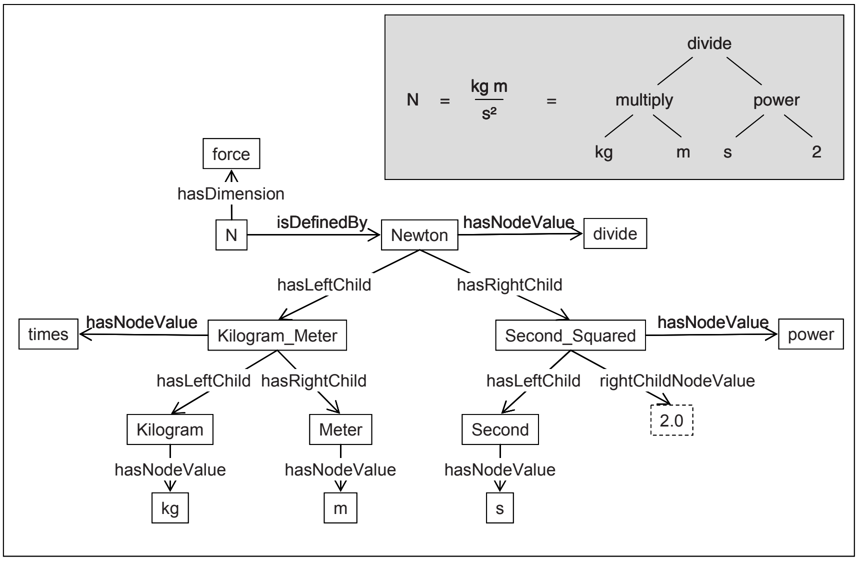

Fig. 6.16: Definition of the *derived SI unit* '**N**'

6.3.1.1 Concept Descriptions

Individual concepts of the module *SI unit* are defined below. For an extensive description of *SI units* and *SI prefices*, we refer to Morbach et al. (2008h). Alternatively, BIPM (2006) provides comprehensive definitions of SI units.

Class Descriptions

Prefixed derived unit
A *prefixed derived unit* is an *SI unit* with an *SI prefix*. Examples are **kJ** (kilo-joule), **hPa** (hecto-pascal), or **mm** (milli-meter).
Formal definition: A *prefixed derived unit* defines a *node*, the left child *node* of which has an *SI prefix* as a node value.

SI base unit
The seven *base units* of the SI system are: ampere, candela, kelvin, kilogram, meter, mole, and second (BIPM 2006).
Formal definition: An *SI base unit* is one of the following individuals: **A**, **cd**, **K**, **kg**, **m**, **mol**, **s**.

SI derived unit
"Derived units are units which may be expressed in terms of base units by means of the mathematical symbols of multiplication and division. Certain derived units

have been given special names and symbols, and these special names and symbols may themselves be used in combination with those for base and other derived units to express the units of other quantities" (BIPM 2006).

SI prefix
An *SI prefix* can be used to prefix any *SI unit* to produce a multiple or submultiple of the original unit (BIPM 2006). So far, the following 20 prefixes have been approved by the General Conference on Weights and Measures: yotta, zetta, exa, peta, tera, giga, mega, kilo, hecto, deca, deci, centi, milli, micro, nano, pico, femto, atto, zepto, yocto.
Formal definition: An *SI prefix* is one of the following individuals: **yotta**, **zetta**, **exa**, **peta**, **tera**, **giga**, **mega**, **kilo**, **hecto**, **deca**, **deci**, **centi**, **milli**, **micro**, **nano**, **pico**, **femto**, **atto**, **zepto**, **yocto**.

SI unit
An *SI unit* is *unit of measure* that complies with the SI system of units (cf. BIPM 2006).
Formal definition: An *SI unit* is either an *SI base unit* or an *SI derived unit*.

Relation Descriptions

defines
The relation defines links a *node* to an *SI derived unit*, which represents the right hand side of a definition equation for the *SI derived unit*.

6.3.2 Derived SI Units

The ontology module *derived_SI_units* establishes a number of frequently used derived SI units and provides the corresponding definition trees. For details, we refer to the formal specification of OntoCAPE.

6.4 Space and Time

The ontology module *space_and_time* introduces spatial and temporal coordinate systems and provides concepts for the representation of spatial and temporal points as well as periods of time.
The concept of a *spatial coordinate system* is introduced as a special type of *coordinate system* (cf. Fig. 6.17). A *spatial coordinate system* has *spatial coordinate system axes*, which may be either *Cartesian* or *curvilinear coordinate system axes*. The ontology module provides some predefined axes, like the **x-**, **y-**, and **z-axis** of a Carte-

sian coordinate system. Moreover, a *spatial coordinate system* has up to three *spatial coordinates* (depending on whether the system is intended for 1D, 2D, or 3D space); these are either *straight coordinates* (representing a distance) or *angular coordinates* (representing an angle). A *spatial point* is represented through up to three *spatial coordinates*, again depending on the dimensionality of the considered space.

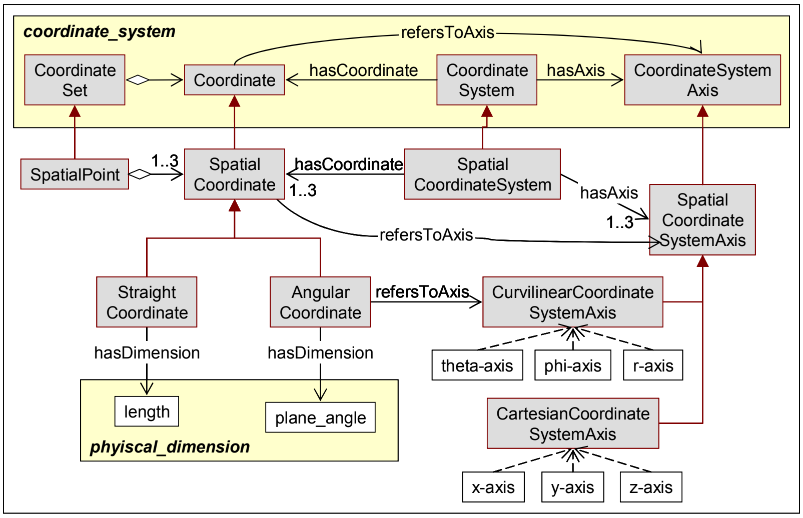

Fig. 6.17: Spatial coordinates

Some special types of *spatial coordinate systems* are introduced in Fig. 6.18; each type is assigned its respective *coordinate system axes*.

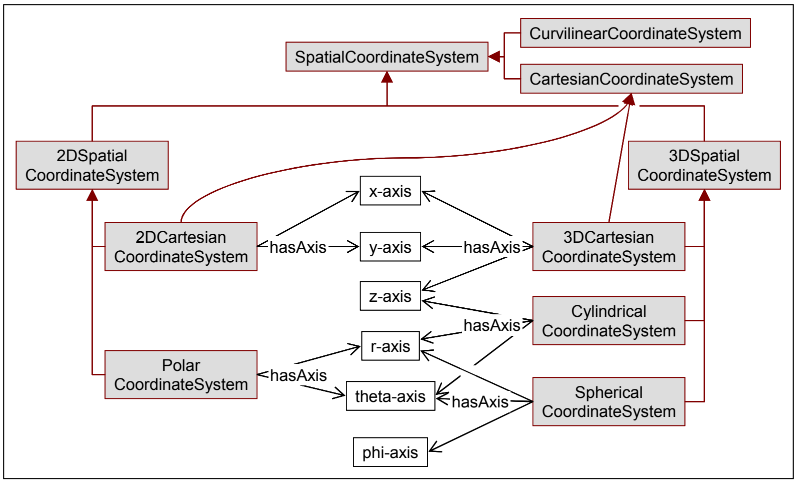
CurvilinearCoordinateSystem
SpatialCoordinateSystem
CartesianCoordinateSystem
2DSpatial CoordinateSystem
3DSpatial CoordinateSystem
x-axis
2DCartesian CoordinateSystem
hasAxis
y-axis
hasAxis
3DCartesian CoordinateSystem
z-axis
hasAxis
Cylindrical CoordinateSystem
r-axis
Polar CoordinateSystem
hasAxis
theta-axis
hasAxis
Spherical CoordinateSystem
phi-axis

Fig. 6.18: Types of spatial coordinate systems

The *spatial coordinate systems* are classified from two perspectives: The first perspective differentiates coordinate systems for 2D and 3D space, while the second perspective distinguishes curvilinear and Cartesian coordinate systems.
An application example for a *polar coordinate system* is given in Fig. 6.19.

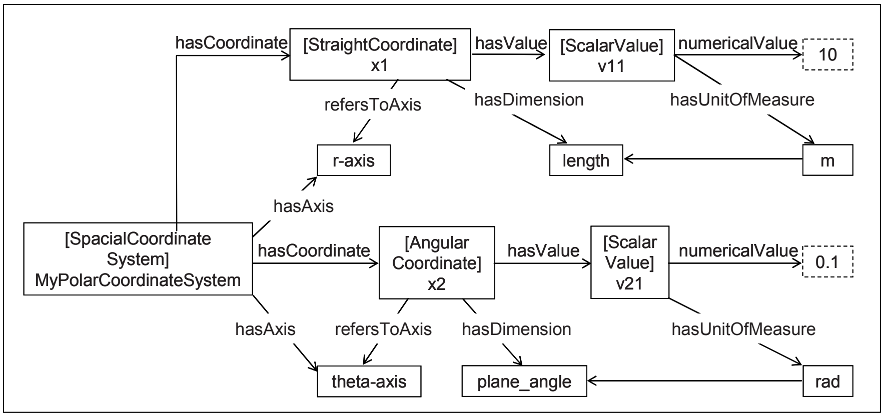

Fig. 6.19: Application example for a *polar coordinate system*

The *spatial coordinate systems* are explicitly classified along the first dimension, i.e., they are categorized as either *2D* or *3D spatial coordinate system*[75].
As shown in Fig. 6.20, the class *temporal coordinate system* is defined analogously to *spatial coordinate system*: A *temporal coordinate system* has one *temporal coordinate*, which refers to a *temporal coordinate system axis*. The **t-axis** is defined as the default axis of a *temporal coordinate system*.
An important *temporal coordinate system* is the **UTC-System**. UTC stands for Coordinated Universal Time and denotes an international time standard, which is disseminated by the International Bureau of Weights and Measures (BIPM 2007). The *temporal coordinate* of the **UTC-System** is named **CoordinatedUniversalTime**. Its *coordinate value* has the *unit* **UTC**, and the numericalValue attribute of the *coordinate value* should be specified in the format of the XML datatype *dateTime* (Biron and Malhotra 2004).

[75] Implementation advise: According to the principle of ontology normalization (cf. Chap. 4), the affiliation to the second dimension should be indirectly defined via necessary and sufficient conditions. This principle is followed in the case of *curvilinear coordinate systems* (i.e., the class *curvilinear coordinate system* is defined through necessary and sufficient conditions, such that the subclasses of *curvilinear coordinate system* can be automatically inferred by a reasoner). However, it is not appropriate to define the class *Cartesian coordinate system* through necessary and sufficient conditions, since such a definition would severely deteriorate the reasoner performance. Thus, deviating from the principle of ontology normalisation, multiple classification is applied here.

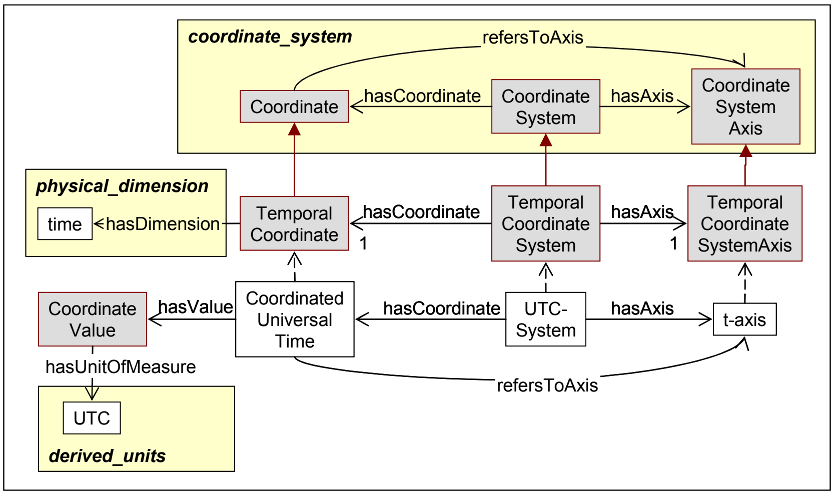

Fig. 6.20: Temporal coordinates

A *time period* is a *scalar quantity* whose *scalar value* denotes the temporal duration of a period of time. Optionally, the starting time of the *time period* can be indicated; to this end, the *scalar value* of the *time period* refers to the *value* of a *temporal coordinate* via the relation hasStartingTime (cf. Fig. 6.21).

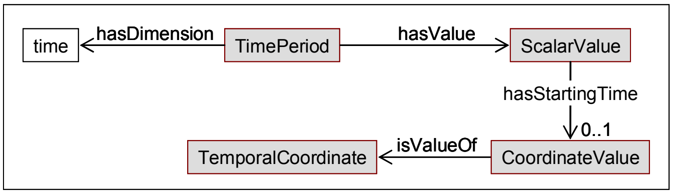

Fig. 6.21: Representation of a time period with a definite starting time

An application example is shown in Fig. 6.22.

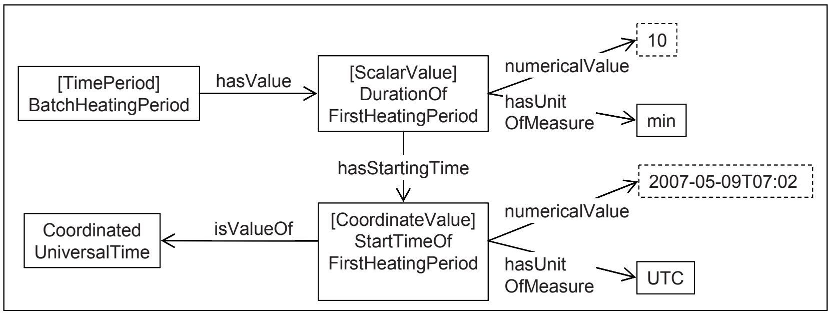
[TimePeriod] BatchHeatingPeriod
hasValue
[ScalarValue] DurationOf FirstHeatingPeriod
numericalValue
10
hasUnit OfMeasure
min
hasStartingTime
Coordinated UniversalTime
isValueOf
[CoordinateValue] StartTimeOf FirstHeatingPeriod
numericalValue
2007-05-09T07:02
hasUnit OfMeasure
UTC

Fig. 6.22: Application of the concepts *time period* and *temporal coordinate*

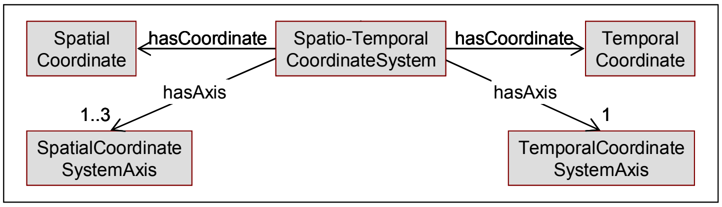
Spatial Coordinate
hasCoordinate
Spatio-Temporal CoordinateSystem
hasCoordinate
Temporal Coordinate
hasAxis
1..3
SpatialCoordinate SystemAxis
hasAxis
1
TemporalCoordinate SystemAxis

Fig. 6.23: Spatio-temporal coordinate system

Finally, the concept of a *spatio-temporal coordinate system* is introduced, which denotes a *coordinate system* that has both *spatial* and *temporal coordinates* (Fig. 6.23).

6.4.1 Concept Descriptions

Individual concepts of the module *space_and_time* are defined below. For an extensive description of the different instances *of coordinate system axis*, we refer to Morbach et al. (2008h).

Class Descriptions

2D Cartesian coordinate system
A *2D Cartesian coordinate system* is an orthogonal *planar coordinate system* that has two straight, perpendicular axes: the **x-axis** (a.k.a. abscissa) and the **y-axis** (a.k.a. ordinate). A *2D Cartesian coordinate system* has a positive orientation (i.e., the **x-axis** points right and the **y-axis** points up).

3D Cartesian coordinate system
A *3D Cartesian coordinate system* is an orthogonal *3D spatial coordinate system* that has three straight, perpendicular axes: the **x-axis**, the **y-axis**, and the **z-axis**. A *3D Cartesian coordinate system* has a positive (right-handed) orientation; that is, the *xy*-plane is horizontal, the **z-axis** points up, and the **x-axis** and the **y-axis** form a positively oriented *2D Cartesian coordinate system* in the *xy*-plane if observed from above the *xy*-plane.

3D spatial coordinate system
A *3D spatial coordinate system* is a *spatial coordinate system* for describing positions in 3D space.

Angular coordinate
An *angular coordinate* is an angle that acts as a *spatial coordinate*. An *angular coordinate* is a *spatial coordinate* that has the *physical dimension* of **plane_angle**.

Cartesian coordinate system
A *Cartesian coordinate system* is a *spatial coordinate system*, the coordinate surfaces of which are planes (in 3D) or straight lines (in 2D).
Formal definition: A *Cartesian coordinate system* is a *spatial coordinate system* that has (1) some *Cartesian coordinate system axes* and (2) only *Cartesian coordinate system axes.*

Cartesian coordinate system axis
A *Cartesian coordinate system axis* is an axis of a *Cartesian coordinate system.*

Curvilinear coordinate system
A *curvilinear coordinate system* is a *spatial coordinate system* the coordinate surfaces of which are curved surfaces (in 3D) or curved lines (in 2D).
Formal definition: A *curvilinear coordinate system* is a *spatial coordinate system* that has some *curvilinear coordinate system axes.*

Curvilinear coordinate system axis
A *curvilinear coordinate system axis* is an axis of a *curvilinear coordinate system.*

Cylindrical coordinate system
A *cylindrical coordinate system* is an orthogonal *3D spatial coordinate system* that has cylindrical coordinates (i.e., *radius*, *height*, and *azimuth angle*). It is especially suited to describe positions on rotationally symmetrical shapes like cylinders or cones.

Planar coordinate system
A *planar coordinate system* is a *spatial coordinate system* for describing positions located on a two-dimensional plane.

Polar coordinate system
A *polar coordinate system* is a *planar coordinate system* that has polar coordinates (i.e., *radius* and *polar angle*). It is especially suited for describing positions on a circle or ellipse.

Spatial coordinate
A *spatial coordinate* is a *coordinate* that denotes a spatial position.
Formal definition: A *spatial coordinate* is either a *straight coordinate* or an *angular coordinate.*

Spatial coordinate system
A *spatial coordinate system* is a *coordinate system* for describing spatial positions.

Spatial coordinate system axis
A *spatial coordinate system axis* is the *coordinate system axis* of some *spatial coordinate system.*

Spatio-temporal coordinate system
A *spatio-temporal coordinate system* denotes positions in space and time.

Spatial point
A *spatial point* is a point in space; it is represented through a *coordinate set* comprising up to 3 *spatial coordinates*.

Spherical coordinate system
A *spherical coordinate system* is an orthogonal *3D spatial coordinate system* that has spherical coordinates (i.e., *radius*, *azimuth angle*, and *zenith angle*). It is especially suited for describing positions on a sphere or spheroid.

Straight coordinate
A *straight coordinate* is a distance that acts as a *spatial coordinate*.
Formal definition: A *straight coordinate* is a *spatial coordinate* that has the *physical dimension* of **length**.

Temporal coordinate
A *temporal coordinate* is a *coordinate* that denotes a temporal position.

Temporal coordinate system
A *temporal coordinate system* is a *coordinate system* for describing temporal positions.

Temporal coordinate system axis
A *temporal coordinate system axis* is the *coordinate system axis* of some *temporal coordinate system*.

Time period
A *time period* is a *scalar quantity* that denotes the temporal duration of a period of time. Optionally, the starting time of the *time period* can be indicated.

Relation Descriptions

hasStartingTime
Indicates the starting time of a *time period*.

6.5 Geometry

The module *geometry* provides the concepts for describing the shapes and main dimensions of simple geometric figures. Two major classes of figures are introduced, which are both defined as subclasses of *system*: *solids* and *surfaces*.

- A *solid* (a.k.a. geometric solid or solid geometric figure) is a bounded three-dimensional geometric figure in Euclidean space.
- A *surface* is a bounded geometric figure in a two-dimensional submanifold of three-dimensional Euclidean space.

Solids and *surfaces* may have certain geometric *properties*. Some of them are *scalar quantities*, such as, *diameter*, *radius*, or *volume* (the latter is only defined for *solids*, cf. Fig. 6.24); others, such as *edge length*, *height*, and *surface area*, may alternatively be a *scalar quantity* or a *vector quantity*.

– As a *scalar quantity*, these quantities simply indicate a size – *surface area*, for instance, indicates either the area of a *surface* or of (one of) the exterior surface(s) of a *solid*.
– As a *vector quantity*, the quantities additionally indicate the orientation of the respective line or surface – in case of the *surface area*, the vector would have the same orientation as the surface normal, while the Euclidean norm of the vector would equal the area of the surface.

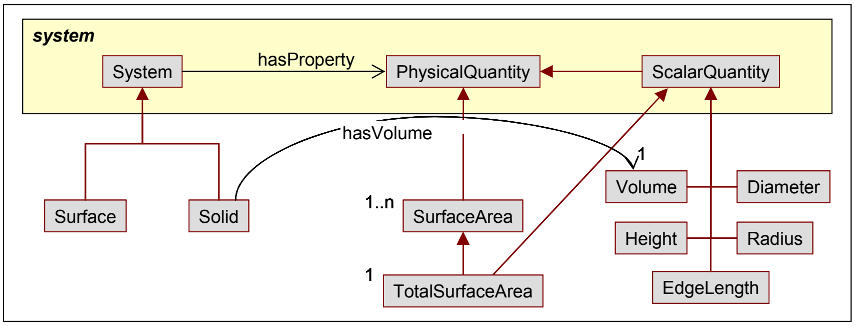

Fig. 6.24: Key concepts of *geometry*

There are two further specializations of the *surface area* concept:

– *Side area*, which is only defined for *solids*, corresponds to one particular exterior surface of a *solid*. This concept should be applied if the *solid* has several distinguishable exterior surfaces (as e.g., in the case of a *cuboid*).
– The *total surface area* indicates the total area of either a *surface* or of (all) the exterior surface(s) of a *solid*. A *total surface area* is always a scalar.

The relations hasArea, hasLength, and hasVolume are introduced as specializations of hasProperty (Fig. 6.25). However, these relations are merely auxiliary constructs used as replacements for qualified number restrictions. They will drop out again, as soon as qualified number restrictions are made available in OWL.

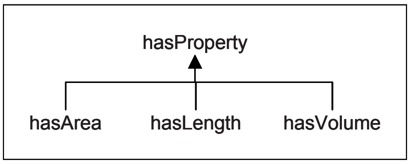

Fig. 6.25: Specializations of hasProperty as workarounds for missing QNR

In the following, some special types of *solids* and *surfaces* and their geometric properties will be introduced. Note that most of the terminology as well as the textual definitions for these concepts have been adopted from the interactive mathematics dictionaries *Mathwords* (Simmons 2007) and *MathWorld* (Weisstein 2007).
Disk and *rectangle* are two special kinds of *surfaces*, which have the following properties: a *disk* has exactly one length of type *diameter* or *radius*, whereas a rectangle has exactly two lengths of type *edge length* (cf. Fig. 6.26).

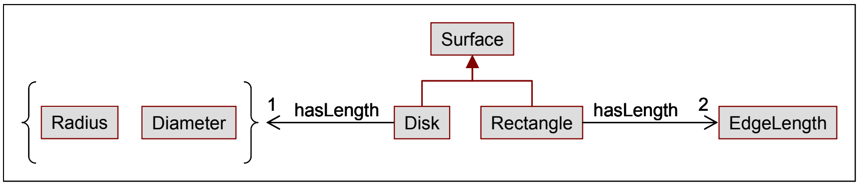

Fig. 6.26: Disk and Rectangle

Cuboid and *sphere* are special types of *solids*, for which the following properties are defined (cf. Fig. 6.27):

– A *sphere* has exactly one length of type *radius* or *diameter* as well as one area of type *total surface area*.
– A *cuboid* has exactly three lengths of type *edge length* as well as three areas of type *side area*.

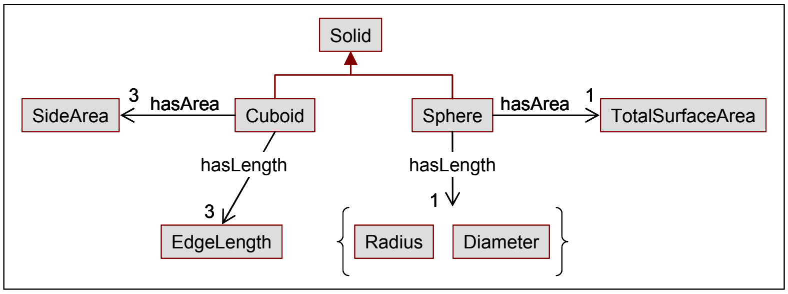

Fig. 6.27: *Cuboid* and *sphere*

A *cylinder* is a *solid* with parallel congruent bases. The bases can be shaped like any closed plane figure (not necessarily a circle) and must be oriented identically. In order to differentiate the bases from the other exterior surfaces of the *cylinder*, two specializations of *side area* are introduced, namely *base area* and *lateral surface area* (cf. Fig. 6.28). Then it can be stated that a *cylinder* has exactly two *base areas* and at least one *lateral surface area*. Moreover, a *right circular cylinder*, which is a *cylinder* with circular bases that are aligned one directly above the other, has exactly one *lateral surface area*.

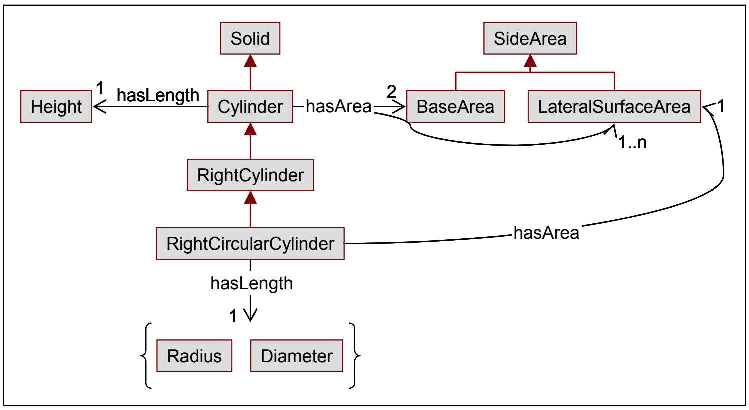

Fig. 6.28: *Types of cylinders*

The last type of *solid* introduced here is the *cone*, which is a figure with a single base tapering to an apex. If the apex is aligned directly above the center of the base, the cone can be classified as a *right cone*. Furthermore, if the base of a *right cone* takes the form of a circle, it can be classified as a *right circular cone*, which has exactly one *lateral surface area* (cf. Fig. 6.29).

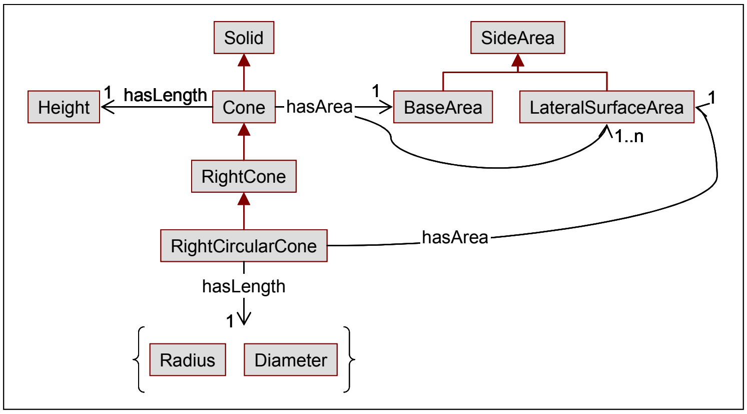

Fig. 6.29: Types of *cones*

6.5.1 Usage

There are two alternative ways to specify the geometry of a *system*. The first alternative is to summarize all the geometric aspects of the *system* in a separate *aspect system*. To this end, the *geometry* module provides the relations hasShapeRepresentation and hasSurfaceGeometry, which are specializations of the relation hasAspectSystem. These relations may be used to link a *solid* or a *surface* to a *system* (cf. Fig. 6.30); a reasoner will then infer that the respective *solid* or *surface* is an *aspect system* of the main *system*.

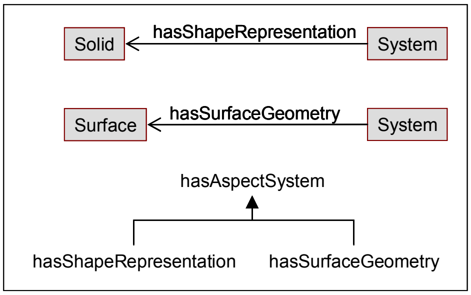

Fig. 6.30: Representation alternative 1: The geometric properties are summarized in an *aspect system*

An example is shown in Fig. 6.31: The shape of the distillation column **Column C1** is represented by the individual **Shape of C1**. As indicated by the brackets in Fig. 6.31, **Column C1** is an instance of *system*, while **Shape of C1** is an instance of *right circular cylinder*. The two individuals are linked via the relation hasShapeRepresentation; thus, it can be inferred that **Shape of C1** is an *aspect system* of the main *system* **Column C1**.
Shape of C1 is further characterized through its properties and their values. Exemplarily shown is its *height* **H_C1**, which has a value of 10 m.

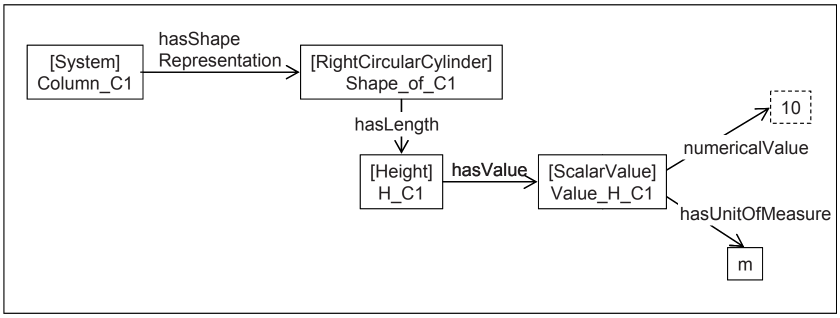

Fig. 6.31: Application example of representation alternative 1

Depending on the application, the above representation alternative may be too complicated for practical use. In the following, a more simple alternative is presented. This alternative can be applied if the shape of the respective *system* is obvious from the context or a matter of common knowledge – for instance, one may safely assume that a distillation column, unless otherwise indicated, has the shape of a *right circular cylinder*. In such a case, the description of the *system*'s geometry is simply a matter of specifying its main dimensions, which can be done by assigning the geometric *properties* directly to the *system*. This is exemplarily shown in Fig. 6.32, where the *height* **H_C1** is directly assigned to the *system* **Column C1**.

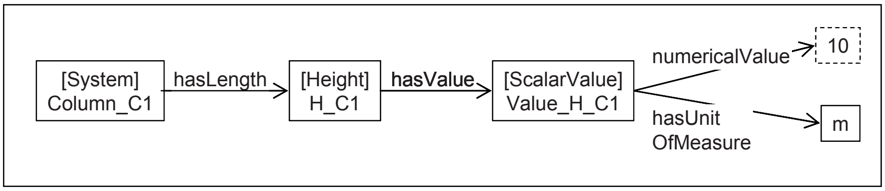

Fig. 6.32: Application example of representation alternative 2

As a further simplification, the geometric *properties* may be replaced by specializations of the hasCharacteristic relation (cf. Sect. 5.1.14). This approach is taken in module *plant* (cf. Sect. 8.3.1).
With respect to information sharing, the users certainly favorable to agree on one convention (i.e. alternative1 or alternative 2) to avoid misconceptions.

6.5.2 Concept Descriptions

Individual concepts of the module *geometry* are defined below.

Class Descriptions

Base area
The base is the bottom of a *solid*. If the top is parallel to the bottom (as in a trapezoid or prism), both the top and bottom are called bases. The *base area* is the *surface area* of (one of) the base(s).

Cone
A three-dimensional figure with a single base tapering to an apex. The base can be any simple closed curve (Simmons 2007).

Cuboid
A closed box composed of three pairs of rectangular faces placed opposite each other and joined at right angles to each other, also known as a rectangular parallelepiped (Weisstein 2007).

Cylinder
A *solid* with parallel congruent bases. The bases can be shaped like any closed plane figure (not necessarily a circle) and must be oriented identically (Simmons 2007).

Diameter
The length of a line segment between two points on a circle or *sphere* which passes through the center of the circle or *sphere*.

Disk
A *disk* is the union of a circle and its interior (Simmons 2007). A circle is given by the set of points in a plane that are equidistant from a given point.

Edge length
The length of a (straight) edge of a *surface* or *solid*.

Height
The shortest distance between the base of a geometric figure and its top, whether that top is an opposite vertex, an apex, or another base (Simmons 2007).

Lateral surface area
The *surface area* of a single lateral surface of a *solid* (i.e., any *side area* that is not a *base area*).

Radius
The length of the line segment between the center and a point on a circle or *sphere*.

Rectangle
A *rectangle* is a box shape on a plane. Formally, a *rectangle* is a quadrilateral with four congruent angles (all 90°) (Simmons 2007).

Right circular cone
A *right cone* with a base that is a circle.

Right circular cylinder
A *right cylinder* with bases that are circles (Simmons 2007).

Right cone
A *cone* that has its apex aligned directly above the center of its base (Simmons 2007).

Right cylinder
A *cylinder* which has bases aligned one directly above the other (Simmons 2007).

Side area
The area of one particular exterior surface of a *solid*. This concept should be applied only if the *solid* has several distinguishable exterior surfaces. Otherwise (e.g., for a *sphere*) use *total surface area*. A *side area* can either be a *scalar quantity* or a *vector quantity*. In case of the latter, the vector is perpendicular to the surface (i.e., it has the same orientation as the surface normal), and its Euclidean norm equals the area of the surface.

Sphere
A *solid* consisting of all points equidistant from a given point. This point is the center of the *sphere* (Weisstein 2007).

Solid
A *solid* (a.k.a. geometric solid or solid geometric figure) is a collective term for all bounded three-dimensional geometric figures. This includes polyhedra, pyramids, prisms, cylinders, cones, spheres, ellipsoids, etc. (Simmons 2007).

Surface
A *surface* is a two-dimensional submanifold of three-dimensional Euclidean space.

Surface area
The area of a *surface* or of (one of) the exterior surface(s) of a *solid*. More precisely, the class alternatively denotes

- the area of a *surface*, or
- the area of a single exterior surface of a *solid*, or
- the total area of the exterior surface(s) of a *solid*.

A *surface area* is either a *side area* or a *total surface area*.

Total surface area
The total area of a *surface* or of (all) the exterior surface(s) of a *solid*.

Volume
The total amount of space enclosed in a *solid* (Simmons 2007).

Relation Descriptions

hasArea
Workaround for Qualified Cardinality Restriction (QCR) (cf. Sect. 2.3.4) which is a feature of modeling currently not available from OWL.

hasLength
Workaround for Qualified Cardinality Restriction (QCR) (cf. Sect. 2.3.4) which is a feature of modeling currently not available from OWL.

hasVolume
Workaround for Qualified Cardinality Restriction (QCR) (cf. Sect. 2.3.4) which is a feature of modeling currently not available from OWL.

hasShapeRepresentation
The relation hasShapeRepresentation points from a *system* to the *solid* that represents its geometry.

hasSurfaceGeometry
The relation hasSurfaceGeometry points from a *system* to the *surface* that represents its geometry.

representsShapeOf
The relation representsShapeOf points from a *solid* to the *system* whose geometry the *solid* represents.

representsSurfaceGeometryOf
The relation representsSurfaceGeometryOf points from a *surface* to the *system* whose geometry the *surface* represents.

6.6 References

BIPM (2006) *The International System of Units (SI)*, 8th edition. SI brochure, published by the International Committee for Weights and Measures (Bureau International des Poids et Measures, BIPM). Online available at http://www.bipm.fr/en/si/si_brochure/. Accessed September 2007.

BIPM (2007) *BIPM: Bureau International des Poids et Mesures*. Website, available at www.bipm.org. Accessed October 2007.

Biron PV, Permanente K, Malhotra A (2004) *XML Schema Part 2: Datatypes Second Edition.* W3C Recommendation. Online available at http://www.w3.org/TR/xmlschema-2/. Accessed January 2007.

Carlisle D, Ion P, Miner R, Poppelier N, eds. (2003) *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*. W3C Recommendation, online available at http://www.w3.org/TR/MathML2/. Accessed September 2007.

Chertov AG (1997) Units of physical measure. In: Grigoriev IS, Meilikhov EZ (eds.): *Handbook of Physical Quantities*, CRC Press.

Daintith J (2005) *Oxford Dictionary of Physics.* Oxford University Press.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

Morbach J, Yang A, Wiesner A, Marquardt W (2008h) *OntoCAPE 2.0 – Supporting Concepts*. Technical Report (LPT-2008-26), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-26.

Simmons B (2007) *Mathwords* – website. Online available at http://www.mathwords.com/. Accessed October 2007.

Weisstein E (2007) *MathWorld* – website. Online available at http://mathworld.wolfram.com/. Accessed October 2007.

7 Material

7.1 Material (Partial model)

The partial model **material** provides an *abstract description of matter*. In this context, “matter” refers to “anything that has mass and occupies space” (Gold et al. 1982). As for the ‘abstract description’, the partial model does not consider all the characteristics of matter, but accounts only for those that are independent of the *shape*, *size*, or *amount* of a particular occurrence of matter.
The partial model **material** originates from the CLiP (cf. Sect. 11.1.2) partial model ‘**chemical_process_material**’, which is described in an article by Yang et al. (2003); several passages of this article have been included in the present documentation (mostly in paraphrased form). In comparison to CLiP, the partial model **material** has a somewhat different structure[76], and it is modeled partly differently in order to correct certain flaws and logical contradictions of the CLiP model. In addition, **material** incorporates concepts of the ChEBI ontology (EBI 2007; OLS 2006) to describe matter at the molecular level, which was not enabled by the CLiP model.

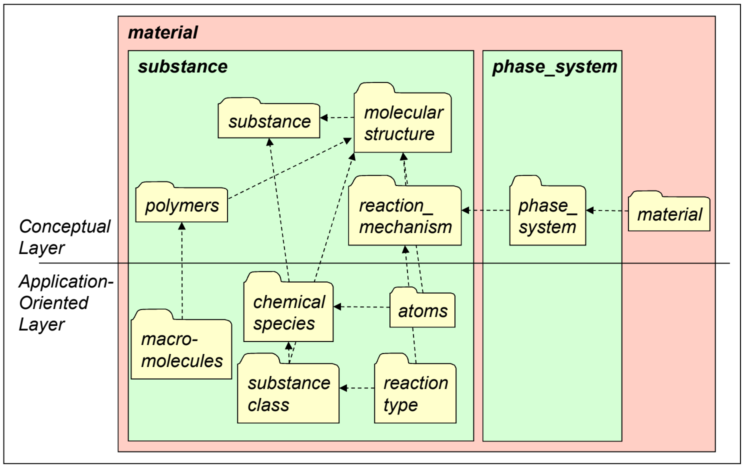

Fig. 7.1: Structure of the partial model **material**

[76] In particular, the CLiP partial model ‘mathematical model of phase system’, which forms part of the CLiP material model, has been relocated to the OntoCAPE partial model mathematical model (cf. Chap. 9).

Besides the ontology module *material*, **material** contains two further partial models, called **substance** and **phase_system** (cf. Fig. 7.1). While **substance** comprises several modules, **phase_system** consists of a single module.

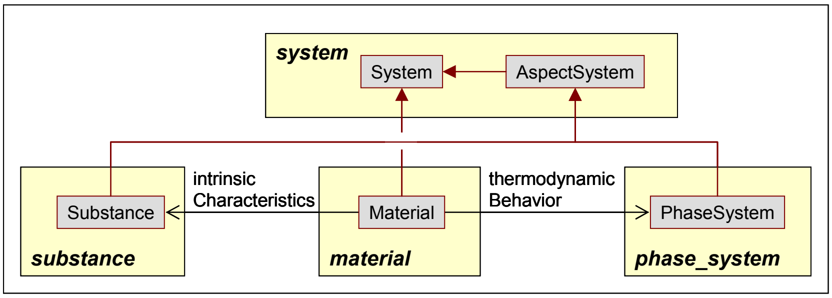

Fig. 7.2: Major classes of the partial model **material**

Material is the main class of the ontology module *material*. As shown in Fig. 7.2, *material* is a special type of *system*. A *material* can be viewed from two different perspectives, which are represented as *aspect systems* of *material*:

– *Substance* represents the intrinsic characteristics (i.e., the physicochemical nature) of a *material*.
– *Phase system* describes the macroscopic thermodynamic behavior of a *material*.

The critical concept for *substance* and *phase system* is the *physical context* (which is introduced in the ontology module *phase_system*). This class is defined as a set of independent *properties*[77] with known *values*, which is sufficient to determine other *properties* of interest; temperature, pressure, and molar concentrations are examples of *properties* that constitute a *physical context*. *Material* without consideration of the *physical context* is modeled as *substance*. The *properties* of *substance* are constants, such as molecular weight or critical properties; their *values* cannot be altered by mechanical, thermo-physical or chemical processes. In contrast, *material* within a certain *physical context* is modeled as *phase system*. The *values* of the *phase system properties* are subject to the respective *physical context*. *Phase system properties* include the thermodynamic state variables and other quantities derived thereof. The two *aspect systems* are represented in separate partial models (i.e., **substance** and **phase_system**), which subdivide the partial model **material**. This partition facilitates the usage of **material** in different types of applications: The partial model **substance** can be used in applications where merely the intrinsic characteristics of *materials* are of interest. Correspondingly, applications that are interested in the thermodynamic behavior of *materials* will use the concepts of the partial model **phase_system**.

[77] often called ‘state variables’ in the thermodynamics literature

Similarly to materials, the concept of a chemical reaction can be divided into a context-dependent and context-independent part. Reaction mechanism and reaction stoichiometry are context-independent and thus form part of the partial model **substance**. The context-dependent reaction property, namely reaction equilibrium constant, is introduced in the partial model **phase_system**.

7.1.1 Relation Between 'Phase System' and 'Material Amount'

A *phase system* is an *abstraction of a concrete occurrence of matter*. By 'concrete occurrence', we mean the actual spatiotemporal setting, for example, the manufacturing of some material in a chemical plant or its usage as a construction material. The concept of a *material amount*, introduced in the partial model **CPS_behavior** (cf. Sect. 8.6), is complementary to the *phase system* concept: A *material amount* represents the concrete occurrence of matter in, for example, a chemical plant. To better understand the distinction between *phase system* and *material amount*, the idea of an 'abstraction of a concrete occurrence of matter' is further explained in the following.
The "physical prototype" of a *phase system* is an *arbitrary amount* of (static or flowing) material in an equilibrium state and with a rather *simple geometry*. A *phase system* may take the form of either a *single phase* or a *multiphase system* where the constituting *single phases* are connected through *phase interfaces* with the most plain geometry (i.e., planar surfaces). Thus, the abstraction excludes all material characteristics that depend on the shape, size, or amount of a particular occurrence of a material. Consequently, *extensive properties* as well as the distribution of *intensive properties* in space and the distribution of any *property* in time are not associated with a *phase system*. Rather, they are modeled as *properties* of *material amount*. Furthermore, a *phase system* is meant to represent only the physical properties which are applicable to material in an equilibrium state. Consequently, any properties that describe the rates of chemical reaction or transport phenomena or the spatial gradients of physical quantities involved in non-equilibrium events, are associated not with *phase system* but rather with *material amount*. For *single phases*, in particular, only the following *properties* are considered: (i) intensive thermodynamic states, and (ii) *properties* that can be determined solely from (i). For *multiphase systems*, only those *properties* are taken into account that are (i) solely dependent on the *properties* of the constituting *single phases* and (ii) not associated to a particular surface geometry other than the most plain one as mentioned above.
The distinction between a *phase system* and a *material amount* has been made not only because it is conceptually feasible, but also because it is practically useful. It was inspired by the physical properties packages that are commonly used in the field of process modeling. These packages handle the computation of such material properties without reference to amounts and geometry. Due to their independence of amount and geometry, the packages have proven to be highly reusable in

modeling various kinds of chemical processes. Correspondingly, it is expected that the concepts of the partial model **material** prove reusable for supporting the modeling of different *material amounts*.

7.1.2 Material (Ontology Module)

The only purpose of the ontology module *material* is to establish the relations between the concepts in the partial model **substance** (representing the intrinsic aspects of matter) and those in partial model **phase_system** (describing the thermodynamic behavior of matter). Thus, the module contains only very few concepts, which are listed below.

7.1.2.1 Concept Descriptions

Individual concepts of the module *material* are defined below.

Class Descriptions

Material
The class *material* represents all kinds of matter.

Relation Descriptions

intrinsicCharacteristics
The relation designates the *aspect system* which represents the intrinsic characteristics of some *material*.

thermodynamicBehavior
The relation designates the *aspect system* which represents the thermodynamic behavior of some *material*.

7.2 Substance (Partial Model)

The partial model **substance** comprises 4 ontology modules on the Conceptual Layer, and (currently) four additional modules on the Application-Oriented Layer (cf. Fig. 7.3). The following modules are located on the Conceptual Layer:

- *Substance* is the main ontology module of the **substance** partial model. It provides essential concepts for the description of pure substances and mixtures, primarily at the macroscopic scale.
- The ontology module *molecular_structure* is concerned with the characterization of pure substances at the atomic scale.
- *Polymers* supplements *molecular_structure* by concepts for the description of macromolecular structures.
- Finally, *reaction_mechanism* allows to represent the mechanism and the stoichiometry of chemical reactions.

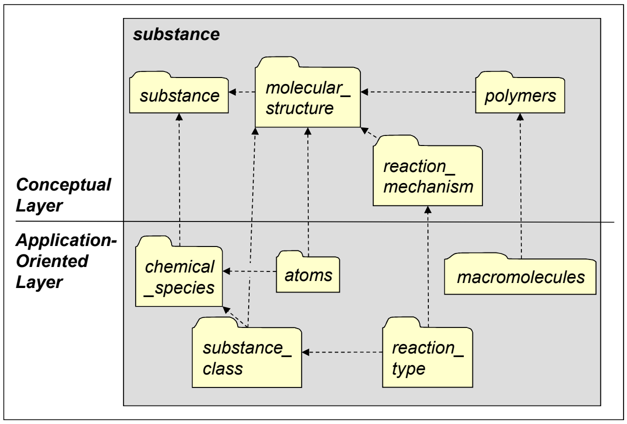

Fig. 7.3: Structure of the partial model **substance**

The Application-Oriented Layer comprises the following ontology modules:

- The module *chemical_species* instantiates concepts from the *substance* module to establish an information base about pure substances.
- *Atoms* refines *molecular_structure* by instance data about the chemical elements.
- Similarly, *macromolecules* refines *polymers* by instance data about the molecular structure of technical polymers.
- *Substance_class* categorizes chemical substances into classes with similar chemical properties, such as alcohols, esters, etc.
- *Reaction_type* describes important types of chemical reactions, like esterification or hydrohalogenation.

7.2.1 Substance (Ontology Module)

The ontology module *substance* originates from the CLiP model 'substance' (Yang et al. 2003). The major concepts of the *substance* module are shown in Fig. 7.4.

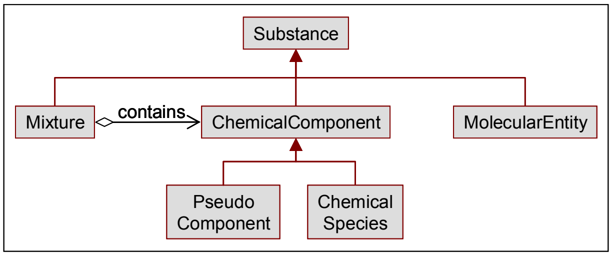

Fig. 7.4: Major concepts of *substance*

The key concept, *substance*, represents "a generalization of matter at or above the atomic level" (Yang et al. 2003). As mentioned earlier, *substance* is an *aspect system* of *material* and reflects its intrinsic, context-independent characteristics.
Three subclasses of *substance* are introduced: *Mixture* and *chemical component* both describe *substances* at the macroscopic scale, while *molecular entity* characterizes *substances* at the atomic scale.
The class *chemical component* subsumes the classes *chemical species* and *pseudo component*.

- A *chemical species* represents pure *substances* at the macroscopic scale. The IUPAC Compendium defines *chemical species* as an "ensemble of chemically identical *molecular entities* […]. The term is applied equally to a set of chemically identical atomic or molecular structural units in a solid array. […] The term is taken to refer to a set of *molecular entities* containing isotopes in their natural abundance. […] The wording of the definition […] is intended to embrace both cases such as graphite, sodium chloride or a surface oxide, where the basic structural units may not be capable of isolated existence, as well as those cases where they are." (McNaught and Wilkinson 1997).
- The class *pseudo components* is introduced alongside *chemical species*. A *pseudo component* is an auxiliary concept, which represents the averaged properties of a number of *chemical species*. *Pseudo components* are often assumed to exist in the context of physical property calculations of complex multicomponent mixtures, such as petroleum (Hariu and Sage 1969), fatty alcohols (Gutsche 1986), or polymers (Kuma and Gupta 1998).

Unlike a *pseudo component*, a *chemical species* has a specific chemical composition, which can be indicated by means of a chemicalFormula or by referring to the corresponding *molecular entity*.

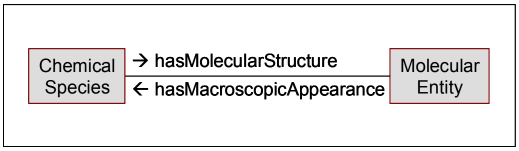

Fig. 7.5: Representation of pure substances at the macroscopic and the atomic scale

According to the IUPAC Compendium, a *molecular entity* denotes “any constitutionally or isotopically distinct atom, molecule, ion, ion pair, radical, radical ion, complex, conformer etc., identifiable as a separately distinguishable entity. Molecular entity is used […] as a general term for singular entities, irrespective of their nature, while chemical species stands for sets or ensembles of molecular entities” (McNaught and Wilkinson 1997). Thus, both *molecular entity* and *chemical species* describe pure substances – the former at the molecular scale, the latter at the macroscopic scale. The inverse relations hasMolecularStructure and hasMacroscopicAppearance interrelate a *chemical species* and its corresponding *molecular entity*; they may be used to navigate between the macroscopic and molecular perspectives (cf. Fig. 7.5).

The concept of *molecular entities* introduced in this module is further elaborated in the ontology module *molecular_structure*. This separation of macroscopic perspective (module *substance*) and molecular perspective (module *molecular_structure*) allows to characterize *substances* at the desired level of detail: For many applications in chemical engineering, the molecular structure of substances is not of interest and may therefore be omitted from an ontological description; if, on the other hand, the molecular properties are relevant to the application, they can be obtained easily by adding the ontology module *molecular_structure*.

A *mixture* is generally a *substance* that contains two or more *chemical components*. The *mixture* concept can represent two different things: a loose collection of segregate *chemical components*, or a compound material formed by several blended *chemical components*. For the latter case, subclasses of *mixture* can be introduced to denote and classify typical mixtures of *chemical components*, such as alloys, polymer blends, types of crude oil, chocolate, sand, saltwater, or air. The composition of a *mixture* is not fixed, as opposed to the composition of a *phase system* (cf. Sect. 7.3). Note that a *mixture* and its constituting *chemical species* are not connected via the hasSubsystem relation, but via the better scaling contains relation (cf. Sect. 5.1.3). This measure is taken since *mixture* and *chemical species* typically have thousands of instances – a data set of this size would cause performance problems during reasoning if its members were connected via hasSubsystem.

Since substance represents the intrinsic, context-independent characteristics of materials, the properties of substances are constants. Some exemplary chemical component constants are shown in Fig. 7.6.

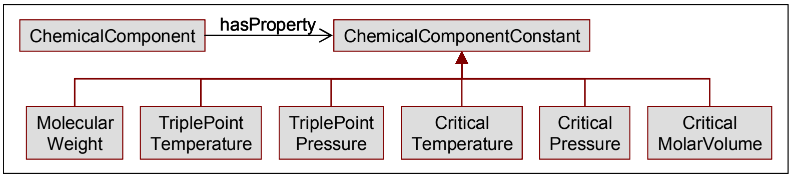

Fig. 7.6: Some *chemical component constants*

Each *substance* has at least one substanceID, such as a (trivial or systematic) name (cf. Fig. 7.7). A substanceID[78] is an unambiguous, but not necessarily a unique identifier (i.e., each substanceID represents exactly one *substance*, but a *substance* can have more than one substanceID). Unique identifiers can be explicitly categorized as uniqueSubstanceIDs. The CAS_RegistryNumber is probably the most well-known uniqueSubstanceID.

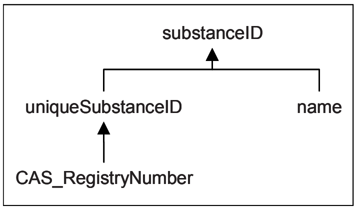

Fig. 7.7: Substance identifiers

A chemicalFormula provides information about the atoms that constitute a particular *molecular entity*. It can only be assigned to *molecular entities* and *chemical species*[79]. Different types of chemicalFormulas exist:

- An empiricalFormula indicates the relative number of each constituting chemical elements of a *molecular entity*.
- A molecularFormula specifies the (absolute) number of constituting atoms of a *molecular entity*, without indicating how they are linked.

[78] In OWL, the substanceIDs are modeled as inverse-functional attributes; uniqueSubstanceIDs are additionally characterized as functional.

[79] Conceptually, a chemicalFormula should be solely assigned to *molecular entity*. For practical usage, however, it is advantageous to assign a chemicalFormula directly to a *chemical species*, as many chemical engineering applications ignore the molecular perspective represented by *molecular entity*.

– A structuralFormula supplies information about the types of bonds and the structural arrangement of the atoms of a *molecular entity* using a linear string notation. Different formats for the representation of structural formals exist – examples are WLN (Wisswesser Line Notation), SMILES (Simplified Molecular Input Line Entry Specification), and InChI (IUPAC International Chemical Identifier).

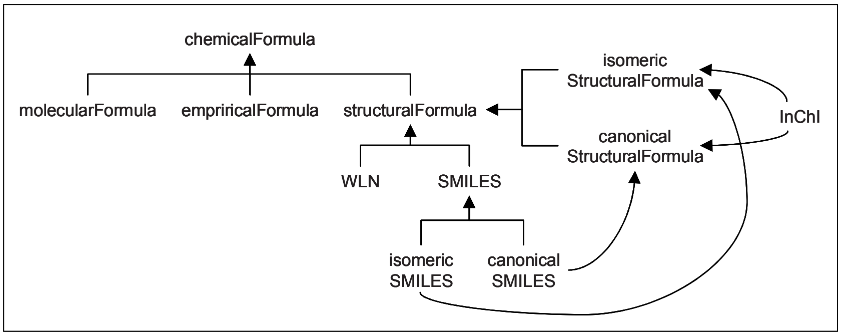

Fig. 7.8: Chemical formulas

A structuralFormula is an unambiguous but not unique (Dietz 1995) representations of a *molecular entity*. If the string representations of the structuralFormulas are generated by means of appropriate canonicalization algorithms, the representation will become unique; the canonicalSMILES notation is an example of such a canonicalStructuralFormula.

Not every structuralFormula is able to distinguish the different isomers of a *molecular entity*. Only a isomericStructuralFormula, like InChI or isomericSMILES, enables such a distinction.

Note that chemicalFormulas cannot be considered as substanceIDs since they are ambiguous (e.g., C_3H_6 could refer to Propene as well as to Cyclopropane). Even though a structuralFormula can unambiguously identify a *molecular entity*, it is ambiguous for *chemical species* (see example in Fig. 7.8) and thus not a substanceID. Similarly, a canonicalStructuralFormula is a unique identifier for a *molecular entity*, but not for a *chemical species*, and can therefore not be considered as a uniqueSubstanceID.

7.2.1.1 Usage

Instances of *chemical species* and *molecular entity* should preferably have a uniqueSubstanceID to be easily identifiable in a software application. The indication of a chemicalFormula is optional for *chemical species* but mandatory for *molecular entities*.

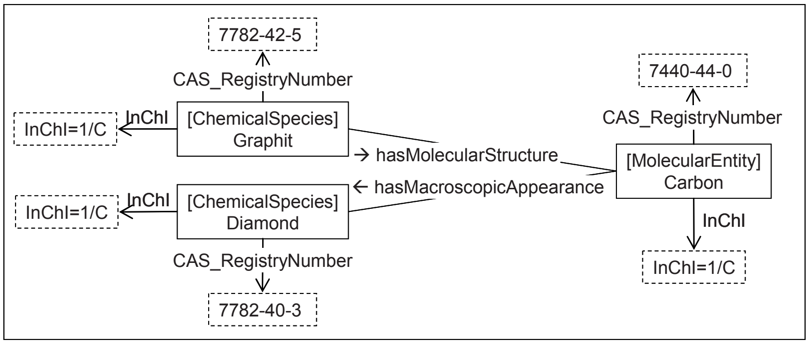

Fig. 7.9: Two different macroscopic appearances of a *molecular entity*

Note that a *molecular entity* can have different macroscopic appearances – **Carbon**, for instance, can macroscopically take the form of **Graphite** or that of a **Diamond** (Fig. 7.9). In such a case, the individuals should have different substanceIDs (here CAS_RegistryNumbers) to be distinguishable from each other. Generally speaking, the different polymorphic forms of a substance can be distinguished by different instances of substance.
More often, however, there is a one-to-one correspondence between a *chemical species* and the corresponding *molecular entity*. In this case, the same substanceID and the same instance identifier may be used for the *chemical species* and the corresponding *molecular entity* (however, the instance identifiers must have different namespace prefixes – cf. Sect. 7.2.5). This is acceptable since, in practice, the distinction between *molecular entities* and *chemical species* is often ignored. Thus, an identifier may refer to a *molecular entity* as well as to the corresponding *chemical species*.

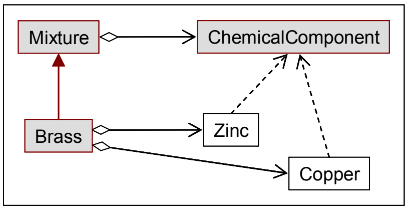

Fig. 7.10: Definition of *brass* as a subclass of *mixture*

Collections of segregated *chemical species* are represented as direct instances of the *mixture* class. By contrast, compound materials and typical mixtures are modeled as subclasses of *mixture*. The constituent *chemical species* can be indicated as part of the class definition, but this is not mandatory. Fig. 7.10 shows exemplarily the definition of *brass*; instances of *brass* would indicate the concrete occurrence of a brass material.

7.2.1.2 Concept Descriptions

Individual concepts of the module *substance* are defined below.

Class Descriptions

Chemical component
The class *chemical component* subsumes *chemical species* and *pseudo components*.
Formal definition: A *chemical component* is either a *pseudo component* or a *chemical species*.

Chemical component constant
A *chemical component constant* is a constant *property* of a *chemical component*.

Chemical species
A *chemical species* represents pure *substances* at the macroscopic scale. It consists of an "ensemble of chemically identical molecular entities […]. The term is applied equally to a set of chemically identical atomic or molecular structural units in a solid array. […] The term is taken to refer to a set of molecular entities containing isotopes in their natural abundance. […] The wording of the definition […] is intended to embrace both cases such as graphite, sodium chloride or a surface oxide, where the basic structural units may not be capable of isolated existence, as well as those cases where they are." (McNaught and Wilkinson 1997).

Critical molar volume
The *critical molar volume* is the volume of one mole of a *chemical component* at the *critical temperature* and *critical pressure*.

Critical pressure
The minimum pressure which would suffice to liquefy a substance at its *critical temperature*. Above the *critical pressure*, increasing the temperature will not cause a fluid to vaporize to give a two-phase system (McNaught and Wilkinson 1997).

Critical temperature
The temperature, characteristic of each gas, above which it is not possible to liquefy a given gas (McNaught and Wilkinson 1997).

Mixture
A *mixture* is a *substance* that contains two or more *chemical components*. The composition of a *mixture* is not fixed, as opposed to the composition of a *phase system*.

Molecular entity
The class *molecular entity* characterizes *substances* at the atomic scale. It represents "any constitutionally or isotopically distinct atom, molecule, ion, ion pair, radical, radical ion, complex, conformer etc., identifiable as a separately distinguishable entity. Molecular entity is used […] as a general term for singular entities, irrespective of their nature" (McNaught and Wilkinson 1997).

Molecular weight
The *molecular weight* of a *chemical component* is the ratio of the mass of one *molecule* of that substance, relative to the unified atomic mass unit (equal to 1/12 the mass of one atom of carbon-12). It is also known as (relative) molar mass or (relative) molecular mass.
If a *chemical component* consists of different *molecules* (as it is the case for a *polymer*), it represents the averaged *molecular weight* of this component.

Pseudo component
A *pseudo component* is an auxiliary concept, that defines a virtual chemical species with the averaged properties of a number of *chemical species*. *Pseudo components* are often assumed to exist in the context of physical property calculations of complex multicomponent mixtures, such as petroleum (Hariu and Sage 1969), fatty alcohols (Gutsche 1986), or polymers (Kuma and Gupta 1998).

Substance
Substance represents matter at or above the atomic level. It reflects the intrinsic, context-independent characteristics of a *material.*

Triple point pressure
The triple point of a *chemical component* is given by the temperature and pressure at which three phases (gas, liquid, and solid) of that *substance* coexist in thermodynamic equilibrium.

Triple point temperature
The triple point of a *chemical component* is given by the temperature and pressure at which three phases (gas, liquid, and solid) of that *substance* coexist in thermodynamic equilibrium.

Relation Descriptions

hasMacroscopicAppearance
The relation hasMacroscopicAppearance points from a *molecular entity* (describing a pure substance from a molecular perspective) to the corresponding *chemical species* (describing the same substance from a macroscopic perspective).

hasMolecularStructure
The relation hasMolecularStructure points from a *chemical species* (describing a substance from a macroscopic perspective) to the corresponding *molecular entity* (describing the same pure substance from a molecular perspective).

Attribute Descriptions

CAS_RegistryNumber
A CAS_RegistryNumber is a uniqueSubstanceID issued by the Chemical Abstracts Service (CAS) a division of the American Chemical Society. A CAS_RegistryNumber can be assigned to *chemical species* and *molecular entities* as well as to some *mixtures*. A CAS_RegistryNumber includes up to 9 digits, which are separated into 3 groups by hyphens (xxxxxx-xx-x). The first part of the number, starting from the left, has up to 6 digits; the second part has 2 digits. The final part consists of a single check digit or checksum that makes it easy to determine whether a CAS number is valid or not. See CAS (2007) for details.

canonicalStructuralFormula
A canonicalStructuralFomula is a structuralFormula that is generated by means of canonicalization algorithms to obtain a unique representation of a *molecular entity*.

canonicalSMILES
canonicalSMILES is the version of the SMILES specification that applies canonicalization rules to ensure that each *chemical species* and/or *molecular entity* has a single, unique SMILES representation.

chemicalFormula
A chemicalFormula is a substanceID that can only be assigned to *chemical species* and/or *molecular entities*. It gives information about the atoms that constitute a particular *molecular entity*. The attribute chemicalFormula subsumes all types of formulas, such as empiricalFormula, molecularFormula, structuralFormula, etc.

empiricalFormula
An empiricalFormula is a chemicalFormula that indicates the relative number of each constituting chemical element of a *molecular entity*.
In an empiricalFormula, the letters representing the chemical elements are listed according to the following convention: In organic compounds, C is always cited first, H second and then the rest, in alphabetical order. In non-carbon-containing compounds, strict alphabetical order is adhered to.

InChI
The IUPAC International Chemical Identifier (InChI) is a non-proprietary identifier for chemical substances that can be used in printed and electronic data sources thus enabling easier linking of diverse data and information compilations (Stein et al. 2003). InChI does not require the establishment of a registry system. Unlike the CAS Registry System, it does not depend on the existence of a database of unique substance records to establish the next number for any new *molecular entity* being assigned an **InChI**. It uses a set of IUPAC structure conventions, and rules for normalization and canonicalization of the structure representation to establish the unique label for a *molecular entity*.

isomericSMILES
isomericSMILES is the version of the SMILES specification that includes extensions to support the specification of isotopes, chirality, and configuration about double bonds.

isomericStructuralFormula
An isomericStructuralFormula is a structuralFormula that allows to distinguish the different isomers of a *molecular entity*.

molecularFormula
A molecularFormula is a chemicalFormula that specifies the (absolute) number of constituting atoms of a *molecular entity*, without indicating how they are linked.
In a molecularFormula, the letters representing the chemical elements are listed according to the following convention: In organic compounds, C is always cited first, H second and then the rest, in alphabetical order. In non-carbon-containing compounds, strict alphabetical order is adhered to.
For polymers and other macromelocules, parentheses are placed around the repeating unit. For example, a hydrocarbon molecule that is described as CH3(CH2)50CH3, is a molecule with 50 repeating CH2 units. If the number of repeating units is unknown or variable, the letter n may be used to indicate this (e.g. CH3(CH2)nCH3).
For ions, the charge on a particular atom may be denoted with a right-hand "+" or "-", e.g., "Na+" or "Cu,2+". The total charge on a charged molecule or a polyatomic ion may also be shown in this way, e.g., "H3O+" or "SO4,2-".

Name
The name attribute holds the various names of a *substance*. Both trivial and systematic names can be indicated here.

SMILES
SMILES (Simplified Molecular Input Line Entry System) is a line notation for unambiguously describing the structure of chemical molecules using ASCII strings (Weininger 1988).

structuralFormula
A structuralFormula is a chemicalFormula that supplies information about the types of bonds and the spatial arrangement of the atoms of a *molecular entity* using a linear string notation.

substanceID
A substanceID is an identifier for a *substance*. The substanceID must be unambiguous but not necessarily unique.

uniqueSubstanceID
A uniqueSubstanceID is a unique identifier for a *substance*. The different isomers of a *substance* are not necessarily distinguished by a uniqueSubstanceID.

WLN

The Wiswesser line notation (WLN), also known as Wiswesser line formula, is a precise and concise means of expressing structural formulas as character strings. The basic idea is to use letter symbols to denote functional groups and numbers to express the lengths of chains and the sizes of rings (Smith 1968).

7.2.2 Molecular Structure

The ontology module *molecular_structure* characterizes the molecular structure of pure *substances*. It refines the definition of *molecular entity*. To this end, the ontology module adopts some ontological concepts from the ChEBI[80] Ontology (EBI 2007; OLS 2006), particularly of its sub-ontology *Molecular Structure*. The ChEBI Ontology is a non-proprietary ontology concerned with the classification of "small" chemical compounds. It consists of four sub-ontologies; the sub-ontology *Molecular Structure* provides (structural) descriptions of molecular entities and of parts thereof, based on the composition and/or the connectivity between the constituting atoms. The terminology and nomenclature of *Molecular Structure* is based on the 'IUPAC Compendium of Chemical Terminology' (McNaught and Wilkinson 1997), informally known as the 'Gold Book'. The compendium extracts the definitions of terms described in the various IUPAC glossaries and other IUPAC nomenclature documents, including 'Glossary of Terms Used in Physical Organic Chemistry' (Müller 1994), 'Glossary of Class Names of Organic Compounds and Reactive Intermediates Based on Structure' (Moss et al. 1995), and 'Basic Terminology of Stereochemistry' (Moss 1996).
The class *ion* is introduced as a first refinement of *molecular entity*. An *ion* is defined as a *molecular entity* that has an *ionic charge* (cf. Fig. 7.11). *Ions* that have a *positive ionic charge* are classified as *cations*, those with a *negative ionic charge* are classified as *anions*.

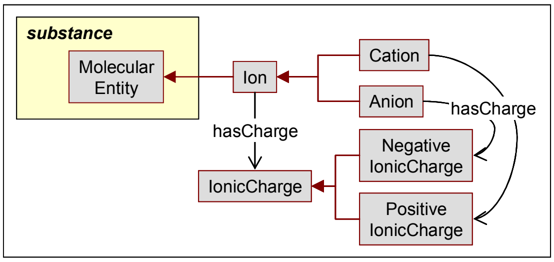

Fig. 7.11: Representation of ions

[80] ChEBI: Chemical Entities of Biological Interest

Fig. 7.12 presents a further refinement of the class *molecular entity*; the refinement is based on two different criteria:

- The differentiation *monoatomic* vs. *polyatomic entity* considers the number of atoms a *molecular entity* consists of. For example, the Na+ ion is a *monoatomic entity*, while the N2 molecule is a *polyatomic entity*.
- The differentiation *homoatomic* vs. *heteroatomic molecular entity* considers the number of elements a molecular entity consists of. The N2 molecule, for instance, is a *homoatomic molecular entity*, as it consists of a single element.

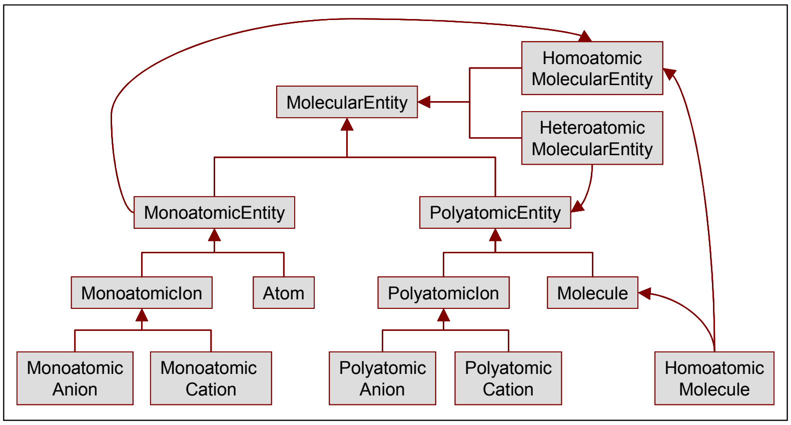

Fig. 7.12: Subclasses of *molecular entity*

Consequently, each *monoatomic entity* is a *homoatomic molecular entity*, and each *heteroatomic molecular entity* is a *polyatomic entity*.
Monoatomic ion and *polyatomic ion* (and their respective subclasses) are defined analogously to *ion*.

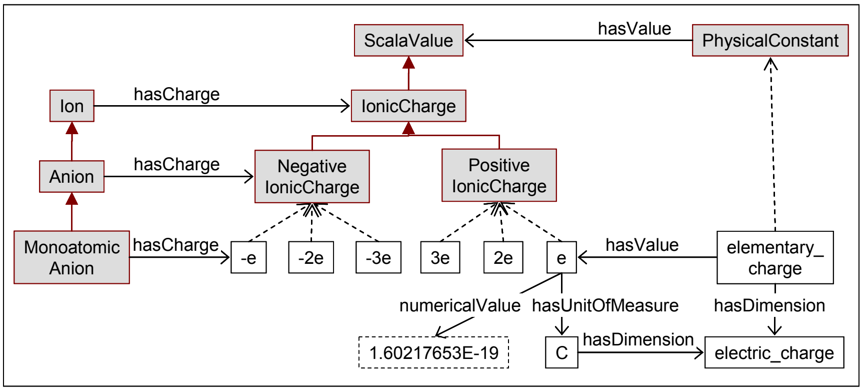

Fig. 7.13: Representation of *ionic charge*

The charge of an *ion* is represented by one of the following instances of *ionic charge*: **e**, **2e**, **3e**, **-e**, **-2e**, or **-3e** (cf. Fig. 7.13). The individual **e** represents the value of the **elementary_charge**, a *physical constant* that denotes the electric charge of a single *monoatomic ion* (1.60217653E-19 C). The other *ionic charges* are (positive or negative) integer multiples of **e**.
Fig. 7.14 presents concepts that can be used to describe the constitution of *molecular entities*: A *molecular entity* may contain a *molecular group*, which denotes either a linked collection of *atoms* or a single *atom* within a *molecular entity*. As each *molecular group* is a *molecular entity*, it may again contain some *molecular* sub-*group*. A restriction is placed on the *atom* class such that it cannot contain a *molecular group*.

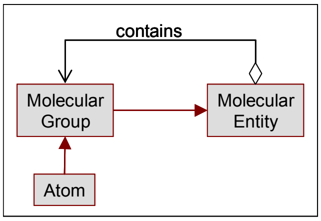

Fig. 7.14: Composition of *Molecular Entities*

7.2.2.1 Usage

The module *molecular_structure* is extended by the ontology modules *polymers*, *atoms*, and *substance_class*. The usage of the concepts of *molecular_structure* is explained in the specifications of these modules (cf. Secs. 7.2.3, 7.2.5, and 7.2.7).

7.2.2.2 Concept Descriptions

Individual concepts of the module *molecular_structure* are defined below.

Class Descriptions

Anion
An *anion* is a monoatomic or polyatomic species having one or more elementary charges of the electron (McNaught and Wilkinson 1997).
Formal definition: An *aninon* is an *ion* that has a *negative ionic charge*.

Atom
An *atom* is the smallest particle still characterizing a chemical element (McNaught and Wilkinson 1997).

Cation
A *cation* is a monoatomic or polyatomic species having one or more elementary charges of the proton (McNaught and Wilkinson 1997).
Formal definition: A *cation* is an *ion* that has a *positive ionic charge*.

Heteroatomic molecular entity
A *heteroatomic molecular entity* is a *molecular entity* consisting of two or more [distinct] chemical elements (OLS 2006).

Homoatomic molecular entity
A *homoatomic molecular entity* is a *molecular entity* consisting of one or more *atoms* of the same element (OLS 2006).

Homoatomic molecule
A *homoatomic molecule* is a *molecule* consisting of *atoms* of the same element (OLS 2006).

Ion
An *ion* is an atomic or molecular particle having a net electric charge (McNaught and Wilkinson 1997).
Formal definition: An *ion* is a *molecular entity* that has some *ionic charge*.

Ionic charge
Ionic charge is a *scalar value* that represents the electric charge of an *ion*, which is either a *positive ionic charge* or a *negative ionic charge*.

Molecular group
A linked collection of atoms or a single atom within a *molecular entity*.

Molecule
A *molecule* is an electrically neutral entity consisting of more than one *atom* (McNaught and Wilkinson 1997).

Monoatomic anion
A *monoatomic anion* is an *anion* consisting of a single *atom*.
Formal definition: A *monoatomic anion* is a *monoatomic ion* that has a *negative ionic charge*.

Monoatomic cation
A *monoatomic cation* is an *cation* consisting of a single *atom*.
Formal definition: A *monoatomic cation* is a *monoatomic entity* that has a *positive ionic charge*.

Monoatomic entity
A *monoatomic entity* is a *molecular entity* consisting of a single *atom*.
Formal definition: *Monoatomic entity* is either an *atom* or a *monoatomic ion*.

Monoatomic ion
A *monoatomic ion* is an *ion* consisting of a single *atom* (OLS 2006)
Formal definition: *Monoatomic ion* is a *monoatomic entity* that has an *ionic charge*.

Negative ionic charge
A *negative ionic charge* is a *scalar value* that represents the negative electric charge of an *ion*.

Polyatomic anion
A *polyatomic anion* is an *anion* consisting of more than one *atom* (OLS 2006).
Formal definition: A *Polyatomic anion* is a *polyatomic entity* that has a *negative ionic charge*.

Polyatomic cation
A *polyatomic cation* is a *cation* consisting of more than one *atom* (OLS 2006).
Formal definition: A *polyatomic cation* is a *polyatomic entity* that has a *positive ionic charge*.

Polyatomic entity
Any *molecular entity* consisting of more than one *atom* is a *polyatomic entity* (OLS 2006).

Polyatomic ion
A *polyatomic ion* is an *ion* consisting of more than one *atom* (OLS 2006).
Formal definition: A *polyatomic ion* is an *polyatomic entity* that has an *ionic charge*.

Positive ionic charge
A *positive ionic charge* is a *scalar value* that represents the positive electric charge of an *ion*.

Relation Descriptions

hasCharge
The relation hasCharge indicates the *ionic charge* of an *ion*.

Attribute Descriptions

atomicNumber
The atomicNumber (also known as the proton number) is the number of protons found in the nucleus of an *atom*.

7.2.3 Polymers

The ontology module *polymers* is concerned with the structural description of *macromolecules*. A *macromolecule* is a *molecule* of high *molecular weight*, the structure of which essentially comprises the multiple repetition of units derived, actually or conceptually, from *molecules* of low *molecular weight* (McNaught and Wilkinson 1997).

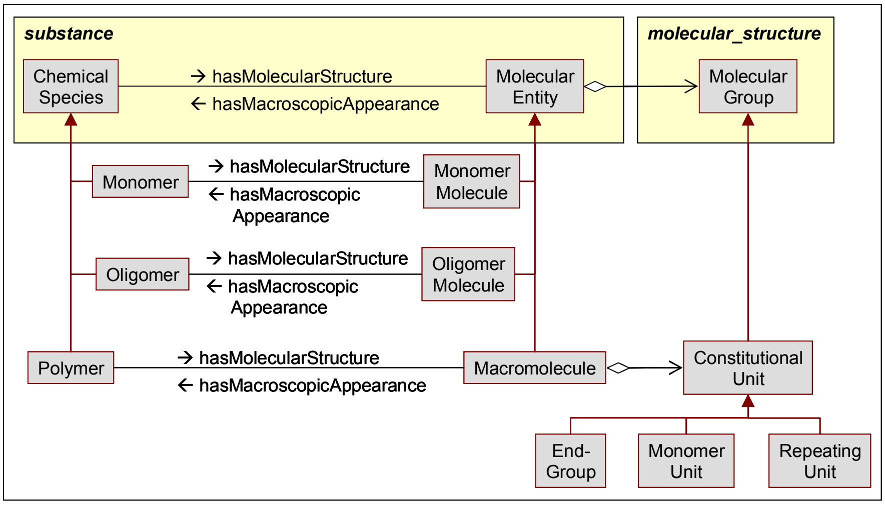

Fig. 7.15: Ontology module *polymers*

Industrially, *macromolecules* are synthesized from *monomer molecules* which undergo polymerization (cf. Fig. 7.15). Besides the *macromolecules*, *oligomer molecules* are often formed as intermediates or by-products of the polymerization reaction. *Oligomers* are *molecules* of intermediate *molecular weight*, the structure of which essentially comprises a small plurality of units derived, actually or conceptually, from *molecules* of lower *molecular weight* (McNaught and Wilkinson 1997).
The structure of a *macromolecule* can be described by indicating its building blocks, the so-called *constitutional units*. Three types of *constitutional units* are distinguished:

– *Monomer units* are *constitutional units* resulting from a *monomer molecule* that has been polymerized.
– *Repeating units* are the shortest *constitutional units* that can be found repeatedly in a *macromolecule*.
– Finally, *end-groups* are *constitutional units* that form the extremities of a *macromolecule* or *oligomer molecule*.

Monomers are *chemical species* composed of *monomer molecules*, and *oligomers* are *chemical species* composed of *oligomer molecules*. Similarly, *polymers*, such as polyethylene or polyamide, are *chemical species* that are composed of *macromolecules*. Note, however, that a *polymer* consists of a statistical distribution of different *macromolecules*, which vary with respect to chain length, side branches, cross-linkage, etc. Thus, an instance of *polymer* represents one specific configuration of *macromolecules*.

7.2.3.1 Usage

Depending on the application, an instance of *macromolecule* can represent either one specific molecule or a group of molecules of the same type, yet with different molecular structures and chain lengths. In the latter case, the chemicalFormula of the *macromolecule* is indicated in a generic form, with a variable number of repeating units (e.g., CH_3-(C2H4)$_n$-CH_3 for the polyethylene molecule shown in Fig. 7.16).

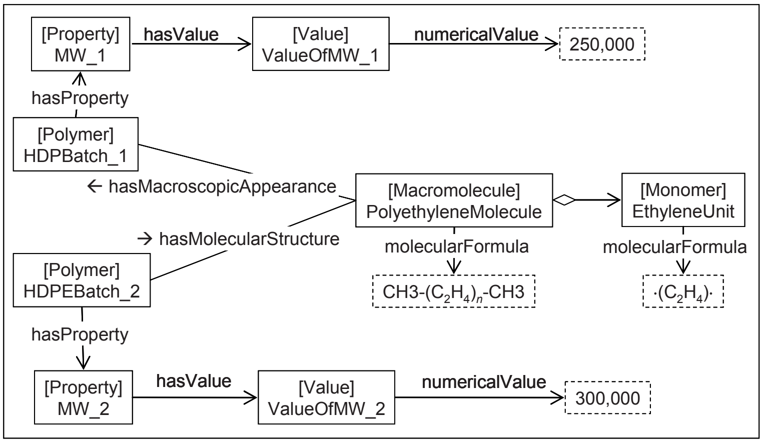

Fig. 7.16: Two batches of HDPE with different *molecular weights*

Like any *chemical species*, a *polymer* has *chemical component constants*, such as the *molecular weight*. Since each instance of *polymer* represents one individual configuration of *macromolecules*, the *values* of its *chemical component constants* are valid for this specific configuration only. Fig. 7.16 shows exemplarily two different *polymers* (**HDPE_batch_1** and **HDPE_batch_2**) that have different *molecular weights* (**MW_1** and **MW_2**). Both *polymers* have the same (generic) molecular structure, which is represented by an instance of *macromolecule* (**PolyethyleneMolecule**) and further characterized through indication of its constituting *monomer unit* (**EthyleneUnit**).

Please note that polymer materials may also be modeled as *mixtures* consisting of a number of *pseudo components* in the substance module (cf. Sect. 7.2.1). However, this kind of representation is not covered by this ontology module.

7.2.3.2 Concept Descriptions

Individual concepts of the module *polymers* are defined below.

Class Descriptions

Constitutional unit
A *constitutional unit* is a *molecular group* that constitutes a characteristic part of a *macromolecule*.

End-group
An *end-group* is a *constitutional unit* that is an extremity of a *macromolecule* or *oligomer molecule* (McNaught and Wilkinson 1997).

Macromolecule
A *molecule* of high *molecular weight*, the structure of which essentially comprises the multiple repetition of units derived, actually or conceptually, from *molecules* of low *molecular weight* (McNaught and Wilkinson 1997).

Monomer
A *substance* composed of *monomer molecules* (McNaught and Wilkinson 1997).

Monomer molecule
A *molecule* which can undergo polymerization thereby contributing *constitutional units* to the essential structure of a *macromolecule* (McNaught and Wilkinson 1997).

Monomer unit
The largest *constitutional unit* contributed by a single *monomer molecule* to the structure of a *macromolecule* or *oligomer molecule*.
Note: The largest constitutional unit contributed by a single monomer molecule to the structure of a *macromolecule* or *oligomer molecule* may be described as either monomeric, or by monomer used adjectivally.

Oligomer
A *substance* composed of *oligomer molecules* (McNaught and Wilkinson 1997).

Oligomer molecule
A *molecule* of intermediate *molecular weight*, the structure of which essentially comprises a small plurality of units derived, actually or conceptually, from *molecules* of lower *molecular weight*. A *molecule* is regarded as having an intermediate *molecular weight* if it has properties which do vary significantly with the removal of one or a few of the units. (McNaught and Wilkinson 1997).

Polymer
A *substance* composed of *macromolecules* (McNaught and Wilkinson 1997).

Repeating unit
A *repeating unit* is the shortest *constitutional unit* that can be found repeatedly in a *macromolecule*.

7.2.4 Chemical Species

The ontology module *chemical_species* is an application-oriented extension of the ontology module *substance*. It contains several thousand instances of *chemical species*, their substanceIdentifiers, chemicalFormulas and *molecular weights*. The data stems from the NIST Chemistry WebBook (Linstrom and Mallard 2005), a non-proprietary online database providing physical property data of pure substances.
The following entries of the NIST Chemistry WebBook are available in the *chemical_species* module (cf. Fig. 7.17):

– Species name: used as identifiers[81] for the instances of *chemical species*.
– Other names (synonyms) for the species: represented as values of the name attribute.
– Chemical formula: represented as values of the attribute molecularFormula.
– CAS Registry Number (if available): represented as values of CAS_RegistryNumber.
– IUPAC International Chemical Identifier (if available): represented as values of the InChI attribute.
– Molecular weight: represented as a *scalar value* of *molecular weight*.

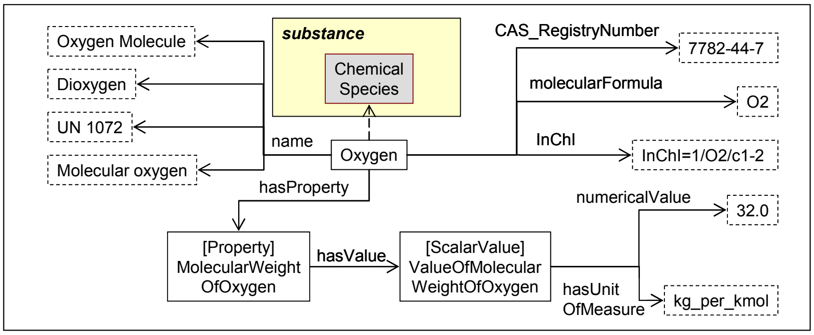

Fig. 7.17: Ontological representation of **Oxygen**

[81] In order to comply with the OWL syntax, certain characters in the species names had to be replaced by character symbols. Fig. 7.17 lists the transformation rules that were applied for this task.

7.2.4.1 Usage

The ontology module *chemical_species* contains instance data of about approximately 25,000 *chemical species*. Most of the current ontology management systems and reasoners cannot cope with such a large amount of information. For that reason, the OWL file *chemical_species.owl* provides a reduced version of *chemical_*species, comprising only ca. 2,000 selected *chemical species*. The full-fledged version of the ontology module is available as the OWL file *chemical_species_all.owl*.

7.2.5 Atoms

The ontology module *atoms* is an application-oriented extension of the ontology module *molecular_structure*. It represents the chemical elements as instances of the *atom* class. Each *atom* refers to the corresponding instance of *chemical species* (represented in the ontology module *chemical_species*) via the relation hasMacroscopicAppearance. An example is given in Fig. 7.18.
The same instance identifiers are used for *chemical species* and corresponding *atoms*. This is done since, in practice, the distinction between *molecular entities* and *chemical species* is often neglected. Thus, the name of a compound may refer to the respective *molecular entity* or to the corresponding *chemical species* (e.g., **Aluminum** may mean a single atom of Al or a macroscopic amount of Al). By means of the namespace prefix, the instances can be kept apart (e.g., **chemical_species:Aluminum** vs. **atoms:Aluminum**).

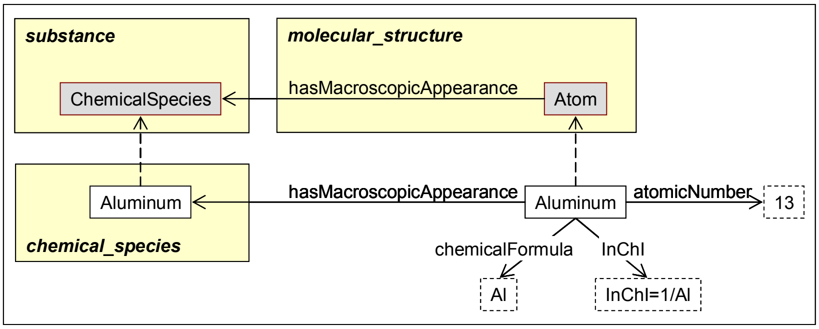

Fig. 7.18: Ontological representation of the **Aluminum** *atom*

Each *atom* is characterized by its atomicNumber. Additionally, the InChI and molecularFormula of the *atom* are indicated (this would not be necessary since the InChI and molecularFormula have already been specified in the *chemical_species* module, but for reasons of user convenience the information is duplicated here).

7.2.6 Macromolecules

The ontology module *macromolecules* provides a few examples (i.e., individuals) of *macromolecules* in order to demonstrate the usage of the module *polymers* (cf. 7.2.3). For the description of the respective individuals, please refer to the formal specification.

7.2.7 Substance Class

The ontology module *substance_class* is an application-oriented extension of the **substance** partial model; it is concerned with the classification of pure *substances* and *molecular entities*. Of course, there are different ways to classify substances, and each has its individual assets and drawbacks. The *substance_class* module merely suggests one possible approach to substance classification, which may or may not be applicable to a particular problem at hand.
The classification criterion that is applied here is *chemical similarity* (Schummer 1998): *Molecular entities* are categorized into *substance classes* like *alcohols*, *carboxylic acids*, *esters*, *aldehydes*, *amines*, etc. Note that a *molecular entity* may be assigned to multiple substance classes – for example, it may belong to the class of *acids* as well as to the class of *aromatics*.
Molecular entities are firstly classified into *organic compounds* and *inorganic compounds* (Fig. 7.19). Below, the various substance classes are defined, which may or may not have further subclasses of their own.

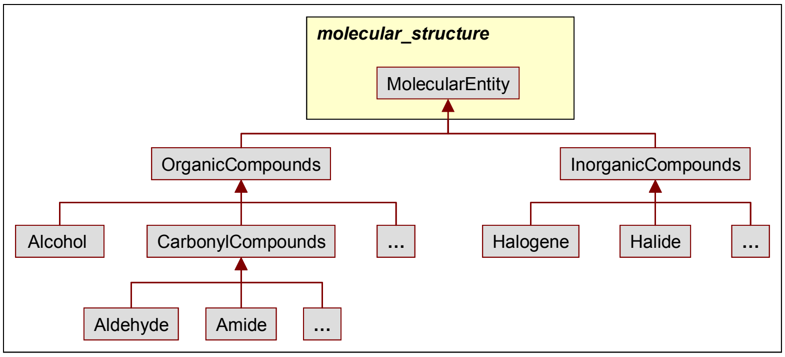

Fig. 7.19: Some exemplary substance classes

Each substance class has one characteristic *functional group*, as indicated in Fig. 7.20. A *functional group* is a *molecular group* that has “similar chemical properties whenever it occurs in different compounds. It defines the characteristic physical and chemical properties of families of organic compounds” (McNaught and

Wilkinson 1997). *Molecular entities* that belong to the same substance class contain the same *functional group*. For example, all *alcohols* contain a **HydroxylGroup**.

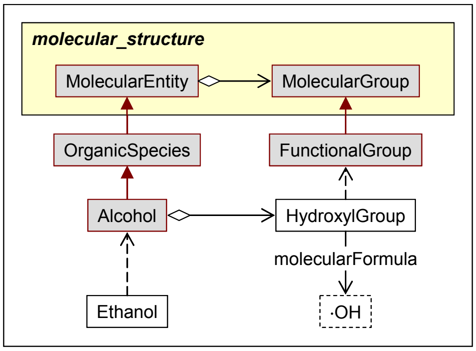

Fig. 7.20: *Functional groups*

Only a limited number of substance classes have been defined, so far. In a later version of OntoCAPE, it is planned to integrate further parts of the ChEBI ontology (EBI 2007), which gives an extensive classification of *molecular entities*.

7.2.7.1 Concept Descriptions

Individual concept of the module *substance_class* is defined below.

Concept Definitions

Functional group
"Organic compounds are thought of as consisting of a relatively unreactive backbone, for example a chain of hybridized carbon atoms, and one or several functional groups. The functional group is an atom, or a group of atoms that has similar chemical properties whenever it occurs in different compounds. It defines the characteristic physical and chemical properties of families of organic compounds" (McNaught and Wilkinson 1997).
For a description of the individual substance classes, refer to the OWL implementation of the ontology module *substance_class*.

7.2.8 Reaction Mechanism

The ontology module *reaction_mechanism* allows the representation of the mechanism, i.e., the stoichiometry of chemical reactions, as shown in Fig. 7.21.

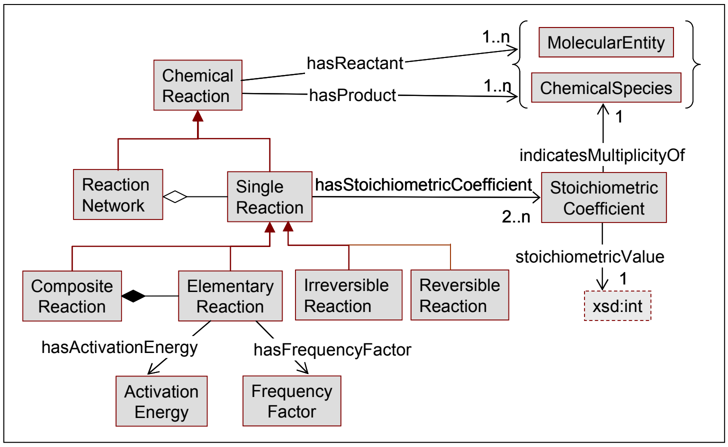

Fig. 7.21: Ontology module *reaction_mechanism*

Chemical reaction is the key concept of this module. In OntoCAPE, a *chemical reaction* is modeled as a subclass of *system* (cf. Sect. 5.1) and is regarded generally as a mechanism that produces chemical products by consuming some reactants.
A *chemical reaction* may be a *reaction network* or a *single reaction*. A *reaction network* represents a loose collection of a number of *single reactions*, usually for indicating their co-existence in a specific circumstance. In particular, we consider a collection of *single reactions* in a *reaction network* if we want to explicitly refer to each of the *single reactions* of a more complex reaction mechanism involving many parallel and consecutive reactions. A *single reaction* in turn may be a *composite reaction* or an *elementary reaction*. An *elementary reaction* refers to a molecular transformation for which no molecular intermediates have been detected or need to be postulated for a proper description. In contrast, a *composite reaction* refers to molecular transformations involving a number of reaction steps and intermediates with a known mechanism the details of which are however not of interest on the granularity of the reactor hosting the reaction. In other words, *composite reactions* refer to a net reaction of a more complex *reaction network* which is not described in detail in a certain context. If a detailed description is desired in addition, a *reaction network* could be formulated and associated with the *composite reaction*. *Single reactions* can also be classified into *irreversible reactions* and *reversible reactions*.
The reactants and products of the *chemical reaction* are identified via the relations hasReactant and hasProduct, respectively. The reactants and products can be either *molecular entities* or *chemical species*, depending on whether one wants to describe the reaction on the molecular level (conversion of some *molecular entities* into some other *molecular entities*) or on the macroscopic level (conversion of *chemical species*). The stoichiometry of a *single reaction* is modeled by means of *stoichiometric coefficients*, each of which indicates the multiplicity of the respective reactant or product. The numerical value of a *stoichiometric coefficient* is specified by the attribute stoichiometricValue.

A *single reaction* is a *chemical reaction* that has a fixed stoichiometry which does not change with reaction conditions or extent of reaction (Levenspiel 1999). The stoichiometry of a *reaction network* is given and fixed by a stoichiometric matrix. Such a matrix can be assembled implicitly from the *stoichiometric coefficients* of the individual (elementary or composite) reactions. Hence, this matrix may be either the true one (as it would be the case if there were only elementary reactions) or the one which is the result of an aggregation process of some *single reactions* into a *composite reaction* which is part of a *reaction network*. In case a *reaction network* is represented by a *single reaction* only, the apparent stoichiometry corresponding to this *single reaction* may change with the environment in which these reactions occur.
The reactants and products of the *chemical reaction* are identified via the relations hasReactant and hasProduct, respectively. The reactants and products can be either *molecular entities* or *chemical species*, depending on whether one wants to describe the reaction on the molecular level (conversion of some *molecular entities* into some other *molecular entities*) or on the macroscopic level (conversion of *chemical species*). The stoichiometry of a *single reaction* is modeled by means of *stoichiometric coefficients*, each of which indicates the multiplicity of the respective reactant or product. The numerical value of a *stoichiometric coefficient* is specified by the attribute stoichiometricValue.
The *reaction_mechanism* module further introduces two constant *properties* for a *elementary reaction* in connection with the application of the Arrhenius Equation to depict the temperature-dependency of the reaction rate. These two properties are *activation energy* and *frequency factor*. Note that these two concepts are introduced only to address the situations where the activity energy is regarded as independent of temperature, i.e. where the linear Arrhenius behaviour is exhibited. Chemical kinetics of more complex situations will be directly and modelled through proper kinetics *laws* as part of the *mathematical models* to be covered by Sect 9.6.
Note that the *reaction_mechanism* module only captures chemical reaction related properties which are independent of the physical context. Properties that change with the physical context are introduced later either in the **phase_system** (cf. Sect. 7.3.4) or the **CPS_behavior** (cf. Sect. 8.6.1.7) partial model, depending on whether a property is dedicated to equilibrium reactions or not.

7.2.8.1 Usage

The usage of the *reaction_mechanism* module is demonstrated by the following example shown in Fig. 7.22: Consider the overall (composite) reaction

$$2\ H_2 + 2\ NO \rightarrow N_2 + 2\ H_2O$$

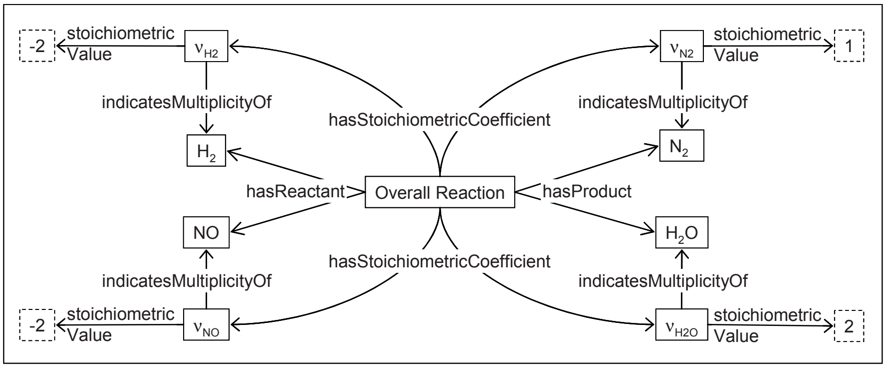

Fig. 7.22: Stoichiometry of the reaction **2 NO → N_2O_2**

The reaction mechanism of the overall reaction is given by the following *elementary reactions*:

$$2\,NO \rightarrow N_2O_2$$
$$H_2 + N_2O_2 \rightarrow N_2O + H_2O$$
$$H_2 + N_2O \rightarrow N_2 + H_2O$$

In Fig. 7.23, the **Overall Reaction** is decomposed into the three *elementary reactions*. The reactants and products of the reactions are shown, as well. The *stoichiometric coefficients* are not represented for reasons of clarity.

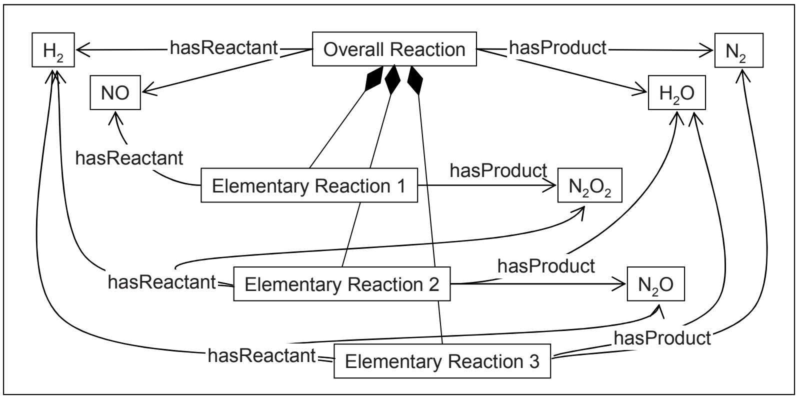

Fig. 7.23: The **Overall Reaction** is composed of three *elementary reactions*

7.2.8.2 Concept Descriptions

Individual concepts of the module *reaction_mechanism* are defined below.

Class Descriptions

Activation energy
Activation energy is a constant *property* as involved in the Arrhenius Equation for depicting the temperature-dependency of reaction rates.

Chemical reaction
A *chemical reaction* converts some *chemical species* (or *molecular entities*) into some other *chemical species* (or *molecular entities*).

Composite reaction
A *composite reaction* is a *chemical reaction* that could in principle be decomposed into several *elementary reactions* to reflect the known mechanism of a complex molecular transformation with the molecular intermediates. The composite reaction does not strive for a mechanistic representation of the molecular transformation but only for a coarse description of the net reaction typically from the perspective of the chemical reactor hosting the *composite reaction.* The stoichiometry of a *composite reaction* may change with its environmental conditions.

Elementary reaction
An *elementary reaction* is a *chemical reaction* for which no reaction intermediates have been detected or need to be postulated in order to describe the *chemical reaction* on a molecular scale. An *elementary reaction* is assumed to occur in a single step and to pass through a single transition state (McNaught and Wilkinson 1997).

Frequency factor
Frequency factor is a constant *property* as involved in the Arrhenius Equation for depicting the temperature-dependency of reaction rates.

Irreversible reaction
An *irreversible reaction* is *chemical reaction* that converts reactants to products which cannot be readily reversed to restore the reactants to its original state.

Reaction network
A *reaction network* refers to a set of *chemical reactions* that represent a loose collection of a number of *single reactions*, usually for indicating their co-existence in a specific circumstance. The stoichiometry of a *reaction network* is given and fixed by a stoichiometric matrix which can be assembled from the *stoichiometric coefficients* of the individual *elementary reactions* or *composite reactions.*

Reversible reaction
A *reversible reaction* is a *chemical reaction* that proceeds in both directions at the same time, as the product decomposes back into reactants as it is being produced.

Single reaction
A *single reaction* is a *chemical reaction* that has a fixed stoichiometry which does not change with reaction conditions or extent of reaction (Levenspiel 1999).

Stoichiometric coefficient
A *stoichiometric coefficient* indicates the multiplicity of a *chemical species* or *molecular entity* that participates in a *chemical reaction*.

Relation Descriptions

hasActivationEnergy
The relation hasActivationEnergy identifies the *activation energy* of a *elementary reaction*.

hasFrequencyFactor
The relation hasFrequencyFactor identifies the *frequency factor* of a *elementary reaction*.

hasStoichiometricCoefficient
The relation hasStoichiometricCoefficient identifies the *stoichiometric coefficients* of a *chemical reaction*.

hasProduct
The relation hasProduct denotes the products of a *chemical reaction*. A product is a *molecular entity* or a *chemical species* that is formed during a *chemical reaction*.

hasReactant
The relation hasReactant denotes the reactants of a *chemical reaction*. A reactant is a *molecular entity* or a *chemical species* that is consumed in the course of a *chemical reaction*.

indicatesMultiplicityOf
The relation indicatesMultiplicityOf indicates the multiplicity of the reactants and products participating in a *chemical reaction*.

Attribute Descriptions

stoichiometricValue
The attribute stoichiometricValue specifies the numerical value of a *stoichiometric coefficient*. It is positive for products and negative for reactants.

7.2.9 Reaction Type

The *reaction_type* module is an application-oriented extension of the ontology module *reaction_mechanism*. It describes some important types of *chemical reactions*, such as *esterification* or *hydrohalogenation*. Fig. 7.24 shows exemplarily the definition of the class *transesterification*: *transesterification* is a *chemical reaction* that has an *ester* and an *alcohol* as reactants and a different *ester* and a different *alcohol* as products (the classes *ester* and *alcohol* are defined in the ontology module *substance_class*). It is a subclass of the more generic *esterification* reaction, which is defined as a reaction that has an *ester* as a product.

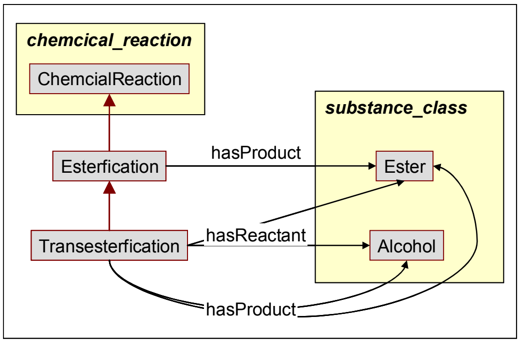

Fig. 7.24: Definition of the *transesterification* reaction

7.2.9.1 Concept Descriptions

For the individual concepts of the module we refer to the OWL implementation due to the high number of concepts.

7.3 Phase System

The partial model **phase_system** describes the macroscopic thermodynamic behavior of *material*, subject to a certain *physical context*. Following the conceptualization presented in the *upper_level* of OntoCAPE, the behavioral aspect of a *system* can be characterized quantitatively by *properties* as well as qualitatively by *phenomena*. In the case of *phase system*, which represents the thermodynamic behavior of *material*, one phenomenon, namely thermodynamic equilibrium, constantly exists according to the definition of *phase system*. There is no other phenomenon applicable to *phase system*. Therefore, there is no practical need to characterize a *phase system* from a phenomenon point of view. Consequently, a *phase system* is only characterized by means of *properties*.

7.3.1 High-Level Concepts

Phase system is the key concept of this partial model. It subsumes the classes *single phase* and *multiphase system* (cf. Fig. 7.25).

- A *single phase* represents a finite, homogeneous region of matter within which the values of its *physical quantities* are uniformly constant, i.e., they do not experience any change in passing from one point in the volume to another.
- A *multiphase system* is composed of several *single phases*; a *single phase* that is a subsystem of a *multiphase system* is characterized as a *single phase in multiphase system*.

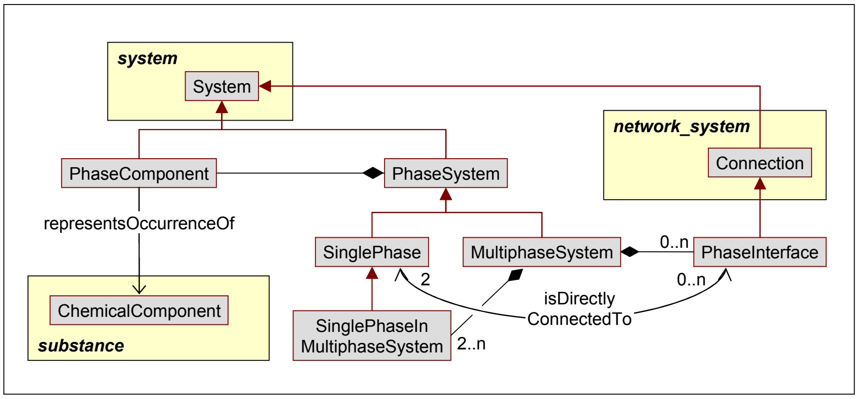

Fig. 7.25: Major concepts of partial model **phase_system**

Optionally, the interface between two adjacent *single phases* can be indicated through the class *phase interface*. A *phase interface* refers to the *single phases* via the relation isDirectlyConnectedTo. If a *phase interface* connects two instances of *single phase in multiphase system*, it is a subsystem of the overall *multiphase system*. Moreover, the connectivity rules stated in the ontology module *network_system* (cf. Sect. 5.2) must be obeyed.

Finally, the class *phase component* represents the occurrence of a *chemical component* in a *phase system*. The relation representsOccurenceOf establishes the relationship between a *phase component* and the corresponding *chemical component*.

The *state of aggregation* (a.k.a. state of matter) is a characteristic of *a single phase* that describes its physical state (cf. Fig. 7.26). Well-known examples of *states of aggregation* are solids, liquids, and gases. There are other, less familiar states, such as plasma or Bose-Einstein condensates.

State of aggregation is linked to a *single phase* via the relation hasStateOfAggregation, a specialization of hasCharacteristic. *State of aggregation* is a subclass of *extensible* *value set*; **solid**, **liquid**, and **gaseous** are predefined instances of *state of aggregation*. The solid state can be further characterized by choosing one of the following instances of *type of solid* instead of **solid**: **SolidSolution**, **HydrateI**, **HydrateII**, **HydrateH**, or **PureSolid** (Drewitz et al. 2006).

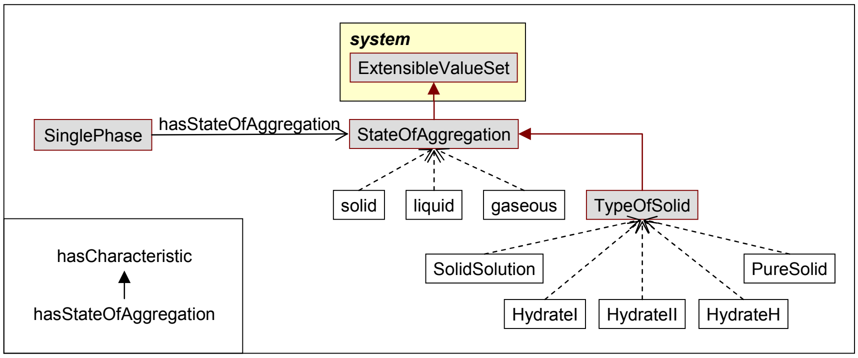

Fig. 7.26: State of aggregation

7.3.2 Properties of Phase Systems

As explained in Sect. 7.1.1, the scope of the properties defined within the partial model **phase_system** is confined to those which equilibrium thermodynamics is concerned with (e.g., Modell and Reid 1983); properties within this scope are traditionally called “physical properties”, including those of stable equilibrium thermodynamic states as well as the transport coefficients that correspond to a certain thermodynamic state (such as dynamic viscosity[82], thermal conductivity, or diffusion coefficients). A common feature of all these properties is that they have no coordinate in time or space and are independent of the geometry and amount of material. Properties that don’t belong to this category will be associated with *material amount* instead as defined in the *behavior* module in Sect. 8.6.1.
As shown in Fig. 7.27, four distinct types of *intensive properties* are defined in partial model **phase_system**: *phase system properties*, *phase interface properties*, *phase component properties*, and *phase reaction properties*.

[82] This is restricted to the dynamic viscosity of Newtonian fluids only. For a non-Newtonian fluid, this property is dependent of the gradient of velocity, thus going beyond the scope of equilibrium thermodynamics (i.e. that of phase system). Therefore, it should be regarded as a property of a *material amount* in that case.

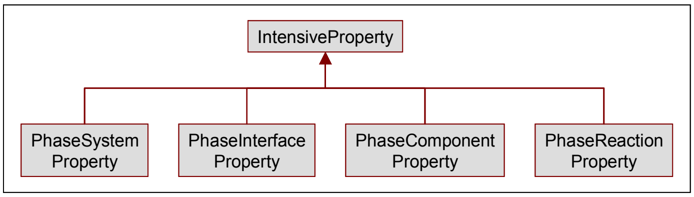

Fig. 7.27: High-level classification of the *intensive properties* defined in partial model **phase_system**

Fig. 7.28 gives an overview on the *phase system properties* currently defined in OntoCAPE. In accordance with the above discussion, *phase system properties* include (1) *thermodynamic state properties* such as *temperature*, *pressure*, *specific volume*, *density*, etc.; (2) *transport phenomena properties* such as *thermal conductivity*, *dynamic viscosity*, *and diffusion coefficient*; and (3) specific *properties* of *single phases in multiphase systems*, such as *phase ratios*. For detailed descriptions of these individual *phase system properties*, we refer the reader to the concept definition section below.

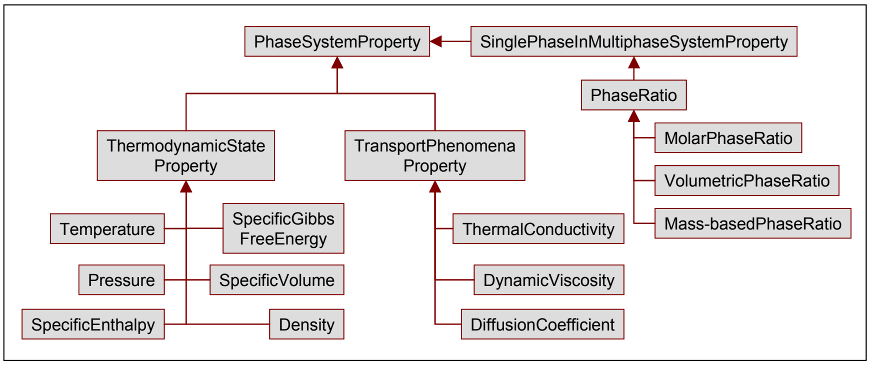

Fig. 7.28: Phase system properties

Properties associated with a *phase interface* are summarized as *phase interface properties*. Currently, OntoCAPE includes one single *phase interface property*, namely *surface tension* (cf. Fig. 7.29).

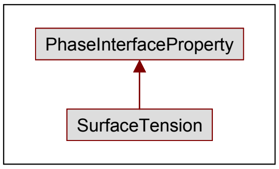

Fig. 7.29: Phase interface properties

A *phase component* reflects the behavior of a *chemical component* within a *phase system* by means of *phase component properties*, such as *activity coefficient*, *fugacity*, or *partial molar volume*. Thus, all *properties* of individual *chemical components* that appear within certain *physical contexts* are attributed to the *phase component* class. Fig. 7.30 shows the *phase component properties* that have been defined in OntoCAPE so far.

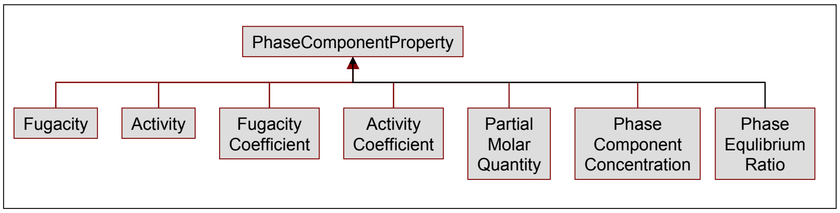

Fig. 7.30: Phase component properties

Note that *phase equilibrium ratio* (a.k.a. K value) is associated with a *phase component* and two *single phases* involved in a *multiphase system*. However, it is primarily regarded as a property of a *phase component* in this ontology, because it indicates a trait of a *phase component* in the context of phase equilibrium (e.g. the "lightness" in vapor-liquid equilibrium; cf. Smith and Van Ness 1975). Furthermore, this property is applicable only to a *phase component* which is associated with a *multiphase system*. Additionally this property refers to a first and a second *single phase* to articulate the nature of the equilibrium ratio. For details of other *phase component properties*, we refer to the individual concept definitions below.

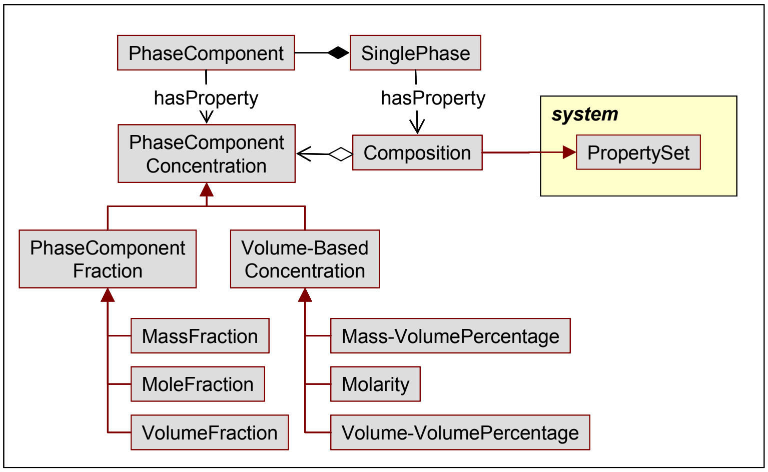

Fig. 7.31: Composition of a phase system

The concentration of a *phase component* within a *phase system* can be represented by different concepts (Fig. 7.31): A *phase component fraction* is a relative concentration, defined as the mass, molar amount, or volume of one *phase component* divided by the mass, molar amount, or volume of all *phase components* in that *phase system*. *Volume-based concentrations* are defined as the mass, molar amount, or volume of a *phase component* of a *phase system* divided by the total volume of that *phase system*. The class *phase component concentration* subsumes the different concentration measures for a *phase component*. The composition of a *phase system* is described through the *composition* concept. A *composition* is a *property set* that comprises the *phase component concentrations* of all *phase components* that constitute the *phase system* (cf. Fig. 7.31).
It is worth noting that, although the *composition* concept introduced above is generally applicable to both *single phases* and *multiphase systems*, in practice the *composition* of a *multiphase system* is usually depicted only by means of *mass fractions* or *mole fractions* of the constituting *phase components*.

7.3.3 Physical Context

A *phase system* is subject to a certain *physical context*. According to Yang et al. (2003), a *physical context* "stands for a set of *phase system properties* with known values in order to characterize a certain *phase system*. Different sets of *properties* might be selected here, but in any case the selected *properties* should be sufficient for determining other *properties* of interest of the *phase system*." In OntoCAPE, a *physical context* is modeled as a *property set* (cf. Fig. 7.32).
A *physical context* comprises a minimum set of *intensive thermodynamic state variables* required to fully characterize a *phase system* in thermodynamic equilibrium. Conventionally, a *physical context* is described by *temperature*, *pressure*, and *phase component fractions*, although other *intensive thermodynamic state variables* may be used alternatively.

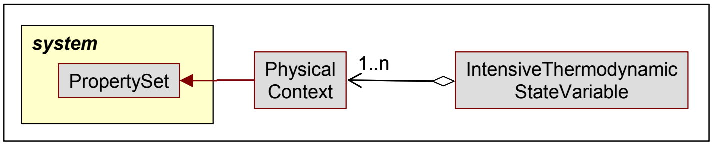

Fig. 7.32: Physical context

An *intensive thermodynamic state variable* (often simply called "state variable" in the literature) denotes any *intensive property* that can be used to characterize the macroscopic thermodynamic state of a *phase system* (cf. Fig. 7.33). It subsumes the classes *thermodynamic state property* and *phase component property*. *Temperature*, *pressure*, and *molar fractions* are the most commonly used *intensive thermodynamic state variables*.

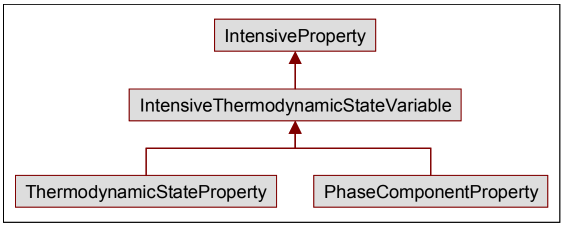
IntensiveProperty
IntensiveThermodynamicStateVariable
ThermodynamicStateProperty
PhaseComponentProperty

Fig. 7.33: Intensive thermodynamic state properties

The minimum number of *intensive thermodynamic state variables* that are required to determine the thermodynamic equilibrium state of a *phase system* is given by the Gibbs Phase Rule (e.g., Sandler 1999) and depends on the characteristic of the *phase system*. Moreover, not all *intensive thermodynamic state variables* are independent, as stated by the Gibbs-Duhem Equation (e.g., Sandler 1999). Thus, the combination of *intensive thermodynamic state variables* appearing in a *physical context* has to be chosen appropriately. In the case of a *multiphase system*, the *physical context* should be composed of the *intensive thermodynamic state variables* of the constituting *single phases*; by contrast, using the *intensive thermodynamic state variables* of the composite *multiphase system* is not recommended for practical use (Sandler 1999).

7.3.4 Reactions in Phase Systems

The occurance of a *chemical reaction* (defined in the module *reaction_mechanisms*; cf. Sect. 7.2.8) in a *phase system* or at a *phase interface* is represented by the hasReaction relation that links the latter two to the former, as shown in Fig. 7.34.

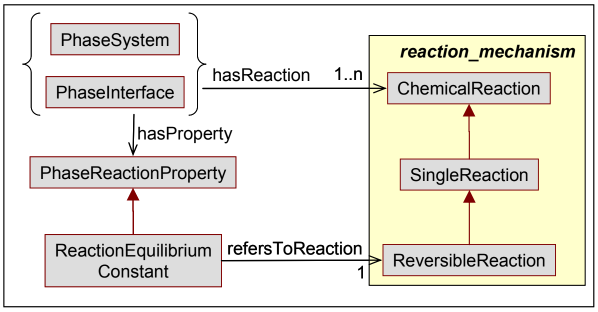

Fig. 7.34: Phase reactions

Furthermore, the **phase_system** partial model introduces a specific *phase reaction property*, namely *reaction equilibrium constant*. A *phase system* or *phase interface* may have one or more *reaction equilibrium constants*. Each instance of this property refers to a particular *reversible reaction* which again has been defined in the *reaction_mechanism* module.

7.3.5 Concept Descriptions

Individual concepts of the module *phase_system* are defined below.

Class Descriptions

Activity
The *activity* of some *phase component* is the ratio of the *phase component*'s *fugacity* to the *fugacity* in its standard state.

Activity coefficient
The *activity coefficient* of a *phase component* is the ratio of its *fugacity* in the actual *phase system*, to its *fugacity* in an ideal mixture.

Composition
Composition represents the composition of a *phase system* by assembling the concentrations of the different *phase components* that constitute the *phase system*.

Density
The mass of a *phase system* divided by its volume. It is the reciprocal of *specific volume*.

Diffusion coefficient
Proportionality constant, relating the flux of amount of some *phase component* to its concentration gradient.

Dynamic viscosity
For a laminar flow of a fluid, the ratio of the shear stress to the *velocity gradient* perpendicular to the plane of shear (McNaught and Wilkinson 1997).

Fugacity
The fugacity of a *phase component*.

Fugacity coefficient
Ratio of *fugacity* to the partial pressure of a *phase component*.

Intensive property
An *intensive property* is a *physical quantity*, the *value* of which does not depend on the system size or the amount of material in the system.

Intensive thermodynamic state variable
An *intensive thermodynamic state variable* is an *intensive property* that characterizes the (macroscopic) thermodynamic state of a *phase system*.
Formal definition: An *intensive thermodynamic state variable* is either a *thermodynamic state property* or a *phase component property*.

Mass-based phase ratio
The ratio of the mass of a *single phase in multiphase system* divided by the mass of the *multiphase system*.

Mass fraction
The mass of a *phase component* divided by the total mass of all *phase components* of a *phase system*.

Molarity
The *molarity* (a.k.a. amount concentration) is the molar amount of a *phase component* divided by the volume of the *phase system*.

Molar phase ratio
The ratio of the (molar) amount of substance of a *single phase in multiphase system* to the (molar) amount of substance of the *multiphase system*.

Mole fraction
The number of moles of a *phase component* divided by the total number of moles of all *phase components* in the *phase system*.

Multiphase system
A *multiphase system* is a *phase system* that consists of two or more *single phases*.

Partial density
Mass of a *phase component* divided by the volume of the *phase system*.

Partial molar enthalpy
The partial derivative of the *specific enthalpy* with respect to the number of moles of one *phase component*.

Partial molar quantity
A *partial molar quantity* is the partial derivative of the considered molar quantity with respect to the number of moles of one *phase component*. The temperature, pressure, and the number of moles of all other *phase components* are held constant when forming the derivative. The class subsumes all kinds of partial molar quantities, such as *partial molar enthalpy* and *partial molar volume*.

Partial molar volume
The partial derivative of the *specific volume* with respect to the number of moles of one *phase component*.

Phase component
A *phase component* represents the occurrence of a *chemical component* in a *phase system.*

Phase component concentration
Phase component concentration subsumes the different concentration measures for a certain *phase component* within a *phase system.*

Phase component fraction
In general, the ratio of two *physical quantities* of the same kind, the numerator quantity applying to one particular *phase component* of a *phase system*, and the denominator to the sum of quantities of all *phase components* of that *phase system*. More concretely, a *phase component fraction* subsumes the following concentration measures: *mass fraction*, *volume fraction*, and *mole fraction*.

Phase component property
A *phase component property* is an *intensive property* that characterizes a *phase component*.
Formal definition: A *phase component property* is an *intensive property* of some *phase component*.

Phase equilibrium ratio
A *phase equilibrium ratio* of a *phase component* in a *multiphase system* is a *phase component property* that denotes the ratio of the *mole fraction* of this *phase component* in one specific *single phase* to that in another specific *single phase*; both *single phases* are part of the *multiphase system*.

Phase interface
A *phase interface* represents the interface between two *single phases*.

Phase interface property
A *phase interface property* is an *intensive property* that characterizes the interface between two *single phases*.

Phase ratio
A *phase ratio* characterizes the proportion of a *single phase* in a *multiphase system* on a mass, molar, or volume basis.
Formal definition: A *phase ratio* is either a *mass-based phase ratio*, or a *volumetric phase ratio*, or a *molar phase ratio*.

Phase reaction property
A *phase reaction property* is an *intensive property* that is a property of the occurance of a *chemical reaction* in a *phase system* or at a *phase interface*.

Phase system
A *phase system* represents the macroscopic thermodynamic behavior of *material* in some *physical context*.
Formal definition: A *phase system* is either a *single phase* or a *multiphase system*.

Phase system property
A *phase system property* is an *intensive property* that characterizes a *phase system*. It can be described or calculated without reference to the shape, size, or amount of a particular occurrence of a material. In the case of calculation, this is consistent with the usage of general-purpose physical property packages, where such information is not required as the input for the calculation.
Formal definition: A *phase system property* is an *intensive property* of a *phase system*.

Physical context
A *physical context* of a *phase system* is a set of *intensive thermodynamic state properties* with known *values* which are sufficient for determining other *properties* of interest of the *phase system*. Conventionally, a *physical context* is described by *temperature*, *pressure*, and *molar fractions*, although other *intensive thermodynamic state variables*, such as *specific volume*, *specific enthalpy*, *specific Gibbs free energy*, or *volume-based concentrations* may be used alternatively.

Pressure
The (total absolute) pressure of a *phase system*.

Reaction equilibrium constant
According to the IUPAC Compendium (McNaught and Wilkinson 1997), the *reaction equilibrium constant* is a *physical quantity* characterizing the chemical equilibrium of a chemical reaction. It is defined by an expression of type $K = \prod_i x_i^{\nu_i}$,

where ν_i is the *stoichiometric coefficient* of a reactant (negative) or product (positive) of the reaction, and x_i stands for a *physical quantity* which can be the equilibrium value either of *pressure*, *fugacity*, *molarity*, *molar fraction*, molality, or *activity*. Depending on the chosen quantity, one obtains one of the following types of *reaction equilibrium constant*: pressure based, fugacity based, concentration based, amount fraction based, molality based, relative activity based, or standard equilibrium constant, respectively. These different types can be modeled as subclasses of the *reaction equilibrium constant*.

Single phase
A *single phase* is a finite volume of material within which the physical properties are uniformly constant, i.e., do not experience any abrupt change in passing from one point in the volume to another.

Single phase in multiphase system
A *single phase* that is part of a *mulitphase system*.
Formal definition: A *single phase* which is an exclusive subsystem of a *multiphase system*.

Single phase in multiphase system property
Class used for grouping the *properties* of a *single phase in multiphase system*.
Formal definition: The *property* of a *single phase in multiphase system*.

Specific enthalpy
Specific enthalpy denotes enthalpy per unit mass (i.e., the enthalpy of a *phase system* divided by its mass). It is defined by the equation $h = u + p$, where u represents the specific internal energy, p the *pressure*, and v the *specific volume* of the *phase system*.

Specific Gibbs free energy
Specific Gibbs free energy denotes the Gibbs free energy per unit mass (i.e., the Gibbs free energy of a *phase system* divided by its mass). It is defined by the equation g = u + p*v - T*s, where u represents the specific internal energy, p the *Pressure*, v the *specific volume*, T the *temperature*, and s the specific entropy of the *phase system*.

Specific volume
Specific volume is the volume of a *phase system* divided by its mass. It is the reciprocal of *density*.

State of aggregation
The *state of aggregation* (a.k.a. state of matter) describes the physical state of a *single phase*; **solid**, **liquid**, and **gaseous** are predefined instances of *state of aggregation*.

Surface tension
Work required to increase a surface area divided by that area. When two phases are studied it is often called interfacial tension (McNaught and Wilkinson 1997).

Temperature
The temperature of a *phase system*.

Thermal conductivity
The *thermal conductivity* λ is the coefficient relating the heat flux q to the temperature gradient ∇T: $q = -\lambda \nabla T$.

Thermodynamic state property
A *thermodynamic state property* is a *phase system property* that can serve as an *intensive thermodynamic state variable* (i.e., characterize the thermodynamic state of a *phase system*).

Transport phenomena property
A *transport phenomena property* is a *phase system property* that subsumes *dynamic viscosity, heat conductivity*, and *diffusion coefficient*.

Type of solid
Type of solid allows to further characterize the solid state of matter, according to the CAPE-OPEN Open Interface Specification "Thermodynamic and Physical properties" (Drewitz et al. 2006).
Formal definition: T*ype of solid* is an enumerated class that can take one of the following instance values: **HydrateI**, **HydrateII**, **HydrateH**, **PureSolid**, **SolidSolution**.

Volume-based concentration
Volume-based concentration denotes the concentration of a certain *phase component* in a *phase system* with respect to the volume of the *phase system*.
Formal definition: A *volume-based concentration* can be one of the following: *mass-volume percentage*, *molarity*, or *volume-volume percentage*.

Volume fraction
The volume of a *phase component* divided by the sum of volumes of all *phase components* of the *phase system* prior to mixing; for ideal mixtures, this equals to the *volume-volume percentage*.

Volume-volume percentage
Volume of a *phase component*, divided by the total volume of the *phase system*. For ideal mixtures, this is the same as the *volume fraction*.

Volumetric phase ratio
The ratio of the volume of a *single phase in multiphase system* to the volume of the *multiphase system*.

Relation Descriptions

hasReaction
The relation hasReaction links a *chemical reaction* to the *phase system* or *phase interface* where it takes place.

refersToReaction
The relation refersToReaction links a *reaction equilibrium constant* to a *reversible reaction*.

hasStateOfAggregation
The relation hasStateOfAggregation indicates the *state of aggregation* of a *single phase*.

representsOccurrenceOf
The relation representsOccurenceOf establishes the relationship between a *phase component* and the corresponding *chemical component*, or between a *phase reaction* and the corresponding *chemical reaction*.

refersToFirstSinglePhase
The relation refersToFirstSinglePhase indicates the first one of the two *single phases* to which a *phase equilibrium ratio* of a *phase component* refers.

refersToSecondSinglePhase
The relation refersToSecondSinglePhase indicates the second one of the two *single phases* to which a *phase equilibrium ratio* of a *phase component* refers.

7.4 References

CAS (2007) CAS Registry Overview. Website, online available at http://www.cas.org/EO/regsys.html. Accessed February 2007.

Dietz A (1995) Yet another representation of molecular structure? *J. Chem. Inf. Comput. Sci.* **35** (5):787-802.

Drewitz W, Szczepanski R, Pinõl D, Banks P, Halloran M, van Baten J, Pons M, eds. (2006) Thermodynamic and physical properties, version 1.1. CAPE-OPEN Interface Standards Specification. Online available at http://www.colan.org/index.html. Accessed March 2007.

EBI – European Bioinformatics Institute (2008) Chemical Entities of Biological Interest (ChEBI). Online available at http://www.ebi.ac.uk/chebi/. Accessed June 2008.

Gold V, Loening KL, McNaught AD, Sehmi P (1987) *Compendium of Chemical Terminology*. Blackwell, Oxford.

Gutsche B (1986) Phase equilibria in oleochemical industry – application of continuous thermodynamics. *Fluid Phase Equilib.* **30**:65-70.

Hariu OH, Sage R (1969) Crude split figured by computer. *Hydrocarbon Process., Int. Ed.* **48** (4):143-148.

Kumar A, Gupta R (1998) *Fundamentals of Polymers*. McGraw-Hill, New York.

Levenspiel O (1999). *Chemical Reaction Engineering*. John Wiley & Sons, Inc., New York.

Linstrom PJ, Mallard WG, eds. (2005) NIST Chemistry WebBook. NIST Standard Reference Database Number 69, June 2005, National Institute of Standards and Technology, Gaithersburg, MD. Online available at http://webbook.nist.gov.

McNaught AD, Wilkinson A, eds. (1997) Compendium of Chemical Terminology, 2nd Edition. Blackwell Science, Oxford, UK. Electronic version online available at http://goldbook.iupac.org/.

Modell M, Reid RC (1983) Thermodynamics and its Applications. Second ed., Prentice-Hall, Englewood Cliffs.

Moss GP, ed. (1996) Basic terminology of stereochemistry. IUPAC Recommendations 1996, *Pure Appl. Chem.* **68**:2193-2222.

Moss GP, Smith PAS, Tavernier D, ed. (1995) Glossary of class names of organic compounds and reactive intermediates based on structure. IUPAC Recommendations 1995, *Pure Appl. Chem.* **67**:1307-1375.

Müller P, ed. (1994) Glossary of terms used in physical organic chemistry. IUPAC Reccomendations 1994, *Pure Appl. Chem.* **66**:1077-1184.

OLS – Ontology Lookup Service (2006) ChEBI Ontology Browser. Online available at http://www.ebi.ac.uk/ontology-lookup/browse.do?ontName= CHE BI. Accessed February 2007.

Sandler SI (1999) *Chemical and Engineering Thermodynamics*. Third ed., John Wiley, New York.

Schummer J (1998) The chemical core of chemistry I: a conceptual approach. HYLE – *Int. J. Philosophy Chem.* **4** (2):129-162.

Smith EG (1968) *The Wiswesser Line-Formula Chemical Notation.* McGraw-Hill, New York.

Smith JM, Van Ness HC (1975) Introduction to Chemical Engineering Thermodynamics, 3rd Edition. McGraw-Hill, New York.

Stein SE, Heller SR, Tchekhovski D (2003) An open standard for chemical structure representation – the IUPAC Chemical Identifier. In: *Proceedings of the 2003 Nimes International Chemical Information Conference*:131-143.

Weininger D (1988) SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **28**:31-36.

8 Chemical Process Systems

Chemical_process_system is the key partial model in the ontology which addresses the body of knowledge of the chemical engineering discipline. However, it is not intended to function as a textbook or handbook for this discipline. Primarily, this partial model is aimed to present an organization of chemical engineering knowledge, by means of positioning, grouping, and linking chemical engineering concepts within a number of ontology modules. The descriptions or definitions of individual concepts presented in this partial model are all serving this purpose; more rigorous and comprehensive explanations of these concepts should not be sought in this section or in this ontology, but rather in the specialized textbooks or handbooks.
Furthermore, as part of a particular version of a continuously developed ontology, this partial model is intended to provide a rather complete coverage of the high level concepts within its intended scope, complemented with the concepts at lower levels (i.e. those of greater detail) which are introduced either because they have been needed by the existing applications of the OntoCAPE ontology or simply for illustrating how the extension to some specific high level concepts can be done.
Due to the broadness of this discipline and the diversity of applications, it is perhaps unrealistic for any particular version of the ontology to cover all chemical engineering concepts. However, the design of this ontology, particularly the differentiation between the Conceptual Layer and the Application-Oriented Layer, will allow any extension to be introduced as needed in the future.

8.1 Chemical Process System (Partial Model)

The central **chemical_process_system** partial model represents all those concepts that are directly related to materials processing and the corresponding operating devices. Hence, it is a representative elaboration of the primitives defined in *technical systems* (cf. Sect. 5.3). Overall, it is composed of 7 partial models which are located on the Conceptual Layer as well as the Application-Oriented Layer and thus it is the largest of all partial models in OntoCAPE. These 7 partial models are: **CPS_behavior**, **CPS_function**, **CPS_peformance**, **CPS_realization**, **process_units**, **plant_equipment** and **process_control_equipment**. An overview on these partial models including the major ontology modules is given in Fig. 8.1. Furthermore, **chemical_process_system** imports modules from other partial models such as **material** or **geometry**.
The four partial models **CPS_function**, **CPS_realization**, **CPS_behavior** and **CPS_performance** define the core of the **chemical_process_system** with respect to the Conceptual Layer.

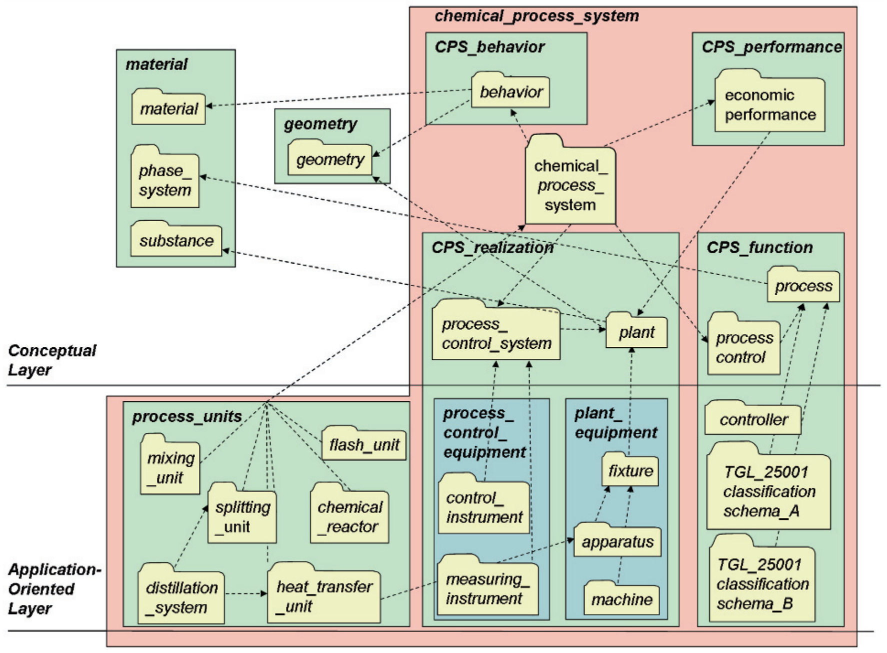

Fig. 8.1: Overview on the partial model **chemical_process_system**

CPS_function represents the *desired behavior* of a chemical process system (cf. Sect. 5.3.1). Thus, it holds the functionality of material processing, describing the chemical, physical, or biological procedures as well as the concepts related to information processing for process automation. Subsequently, **CPS_realization** reflects the physical constitution of the chemical process system (cf. Sect. 5.3.2). Accordingly, it comprises the technical realizations through plant equipments and the technology to operate the plant (i.e., observation and control). Furthermore, **CPS_behavior** describes how a chemical process system operates under certain conditions (cf. Sect. 5.3.3). It consists of (1) the phenominological description of the processed material amount and (2) the required connectivity between certain states of material amount. Finally, **CPS_performance** is introduced which may be applied for the evaluation and benchmarking of a chemical process system (cf. Sect. 5.3.4). Since the understanding of evaluation is very broad, only evaluation in the perspective of economics is included exemplarily for the time being.
The ontology modules that are incorporated within the aforementioned partial models may be distinguished with respect to whether they model the physico-chemical processing part or the operation and control part of a chemical process system. Considering the former one, *process* represents the functional viewpoint which describes the intended, ideal behavior (or intended effects resulting from the behavior) of the chemical plant in terms of certain changes of the significant states of material streams or amounts, which essentially indicates the designated

engineering purpose of a plant or a plant item. *Plant* describes the realization viewpoint which depicts the constitution of a chemical plant in terms of plant items (mainly pieces of equipment). Furthermore, *behavior* depicts the behavioral viewpoint of a *chemical_process_system* which mainly characterizes the physico-chemical phenomena occurring when materials are processed in an equipment. However, to characterize the operation and control part, modules *process_control* and *process_control_system* are introduced. The module *process_control* presents the functional description which basically comprises observation and control devices, and *process_control_system* represents the realization aspect which is essentially the implementation and installation of devices for control. Equally well applicable to both parts is the module *economic_performance* which covers the performance viewpoint mainly with respect to costs and revenue.
Regarding the Application-Oriented Layer, the two partial models **process_control_ equipment** and **plant_equipment** constitute specializations of the partial model **CPS_realization**. Specifically, they comprise more detailed information on *apparatuses*, *machines* and *instrumentation*. Ultimately, the partial model **process units** gives exemplary applications of **chemical_process_system**, e.g. a *reactor* or a *flash_unit*.

8.1.1 Chemical Process System (Ontology Module)

The module *chemical_process_system* introduces the class *chemical process system* as a special *technical system* (cf. Sect. 5.3), which is designed, built, and run in order to produce chemical compound(s) or material product(s) with given specifications. This definition is in accordance with the systems engineering definition of a system given by e.g. Patzak (1982).
Essentially, *chemical process system* and *process unit* are the major classes of this module and they are modeled as special types of a *technical system*. All the information about a *chemical process system* is conveyed via its *aspect systems*. The *aspect system* is introduced in a way such that it can be used to describe the various phases during the lifecycle of a *chemical process system* starting from design to operation. For the time being, the *aspect systems* have been developed and elaborated mainly for the purposes of design processes e.g. desired function of a processing or operating section. Hence, in the following we demonstrate the applicability of the *aspect systems* exemplarily for the design process of a processing part. Note that the aspects dealing with the development of the operating system could have been applied equally well. Further aspects such as operation, maintenance or the like could be easily added. Fig. 8.2 gives a schematic representation on how the aspect systems interact in a hypothetical scenario in which the major aspects in the lifecycle of a chemical plant are fully elaborated.

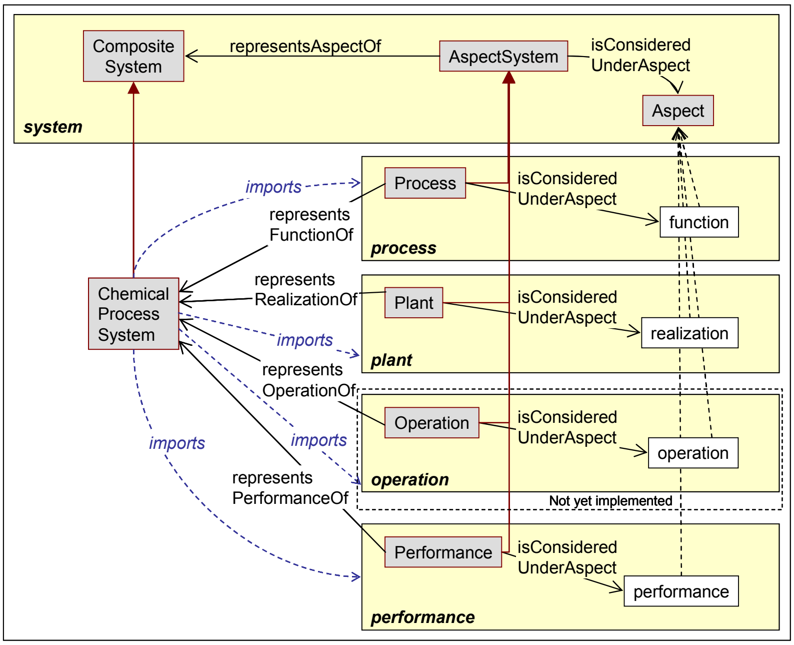

Fig. 8.2: Exemplary presentation of the aspect system considering the processing part of a *chemical process system*

According to the principles of decomposition of systems, a *chemical process system* can be composed of a number of *process units*. Since it is typically not intended to specify a complete *chemical process system* at once, one may rather focus on the modeling of a single *process unit* that builds a part of the overall system. Hence each *process unit* describes a distinct part of the entire *chemical process system* whose aspects may deviate from the one considered overall, e.g. the functional representation of a reactor differs from the one of the plant as a whole. In the following the respective *aspect systems* of a *process unit* are introduced from a design process perspective:

– The class *process step* represents the desired function (cf. Sect. 8.2.1).
– The class *plant item* reflects its physical realization (cf. Sect. 8.3.1).
– A *material amount* describes the physicochemical behavior of a *chemical process system* (cf. Sect. 8.6.1).
– Its (economic) performance is characterized by the concept of *economic performance* (cf. Sect. 8.7.1).

The graphical representation is given in Fig. 8.3. Note that the classes *plant item*, *process step*, *material amount* and *economic performance* are modeled as subclasses of *system* within their respective modules. However, in the *chemical process system* module, they are identified as *aspect systems* of *process unit*. For reasons of clarity the assignment of *economic performance* (i.e. representsPerformanceOf) is omitted.

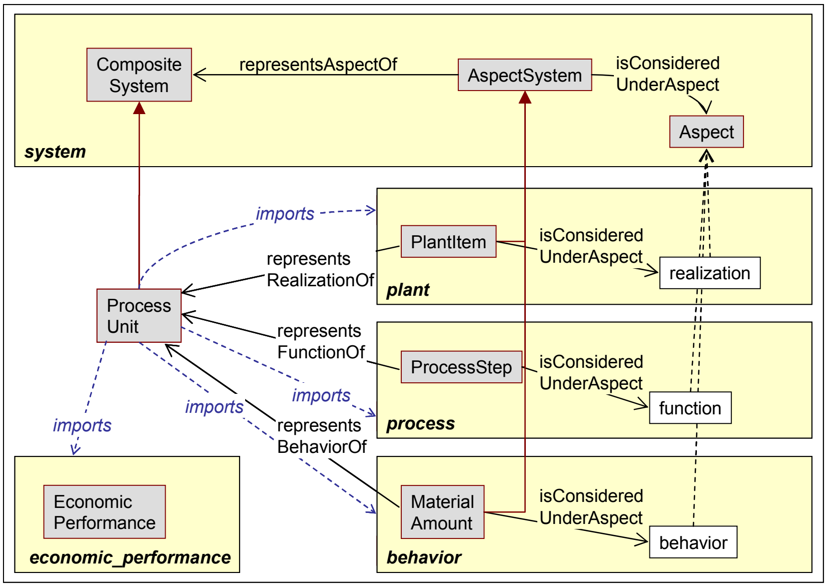

Fig. 8.3: Presentation of the aspect system considerations for *process unit*

Applying the same modeling approach as adopted in the *technical_system* (cf. Sect. 5.3) module, special dependencies of interest between the *aspect systems* are predefined within *chemical_process_system*. Exemplarily, the dependencies between the functional and realization view in a design process are presented in the following. Based on the workflow within the course of a design process, the information provided in a piping and instrumentation diagram (P&ID) are derived from the process flow diagrams (PFD) developed initially. Regarded from the perspective of aspect systems, a P&ID corresponds to a realization view of *process unit* whereas information stated in the PFD is considered to be a functional description. Accordingly, a *plant item* is said to realize a certain *process step* as it is shown in Fig. 8.4.
For example, a *process unit* that is intended to carry out a particular chemical reaction will be given the function of a *reaction* which will be realized by some reactor suitable to yield the desired product.
In the following sections the concepts stated within the respective partial models will be discussed, and their interdependencies considered. To that end, all modules already proposed in Fig. 8.1 will be introduced and described in detail.

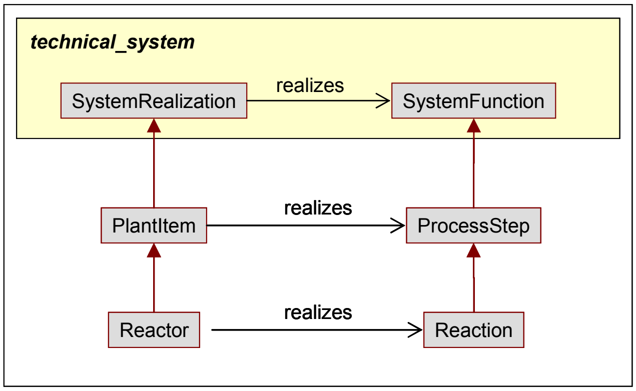

Fig. 8.4: Predefined dependencies between *aspect systems*

8.1.1.1 Concept Descriptions

Individual concepts of the module *chemical_process_system* are defined below.

Class Descriptions

Chemical process system
The *chemical process system* is a *technical system* which is designed, built, and run in order to produce chemical product(s).

Process unit
Process unit comprises parts of a *chemical process system* that are considered under specific aspects but do not represent the entire *chemical process system.*

8.2 CPS Function

The partial model **CPS_function** deals with the functional description of the *chemical_process_system*. Basically, it indicates the fundamental engineering purpose of a plant or a process control system by describing the intended behavior on a conceptual level. The functional description as it is implemented in OntoCAPE for the time being covers material processing devices as well as information processing devices, which are needed for the operation of a chemical process system.

The concepts provided enable a process description on a level of detail which is equivalent to that of a process flow diagram. In this stage of the process design lifecycle, the following items are usually specified:

– *Process steps* which hold certain material processing procedures (e.g. a reaction unit or a separation sequence) to acquire the overall process functionality.
– *Process states* which indicate the state of a processed material leaving or entering a particular *process step* with respect to the operational mode employed (e.g. either batch or continuous) and product classification (e.g. intermediate or waste product).
– Major operating conditions such as pressures and temperatures and some plant item design specifications such as the volume of a process unit.
– Connections representing the order of *process steps* and hence the identification of upstream and downstream parts of the *process*.
– Concepts for *process control*.

Subsequently, a comprehensive description of the partial model is provided. The partial model itself comprises the modules *process*, *process_control* and *controller*.

8.2.1 Process

The module *process* covers the functional viewpoint of the *chemical_process_system* with respect to material processing. It gives a conceptual view on the desired process, as it is often applied in an early stage of a design process. In the following, all relevant information regarding the decomposition will be stated first; subsequently, connectivity issues will be discussed.

8.2.1.1 High-Level Concepts

Within a chemical plant, material is processed in order to produce some specified product from raw materials. This processing comprises physical, chemical, and biological procedures that are performed in a specific order. These procedures can be subsumed as *process steps* (cf. Fig. 8.5).

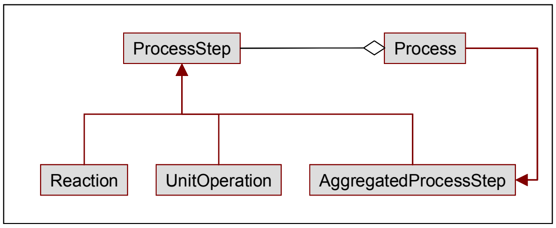

Fig. 8.5: Main elements of the ontology module *process*

Separation, mixing, and evaporation can be considered as examples for *process steps*. Chemical and biological transformations are described by *reactions* whereas *unit operations* are elementary *process steps* where only physical phenomena occur. However, *unit operations*, *reactions* and other *process steps* can be combined to complex procedures; these are captured by the class *aggregated process step* that must contain two or more *process steps*. Finally, the chemical *process* in its entirety is one special *aggregated process step*.

8.2.1.2 Modeling of Decomposition

In the following, initially a further classification of *unit operations* is given and then *reactions* are discussed on a conceptual level.

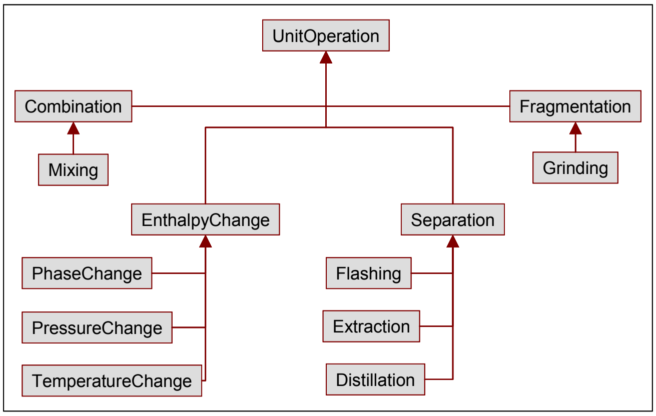

Fig. 8.6: Representation of *unit operation* in *process*

While the number of chemical processes is enormous, there are a relatively small number of *unit operations* that can be combined to form many kinds of *processes*. There are several chemical engineering handbooks dealing with *unit operations* and their realization in some equipment (e.g. Green and Perry 1998; McCabe et al 1993). Also, the data model developed by the pdXi initiative (ISO 10303 1998) contains a class hierarchy of *unit operations* frequently applied for a classification. The TGL 25000 (1974) was a standard of the former German Democratic Republic and gives an exhaustive description of *unit operations*, their application, and equipment for their implementation. In summary, the categorizations of *unit operations* given by the handbooks and the data models can be used to develop an accordant class hierarchy which result in two alternative classifications of *unit operations*: classification schema (1) where the physical state of the processed processing materials serves as the organizing principle and classification schema (2) where the *unit operations* are organized according to theoretical chemical engineering considerations. In OntoCAPE the schema (1) is adopted due to reasons of clarity and since this classification schema is more relevant in the context of process design where unit operations are selected for performing required transformations of chemical products. Further information on the implementation of the two classification can be found elsewhere (Bayer et al. 2001).

Unit operations in OntoCAPE are distinguished according to four major types: *combination*, *enthalpy change*, *separation* and *fragmentation* (cf. Fig. 8.6). Note that *unit operation* is defined as elementary, such that all complex and composite *process steps* are represented by *aggregated process step*. For a complete list of specializations of the four major types, we refer to the formal specification of OntoCAPE.

A *reaction* is a *process step* which is based on chemical, biological or nuclear transformation. It can be elementary or composite. Classification schemas for *reactions* are published in the literature similar to the ones given for *unit operations* (e.g. Shreve 1978; Matthes 1959); in this context, the term unit process is often used to capture chemical reactions (Encyclopedia Britannica 2009). We do not adopt those structures in OntoCAPE.

Finally, for the specification of a particular chemical transformation, considered in a rather macroscopic perspective of an early design stage, the relation refersToChemicalReaction is introduced (cf. Fig. 8.7). That way, a reaction may be assigned to single or aggregated chemical reactions. For the classification of *chemical reactions,* we refer to the module *reaction_mechanism* (cf. Sect. 7.2.8)

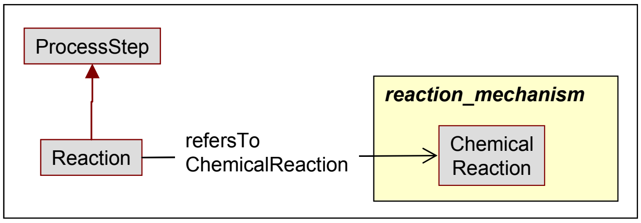

Fig. 8.7: Link to the related *reaction_mechanism* module class

8.2.1.3 Modeling of Connectivity

The individual *process steps* can be connected to each other via *process states*, as it is indicated in Fig. 8.8.
A *process state* represents the material that enters or leaves a *process step*. This includes the interchange between two *process steps* as well as the feeding and removing of material.

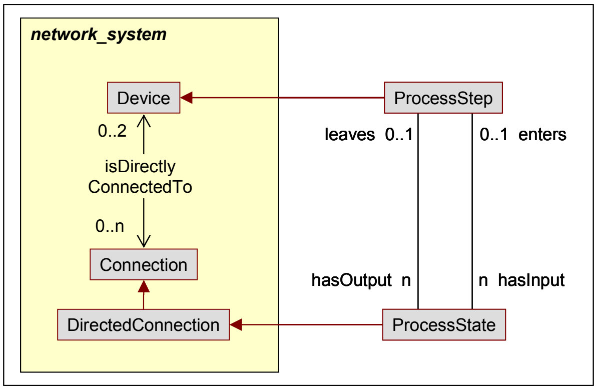

Fig. 8.8: Class diagram representing the topological relations in *process*

To define a *process state*, one must specify the amount of the material as well as its thermodynamic state (i.e., composition, temperature, pressure). For this purpose, a *process state* can be linked to a *generalized amount* (cf. Sect. 8.6.1) via the relation refersToGeneralizedAmount, as shown in Fig. 8.9.

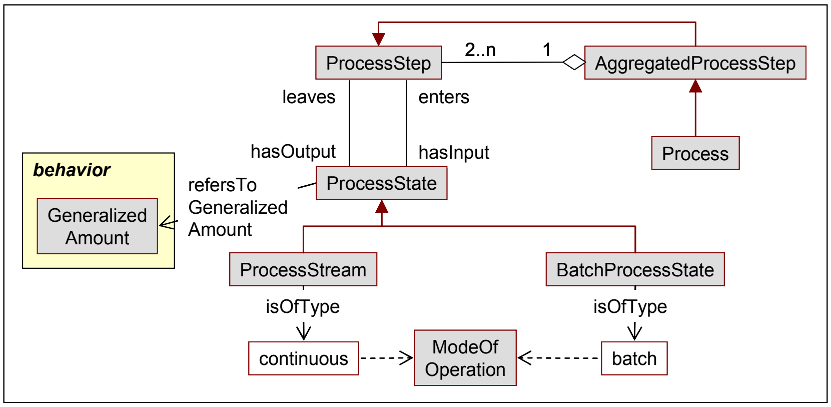

Fig. 8.9: Link to the related *behavior* module class

Depending on the mode of operation (i.e., batch, semi-batch, or continuous), a *generalized amount* may represent either a material flow (if it is a continuous process), or a material hold-up (if it is a batch process).
An assembly of interconnected *process steps* and *process states* forms a directed graph; consequently, the rules by which they can be connected are equivalent to those defined in the module *network_system* on the Upper Layer (cf. Sect. 5.2). Particularly, a *process step* may have various hasInput and hasOutput relations to a *process state* (e.g., a reactor that has multiple feeds). A *process state*, by contrast, must enter and/or leave at most one *process step*.

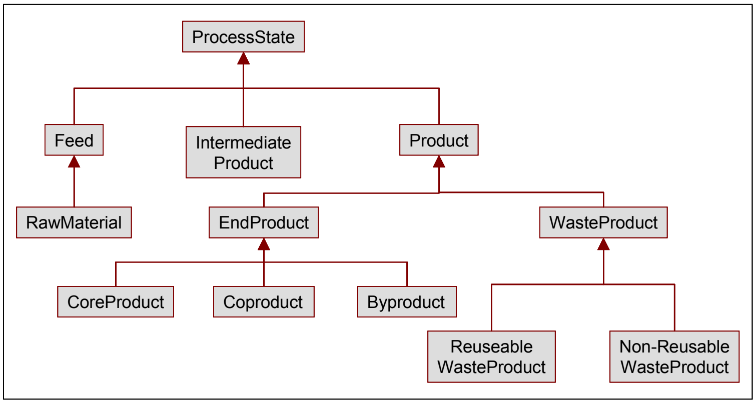

Fig. 8.10: Representation of the *process state* specialization

Note that the interconnected *process steps* and *process states* can be interpreted as a modified representation of a state-task network. A state-task network is a directed bipartite graph with states as its arcs and tasks as its nodes (Miller et al. 1997; Sargent 1998). In our representation, the *process steps* are considered as tasks, whereas the *process states* represent the states.
According to the state-task approach, chemical processes can be described on a very detailed level, independently of their mode of operation (i.e., batch, semi-batch, or continuous). Nevertheless, it may be of interest to specify the mode of operation with which a process state is associated: In a batch process, the *process state* represents the status of the process at a certain time; this is referred to as a *batch process state*. In a continuous process, the *process state* can be associated with a *process stream* at a certain location (cf. Fig. 8.9).
The *process states* might not only be classified in terms of mode of operation but also in terms of the value chain a process achieves. Therefore, the specialization as represented in Fig. 8.10 is proposed.
Note that the represented *process state* may be the result of intended or unintended effects in terms of physicochemical phenomena. The intended effects, corresponding to the functional viewpoint on the *process*, are realized by means of the *process*

steps. However, unintended effects due to non-ideal behavior are not considered here but within the module *behavior* (cf. Sect. 8.6.1).

8.2.1.4 Properties of Process

Both *process steps* and *process states* may have a number of *properties*, which uniquely define the design conditions of a *process step* or the effective operating conditions of a *process state*. Essentially, the predefined patterns for *property* representation described in *system* (cf. Sect. 5.1.9) are applied: First of all, the class *process step property* is introduced, which comprises specific properties with respect to *process steps*.

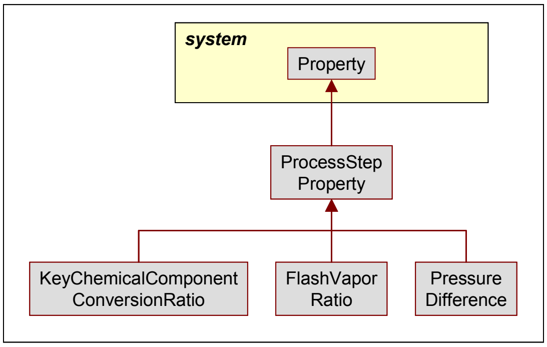

Fig. 8.11: Exemplary description of *process step properties*

Secondly, further specifications such as *key chemical component conversion ratio*, *flash vapor ratio* or a *pressure difference* are introduced exemplarily as shown in Fig. 8.11. Consecutively, the properties may be assigned to the *process steps* using either the hasProperty relation or specializations[83] thereof as depicted in Fig. 8.12.

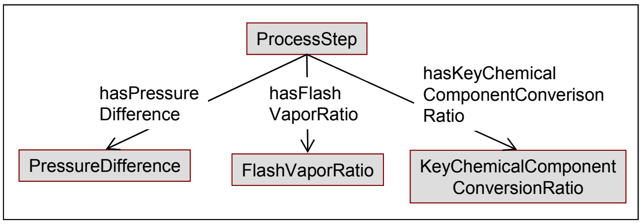

Fig. 8.12: Exemplary *property* assignment to *process step*

[83] These specializations are merely auxiliary constructs used as replacements for qualified cardinality restrictions (cf. Sect. 2.3.4). They may be discarded, as soon as qualified number restrictions are made available in OWL.

The *properties* of a *process state* can only be expressed through the *generalized amount* it refers to, as described in Sect. 8.6.1.

8.2.1.5 Application Example

Finally, an application example is discussed, dealing with the decomposition and connectivity of a process flow diagram. In Fig. 8.13 a simple flow diagram is given, showing the main *aggregated process step* (**Process**) and two major *process states* (**Feed**, **Product**). The **Process** is decomposed into two *process steps* (**Reaction**, **Separation**) linked by intermediate *process states* (**Recycle**, **Intermediate**).

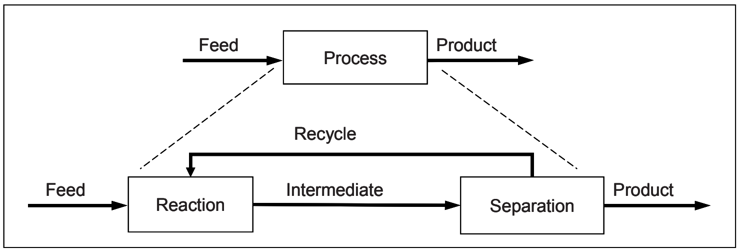

Fig. 8.13: Application example: Decomposition of process flow diagram

The OntoCAPE representation of the process flow diagram is presented in Fig. 8.14.

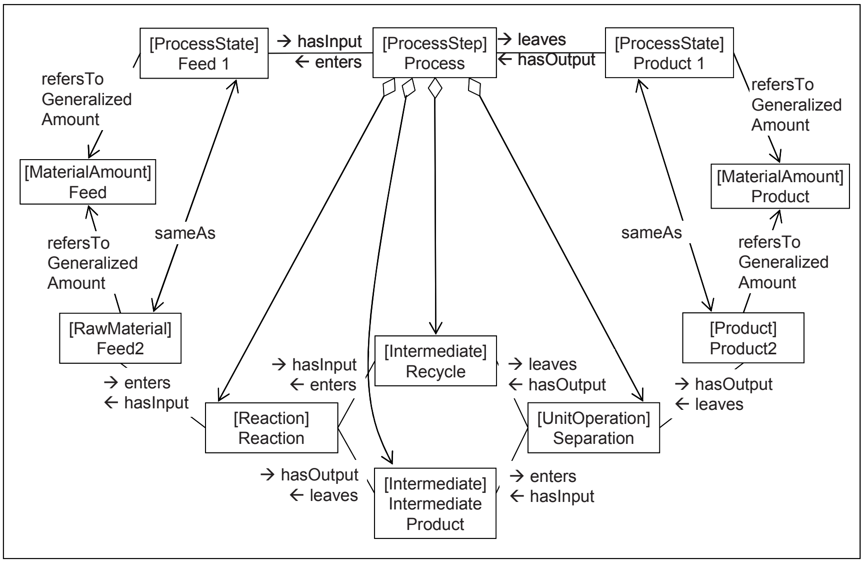

Fig. 8.14: Hierarchical decomposition of a process flow diagram

Both levels of hierarchy are modeled (the classes corresponding to the individuals are given in brackets). As usual, the topological relations are confined to the same decomposition level. Note that **Feed 1** and **Feed 2** denote the same *process state* on different breakdown levels[84]. Its separate occurrences at these levels are linked by the relation sameAs (cf. Sect. 5.2). The same is true for **Product 1** and **Product 2**.

8.2.1.6 Concept Descriptions

Individual concepts of the module *process* are defined below.

Class Descriptions

Aggregated process step
An *aggregated process step* is a *process step* which consists of a number of *process steps* in a certain sequence.

Batch process state
A *batch process state* represents a specific *process state* which is attained during a batch mode of operation.

Byproduct
A *byproduct* is an *end product* whose production is unavoidable while a *core product* is produced.

Combination
C*ombination* refers to these *unit operations* which combine different materials into one.

Core product
A *core product* is the main (or an intended) *end product*.

Coproduct
A *coproduct* is an *end product* whose production is unintended.

Distillation
Distillation is a *unit operation* for the separation of chemical substances based on differences in their volatilities in a boiling liquid mixture. For the realization of the *unit operation* often a distillation column is applied.

Enthalpy change
Enthalpy change is a *unit operation* which changes the enthalpy of the material being processed, which may materialize as the change in temperature, pressure, and/or state of aggregation.

[84] This is due to the unique name assumption required by OWL.

End product
An *end product* is a valuable *output product* that is not further processed.

Extraction
Extraction is a *unit operation* which separates a substance from a mixture by preferentially dissolving that substance in a suitable solvent. By this process a soluble compound is usually separated from an insoluble compound.

Feed
Feed is the supply of material to a certain *process step*.

Flashing
Flashing or flash evaporation is the partial vaporization that occurs when a saturated liquid stream undergoes a reduction in pressure by passing through a throttling device. This process is one of the simplest *unit operations* which is usually realized in a vessel.

Flash vapor ratio
Flash vapor ratio is the ratio between the vapor and the liquid phase in a flash, which is considered as a significant property for flash design.

Fragmentation
Fragmentation refers to *unit operations* to breakup or split a material for further processing to an unknown ratio, shape or structure like the crushing of solid materials.

Intermediate product
A compound which is produced in the course of a chemical synthesis, which is not itself the final product, but is used in further *process steps* which produce the *end product*.

Key chemical component conversion ratio
Key chemical component conversion ratio indicates the conversion ratio of the main product produced in a *reaction process step.*

Mixing
Mixing is a *unit operation* which, as a special type of *combination*, results in a mixture required for further processing. It is usually accomplished by means of stirrer.

Non-reusable waste product
A *non-reusable waste product* is a *waste product* that cannot be integrated in a process anymore and therefore has to be disposed.

Mode of operation
The *mode of operation* distinguishes between the two common modes of continuous and batch operation. *Mode of operation* is an enumeration of its instances **batch, semi-batch**, and **continuous**.

Output product
An *output product* is a material that has been processed by a certain *process step* and may be either further processed or sold or disposed.

Phase change
Phase change refers to the change in the nature of a phase as a result of some variation in externally imposed conditions, such as temperature, pressure, activity of a component or a magnetic, electric or stress field (McNaught and Wilkinson 1997).

Pressure change
Pressure change is a *unit operation* which changes the pressure of the material being processed. Typical examples are pumping, compression, and expansion.

Pressure difference
Pressure difference is a *process step property* which indicates the difference in the pressure of the material when it is processed by the *process step pressure change.*

Process
Process is an *aggregated process step* representing the entire function of the *chemical process system* considered.

Process state
A *process state* represents the collectivity of properties of a certain *material amount* which is produced in the associated *process step.*

Process step
A *process step* is a certain material processing procedure.

Process step property
A *process step property* is a *property* of a *process step*.

Process stream
A *process stream* represents a process state in a *process* operated in a continuous mode of operation.

Raw material
Raw material denotes the *process state* of material that enters a *process step* to be processed there.

Reaction
A *reaction* is a *process step* in which some material is converted to some other material through chemical, biochemical or nuclear reactions.

Reusable waste product
A *reusable waste product* is a *waste product* that can be reused and hence be integrated in the chemical synthesis for production again.

Separation
Separation refers to all *unit operations* which obtain a subset of mixture components at a specified amount by the application of physical or chemical pocesses.

Temperature change
Temperature change is a *unit operation* which transports the heat content of one material to another to result in a change in the temperature of the material being processed. It is applied for heating or cooling purposes.

Unit operation
A *unit operation* is a basic step in a *process*. Examples of *unit operation* are *mixing*, *separation*, *enthalpy change*, etc.

Waste product
A *waste product* is an *output product* that has no market value.

Relation Descriptions

hasFlashVaporRatio
The relation hasFlashVaporRatio indicates the liquid-vapor ratio of a particular mixture at a certain state within a vessel.

hasOperationMode
The relation hasOperationMode indicates by which *operation mode* a particular *process state* is achieved.

hasPressureDifference
The relation hasPressureDifference indicates the intended difference between two pressure states.

refersToGeneralizedAmount
The relation refersToGeneralizedAmount establishes a connection between a *process state* and a *generalized amount* in order to be able to describe extensive as well as intensive properties, e.g. relevant material quantities.

refersToChemicalReaction
The relation refersToChemicalReaction established a connection between a *reaction*, and a *chemical reaction* considered in a rather macroscopic perspective of an early design stage.

8.2.2 Process Control

The module *process_control* deals with the architecture, mechanisms and algorithms for controlling the outputs of a specific *process*. In particular, the concepts for representing the architecture of a control system are centered on a *control loop*.

In the following, we will first introduce the principles of decomposition applied in this module and then connectivity issues are discussed.
A *control loop* consists of *control components* and *action lines*. Fig. 8.15 shows the major elements of the module *process control* including the specialization of *control component*. *Control components* are the different functional components of *control loop*. The four subclasses of *control components* are the *comparing element, reversing element, branching element* and the *function block*. The *function block* represents those control components with major functionality within a *control loop*: *controllers*, *controlled system* (which is identical to some *process step* or *aggregated process step*), *sensor function* and *actuator function*. Besides *function blocks*, *comparing elements*, *reversing points* and *branching points* are needed to describe the function of the information processing which is performed within the *chemical process system*.
Since *function blocks* are typically described by mathematical model, such a connection is given by the relation refersToMathematicalModel. Furthermore, *function block* can be specialized to *sensor functions* and *actuator functions* (Polke 1994). The *sensor function* comprises the entire function of recording, relaying, and writing out *process quantities* within other *control components*. The *actuator function* transforms the output of the *controller* (controller output variable) into the input of the *controlled system* (manipulated variable), which is the actual variable that is changed within the *process step* in order to reach the intended conditions.

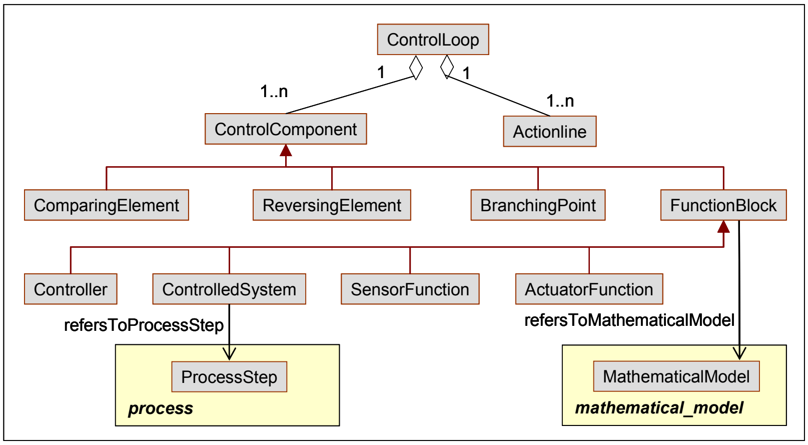

Fig. 8.15: Major elements of the module *process_control*

The architecture of a *control loop* is characteristic and is designed under consideration of the control objective as well as the particular *process* by connecting the different kinds of *control components*. However, frequently applied control loop architectures are summarized as *control loop architecture value types* including four basic types of *control loops* as shown in Fig. 8.16: **open_loop_control**, **feed_ forward_control**, **state_feedback_control** and **output_feedback_control**. Furthermore

complex_control_loops are considered which can contain two or more of the basic ones (e.g. cascade control or pre-control (Schuler 1999)).

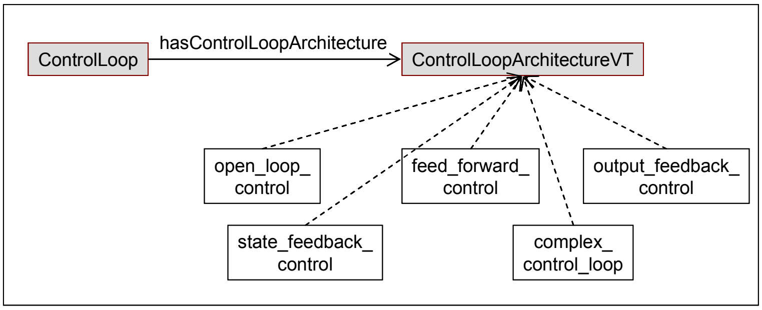

Fig. 8.16: Representation of *control loop architecture value type*

Function blocks are frequently distinguished as *linear function blocks* types and *nonlinear function blocks types* (cf. modeling of polyhierachies in Sect. 4.2.1 and Fig. 8.17). Further, static and dynamic function blocks can be distinguished (not shown in Fig. 8.17). Furthermore, *linear function blocks types* have a distinct transfer function, which can be used to characterize them (Unbehauen 1989). Another characteristic property of *linear function blocks* is their abstract behavior, i.e. the way they react on a change in their input variable. The indication of the response characteristics of a function block is indicated by the relation hasResponseCharacteristics.

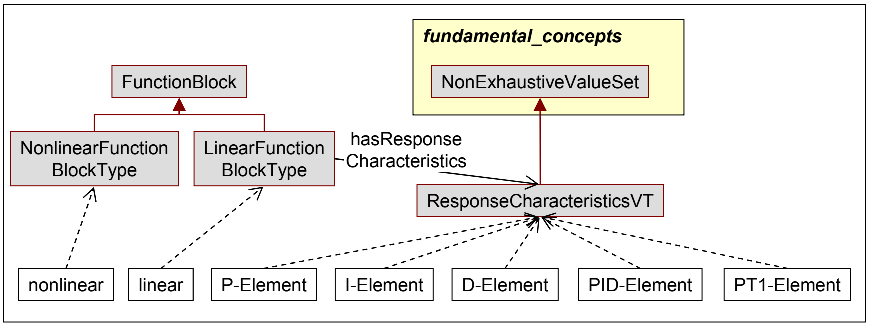

Fig. 8.17: Representation of system value types in the module *process_control*

In Fig. 8.17, some of the so distinguished control elements are shown; for a more complete list refer to literature in the area of control engineering and control theory (e.g. Unbehauen 1989). Some authors give classifications of *function blocks* that can also be integrated here (see for example Föllinger 1992).
Again, the topological relations applied in *process_control* obey the principles given by the partial model *network_system* (cf. Sect. 5.2 and cf. Fig. 8.18).

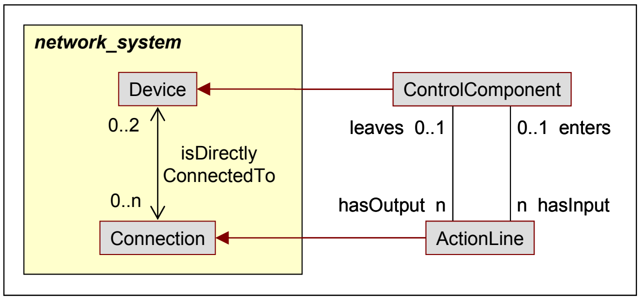

Fig. 8.18: Illustration of the topological relations in the module *process_control*

8.2.2.1 Concept Descriptions

Individual concepts of the module *process_control* are defined below. For a description of the respective value types introduced before, we refer to Wiesner et al (2008a).

Class Descriptions

Action line
An *action line* connects the different *control components* that eventually build the *control loop.*

Actuator function
Actuator function transforms the output of the *controller* into the input of the *controlled system.*

Branching point
Branching point describes the splitting of an action line.

Comparing element
Comparing element indicates how the state of a control component is influenced.

Control component
Control component is the part of a *control loop* which accomplishes substantial tasks in signal processing or generation.

Control loop
Control loop deals with the description of the functional elements and their implementation to manipulate a dynamic system to achieve a desired behavior.

Controlled system
Controlled system describes the functionality of the system to be controlled.

Control loop architecture VT
Control loop architecture value type comprises the different types of control loop structures. *Control loop type* is an enumeration of its instances **open_loop_control**, **feed_forward_control**, **state_feedback_control**, **output_feedback_control** and **complex_control_loop**.

Controller
Controller represents the different types of controller.

Function block
A function block describes the function between input variables and output variables

Linear function block type
A linear function block type represents the desired behavior which may be described by linear functions. It typically has a distinct transfer function, which can be used to characterize them.

Linearity VT
Linearity is an enumeration of its instances **linear** and **non-linear**.

Nonlinear function block type
A nonlinear function block type represents the desired behavior which may be described by nonlinear functions.

Response characteristics VT
Response characteristics VT describes the several characteristics how a controlled system may react on a manipulation.

Reversing element
Reversing element describes the functionality of lead.

Sensor function
The *sensor function* comprises the entire function of recording, relaying, and documenting *process quantities* within other *control components.*

Relation Descriptions

hasControlLoopArchitecture
The relation hasControlLoopArchitecture refers from a function block to its *control loop architecture value type*.

hasLinearity
The relation hasLinearity refers from a *function block* to its *linearity value type*.

hasResponseCharacteristics
The relation hasResponseCharacteristics refers from a *linear function block type* to its *response characteristics value type.*

8.2.3 Controller

The two preceding ontology modules provide fundamental principles for a functional description of a *chemical process system*. However, it is intended to present the major component of a *control loop,* namely the *controller,* in a more detailed way. Further specializations for different applications are possible. A graphical representation of some specialized controller types is given in Fig. 8.19.
The *PID controller* is the one that is most commonly used in industrial practice. *Adaptive controllers* are another special type of *controllers*. Furthermore, some more advanced *controllers* exist like *knowledge-based controllers* (e.g. *fuzzy controller*), or *model-based controllers*. Very important types of the latter are predictive controllers (Schuler 1999); examples are the MPC (*model predictive controller*), the IMC (*internal model controller*), and the *Smith predictor*.

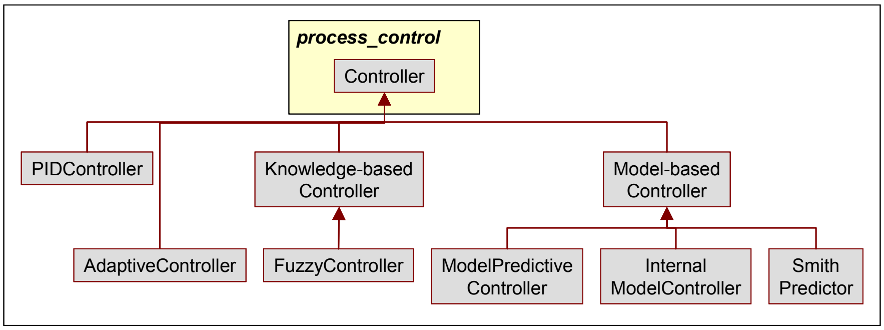

Fig. 8.19: Class diagram representing specialization of *controller*

For the description of the individual concepts of the module *controller*, we refer to Wiesner et al. (2008a).

8.3 CPS Realization

The partial model **CPS_realization** holds the *aspect* of **realization** and therefore the concepts required to specify the physical structure of a chemical plant. It allows describing the individual pieces of process equipment, including their fittings and *fixtures*, the connectivity of the *equipment* through *piping* systems as well as the control components required for the implementation of process automation systems. The concepts provided by **CPS_realization** enable plant descriptions on different levels of details to support various phases in the lifecycle ranging from basic design (a.k.a basic engineering) to detail design (a.k.a detail engineering). For the time being, the elaboration of **CPS_realization** mainly focuses on the following items:

- *apparatuses* and *machines* with respect to their type, construction materials, capacity, and main dimensions,
- the related *fixtures* and fittings;
- major design specifications such as maximum pressure or temperature;
- key *instrumentation* items (valves, sensors, control hardware…);
- *piping networks* and power and signal lines[85].

Geometrical details of the *equipment* and the 3D layout of the *plant* are further important information of a constitutional description of a chemical plant. However, these tasks have been considered with less priority compared to the specifications stated before and are correspondingly less elaborated. Thus, a plant description that is based on concepts from the partial model **CPS_realization** has approximately the information content of a piping and instrumentation (P&ID) diagram. Subsequently, a comprehensive description of the partial model is provided. The partial model itself comprises the modules *plant* and *process control system*.

8.3.1 Plant

Within the module *plant,* all elements of a *plant* are subsumed as *plant items*. There are three major subclasses of *plant item* (cf. Fig. 8.20): *equipment*, *transport channel* and *instrumentation*. In the following, the relevant constitution of the *plant* and the *plant items* involved are discussed first. The topological connectivity between *plant items* is explained thereafter.

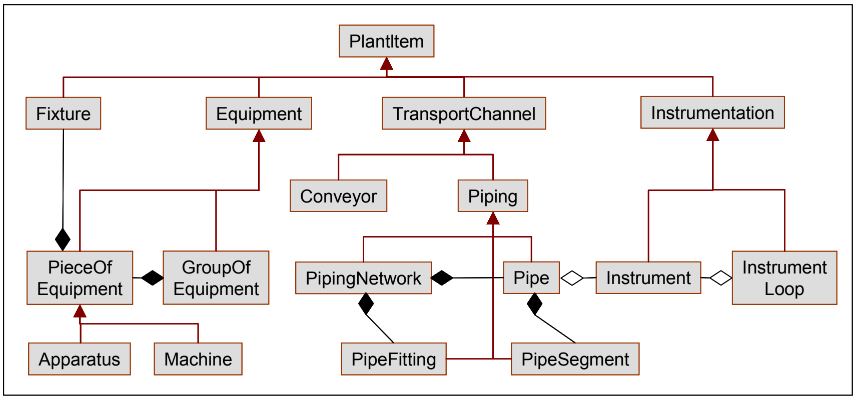

Fig. 8.20: Class diagram of mereological considerations of the module *plant*

[85] Power and signal lines are not considered in OntoCAPE for the time being.

8.3.1.1 Modeling of Decomposition

Equipment is a *plant item* that is capable of independently realizing one or more *process steps* (cf. Sect. 8.2.1). It represents all sorts of apparatuses and machines required for the production of a desired material. *Transport channel* is introduced as a general means to represent any transport of material in chemical plants. *Instrumentation* is required for automation purposes. Within the scope of this module, the on-site instruments for measurement and control are addressed. Additionally, *fixtures* are considered as a subclass of *plant items*. The existence of all kinds of *fixtures* enables the detailed representation of *plant items*, which even allows a description of unique or special *equipment*. The hierarchical organization of a chemical plant is decomposed in two levels of detail (cf. Fig. 8.21).

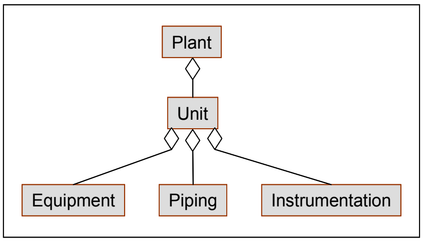

Fig. 8.21: Representation of the composition of *unit* and *plant*

Firstly, *units* are introduced, which comprise *equipment* and the required *piping* and *instrumentation* which in conjunction represent a production unit. A distillation system including the *piping* connecting the *apparatuses* and the sensors and controllers for automation is an example of such a *unit*. Secondly, a *plant* is the collection of specific *units* required for the production of the desired product.
Obviously, a *plant item* may consist of one or more other *plant items*, which motivates the need of decomposition.
Piece of equipment and *group of equipment* can be distinguished as two different subclasses of *equipment*. A *piece of equipment* is an elementary part of *plant*, though elementary as a subclass of *equipment*, it can be further composed of some *fixtures*. An example for a *piece of equipment* is a tubular reactor that contains the fixtures *shell* and *tubes*. Two types of *piece of equipment* are distinguished fully consistent with (TGL 25000 1974): (1) *apparatus*, which consists mainly of non-moving parts and whose main purpose is the transformation of material properties with respect to their intensive state variables and other characteristic attributes; (2) *machine*, which is often used to transform energy (e.g., *pump*) or to transfer energy into some material to achieve some physical effects (e.g., centrifuge). A *group of equipment* is a collection of *pieces of equipment*. For example, a *distillation column* with its reboiler and head condenser is considered as a *group of equipment*.
Piping is a specialization of *transport channel* and thus introduced to address any transport of material between two *plant items* which is often realized through pipes.

In particular, fluids are transported by means of piping. However, also solids need to be transported between *equipment*. Depending on the particulate characteristics of the solid, a fluid may be used as a carrier, which again facilitates transport in *pipes*. In case the solids' state of matter do not allow transportation in pipes, a *conveyor* is introduced as an exemplar for the transport of all other solids and materials[86]. *Piping network, pipe, pipe segments* and *pipe fittings* can be distinguished as four different subclasses of *piping* (cf. Fig. 8.22).
A *pipe* represents the elementary unit of *piping* and can be composed of *pipe segments* and *instruments*. Again, *pipe segments* may have additional properties, e.g. the degree of insulation or the material of construction. However, *piping network* represents a collection of *pipes and pipe fittings*. Forking of a connection to feed several reactors from one tank is an example of a piping network.

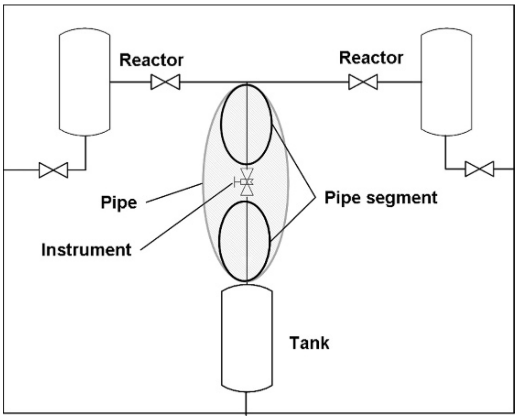

Fig. 8.22: Illustration of the *piping network*

Finally, for *instrumentation* the two subclasses *instrument* and *instrument loop* are introduced. The instrumentation fittings between the several instruments are not yet covered by this module. A further distinction between *actuator* and *sensor* units is presented in adjacent modules. Examples for instruments are *valves* which are used to control the flows of fluids, and *temperature sensors* for measurement. An *instrument loop* consists only of instrumentation, more precisely of at least one *sensor* and one *actuator*. An *instrument loop* is applied as a whole to some equipment, e.g. it might be used for temperature control of a *column*. According to the model of *system* (cf. Sect. 5.1.6), the classes *piece of equipment* and *group of equipment* correspond to *subsystem* and *supersystem*. Similarly, the class *pipe* and *piping network* are classified as *subsystem* and *supersystem*. Two extensions to the relations already defined for *system* are provided. The relations hasConnector and hasFixture are introduced as specializations of the isComposedOf relation and analogously the in-

[86] Note that any non-piping channel is not explicitly covered by OntoCAPE for the time being.

verse relations isConnectorOf and isFixtureOf are specializations of isExclusivelyPartOf.

8.3.1.2 Modeling of Connectivity

In the *system* module, *system interface* is introduced primarily to allow for a consistent and realistic representation of connectivity between *systems*. Since *plant item* is a specialization of *system* and hence *piece of equipment*, *pipe* and *instrument* are *subsystems,* suitable interfaces have to be introduced. These interfaces and the resulting connectivity within the scope of the module *plant* is addressed next. The corresponging class diagram is represented in Fig. 8.23.
The three subclasses of *system interface* introduced to address connectivity are *nozzle*, *pipe segment end* and *instrumentation connection.* This way, all sorts of conceivable connections between *piece of equipment, piping network* and *loop* can be created with these topological concepts. However, *nozzle* can only be connected to either *pipe segment ends* or *instrumentation connection* to build a valid connection. In turn *pipe segment ends* and *instrumentation connection* might be linked to oneself.
A *piece of equipment* and a *group of equipment* may have as many *nozzles* as required to fulfill the function. By definition, a *pipe* always consists of two *pipe ends* to be connected to equipment*, piping or instrumentation.* However, a *piping network* might have more than two *pipe segment ends* for connectivity resulting from the forking of *pipes.* An *instrument* may have either one or two *instrumentation connections* depending on the installation condition. As an example, a temperature sensor fitted to a *nozzle* needs only one *instrumentation connection*, whilst a valve usually is fixed between two *pipe segment ends*.

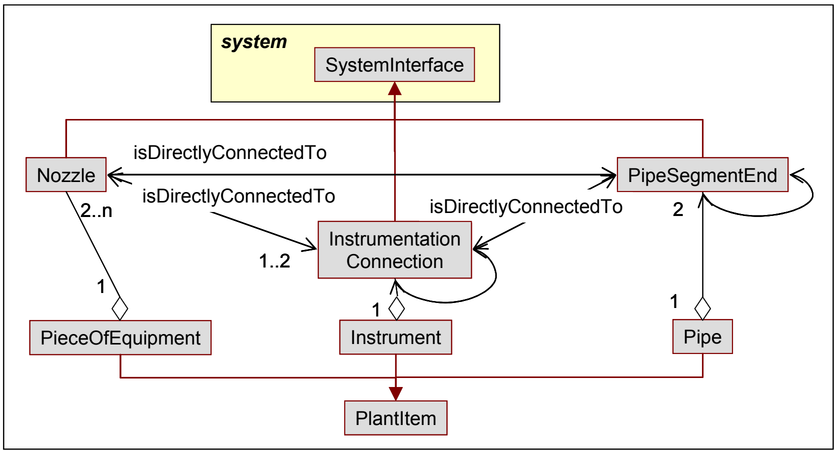

Fig. 8.23: Class diagram of topological considerations of the module *plant*

8.3.1.3 Properties of Plant

The characteristic properties of a *plant item* are modeled according to the patterns governed by *system properties*. Therefore, a *plant item* uses the relation hasProperty to refer to the corresponding *property* which in turn refers to the correct *value*. Essentially, two basic types of plant item properties are introduced, *design property* and *construction property*. *Design properties* can be considered as extended *process step properties* (cf. Sect. 8.2.1) basically including process specification such as maximal and minimal pressure and some equipment specifications like the type of building material of a vessel. *Construction properties* constitute a further enrichment in terms of technical specifications and include very detailed information required for the actual building of a plant item. An example of construction properties may be the maximal allowable pressure or the thickness of the wall of a *vessel* or a *machine*.
Some *properties* can be considered to be constant within the life cycle of a *plant item*. Also others may not only describe a particular feature of a single *plant item* but could be assigned to a wide range of *plant items*. These kinds of *properties* are allocated to *plant items* by the relation hasCharacteristics (cf. Sect. 5.1.14) which is directly linked to either an *enumeration of values* or a *qualitative value*. An example is the characteristic of *geometry*, e.g. a *column* hasCharacteristics **cylindrical** or a *pipe* which is designed according to a specific piping class.

8.3.1.4 Application Example

Subsequently, a simple example is presented to demonstrate the application of the aforementioned classes. The example is shown in Fig. 8.24. The feed to a reactor is stored in a tank, both pieces of equipment are connected through a pipeline. The pipeline consists of two segments and a valve for operational reasons and can therefore be represented on a more detailed level.

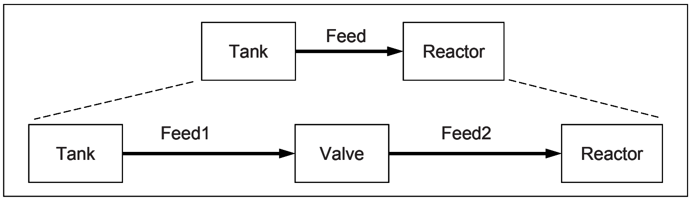

Fig. 8.24: Application example to illustrate the plant ontology

Fig. 8.25 shows the representation of the unit applying the classes of the ontology module *plant* to illustrate the basic connectivity of the elements. A coarse-grained description may simply distinguish between the **tank** and the **reactor** and the connecting **feed**. If, however, the detailed representation of *instruments* is of importance, one may choose a more fine-grained description where the *piping network* is decomposed into *pipe segments* and the in-between instrument (e.g. **valve**).

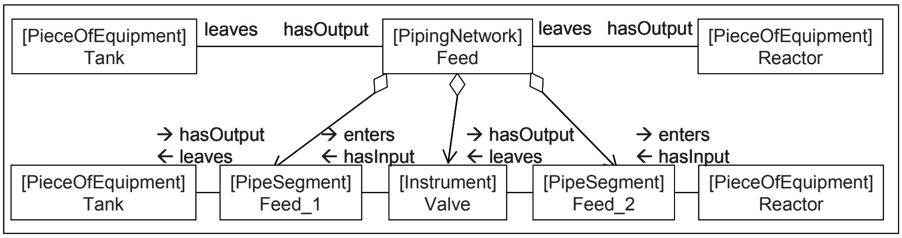

Fig. 8.25: Representation of the application example according to *plant*

Additionally, the use of *system interfaces* may be demonstrated for this example. The resulting representation (with the reactor excluded for simplicity) is shown in Fig. 8.26. It is illustrated how the piece of equipment, i.e. **tank,** is connected via a nozzle, i.e. **tank nozzle 1,** to a *pipe segment end*, i.e. **PSE feed 1 tank**, of the adjacent *piping network*; the *pipe segment end* is furthermore connected to the *instrument connection* and so on.

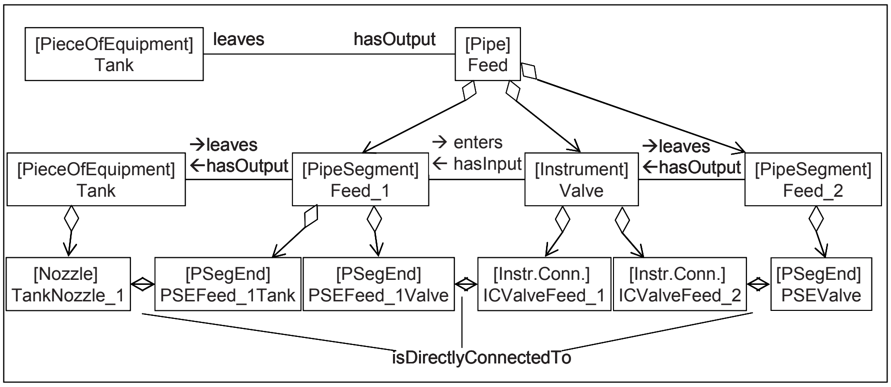

Fig. 8.26: Representation of system interface connections.

8.3.1.5 Concept Descriptions

Individual concepts of the module *plant* are defined below.

Class Descriptions

Apparatus
An *apparatus* is a *piece of equipment* which mainly consists of non-moving parts and provides space for materials to be processed.

Construction property
A *construction property* constitutes a further enrichment in terms of technical specification of the design data and includes very detailed information required for the actual building of a *plant item*.

Design property
A *design property* is an extended *process step property* basically including process data such as maximal and minimal bounds of design properties and some construction considerations.

Equipment
Equipment is a *plant item* that is capable of independently realizing one or more *process steps*.

Fixture
A *fixture* is a *plant item* that is part of *equipment* and therefore not capable of independently realizing a *process step*. The function of a *plant item* depends on the required *fixtures*.

Group of equipment
A *group of equipment* comprises a number of *pieces of equipment* which realizes a *process step*.

Instrument
An *instrument* is a device used to measure or control one or more *properties* of a *system*.

Instrument loop
An *instrument loop* is a set of *instruments* which are arranged to regulate one or more variables of a certain controlled system.

Instrumentation
Instrumentation is about measuring and control and subsumes *instrument* and *instrument loop*.

Machine
A *machine* is any mechanical or electrical device that transmits or converts energy to perform a task.

Nozzle
A *nozzle* represents the interface through which a *plant item* can be connected to another *plant item* or to the environment of a *plant*.

Piece of equipment
A *piece of equipment* is an elementary unit in the sense that it does not include other *equipment* or *pipes* or *instrumentation*.

Pipe
A *pipe* can be used to connect one *plant item* to another *plant item* or to the environment of a *plant*.

Pipe fittings
Fittings are used in *piping networks* to connect straight *pipe* sections, to adapt to different sizes or shapes or to realize forking of *piping*.

Pipe segment
A *pipe segment* is the elementary part of *piping*. A *pipe* may be assembled of a number of *pipe segments*.

Pipe segment end
A *pipe segment end* is an on- or off-page *connector* to another *pipe, piece of equipment* or *instrument*.

Piping
A *piping* is a *plant item* which is used for fluid transport. It may connect *equipment* or/and *instruments*.

Piping network
A *piping network* is a collection of connected *pipes* and *pipe fittings* used to connect multiple *pieces of equipment*.

Plant
A *plant* aggregates at least two *units* to realize a whole *process*.

Plant item
A *plant item* is an object which exists, in a material form, in a *chemical process system* for realizing one or more *process steps*.

Plant item property
A *plant item property* represents all properties that are particularly important for the description and specification of *plant items*.

Unit
A *unit* is a collection of associated *equipment* modules, *instrumentation* modules and *transport channels* in which one or more major *process steps* can be conducted. It represents one section of the overall *plant*.

Relation Descriptions

hasCapacity
The relation hasCapacity indicates an *equipment*'s capacity.

hasConnector
The relation hasConnector refers from a *piece of equipment* or a *pipe segment* or an *instrument* to the corresponding *plant item interface*.

hasConstructionMaterial
The relation hasConstructionMaterial refers to the *material* chosen for the construction of *plant items*.

hasEfficiency
The relation hasEfficiency indicates a *machine*'s efficiency.

hasFixture
The relation hasFixture refers from *equipment* to the corresponding *fixture*.

hasHeight
The relation hasHeight refers to the height of equipment.

hasInsideDiameter
The relation hasInsideDiameter indicates the inside diameter of a *piece of equipment*.

hasOutsideDiameter
The relation hasOutsideDiameter indicates an *equipment*'s outside diameter.

hasPowerOutput
The relation hasPowerOutput indicates a *machine*'s magnitude of power output.

isFixtureOf
The relation isFixtureOf refers from a *fixture* to the corresponding *equipment*.

isConnectorOf
The relation isConnectorOf refers from a *plant item interface* to the corresponding *piece of equipment* or *pipe segment* or *instrument*.

8.3.2 Process Control System

The module *process_control_system* describes the realization of the part of the *chemical_process_system* which is introduced for control purposes. Hence a *process control system*, which belongs to a specific *plant*, incorporates the implementation of the functionality represented by *process control*.
The major elements of this partial model are shown in Fig. 8.27. In a general sense, the *process control system* comprises all devices such as *measuring instruments* (i.e. sensors, measuring instruments, analyzers), *controlling instruments* (actuator systems such as control valves or switches), and *human-process communication devices* (process operator stations, displays and control panels). Consequently, these latter classes are summarized by the class *process control device*. Moreover, the realization of *action lines* linking *process control devices* is represented by *information*

processing devices. Further information about *information processing devices* (e.g. related to system and field busses, stored-program controller systems, process and logging stations) and about *human-process communication devices* can be found in the literature (e.g. Polke 1994; Früh 2000).

All *instruments* are introduced during the design of *plants*. A *measuring instrument* or a *controlling instrument* belongs to both, *plant* and *process control system*. They are devices with two types of *system interfaces*, which are connected to *plant items* to be part of a *plant* and to *information processing devices* to be part of a *process control system* (Wilhelm 1996).

A *process control system* consists mainly of *information processing devices* and *process control devices*, with the latter including particularly *human-process communication devices*. There are different structures of *process control systems* depending on the linkage of the different devices and their respective implementation (not shown in Fig. 8.27): decentralized stored controller systems, integrated decentralized process monitoring and control systems, and intelligent central systems with "unintelligent" peripherals[87]. According to Wilhelm (1996), it is necessary to model the *process control system* explicitly in order to support the needs of control and software engineers, who are responsible for the configuration and maintenance of the system.

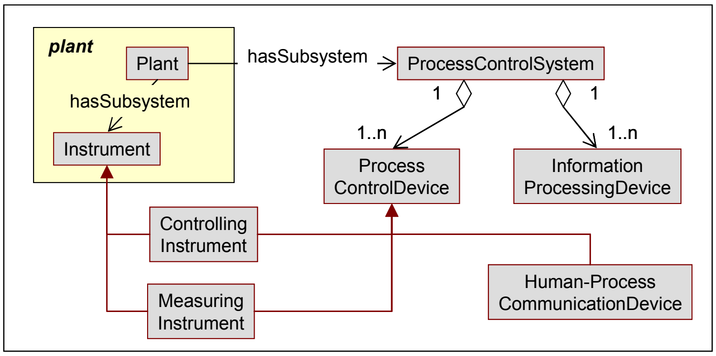

Fig. 8.27: *Process_control_system* and its relation to *plant*

The connectivity considerations in *process_control_system* may be realized by the principles provided by the ontology module *network_system* as illustrated in Fig. 8.28.

[87] Note that these structures are beyond the level of detail of the representation given in the module *process_control* (Sect. 8.2.2) for the time being.

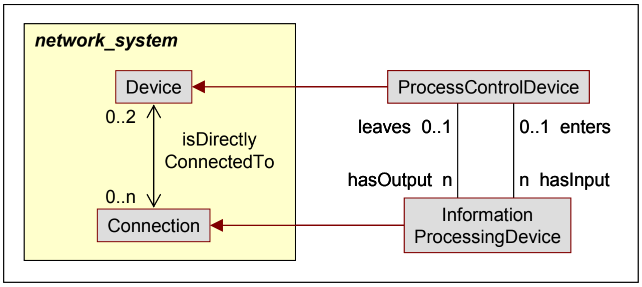

Fig. 8.28: Topological considerations applied in *process_control_system*[88]

8.3.2.1 Concept Descriptions

Individual concepts of the module *process_control_system* are defined below.

Class Descriptions

Controlling instrument
Controlling instrument represents all *instruments* that have a control function in a *plant* or a *process control system*.

Human-process communication device
Human-process communication device represents the hardware devices that implement the human-machine interface in *process control systems*.

Information processing device
Information processing device represents the hardware devices for information processing in *process control systems*.

Measuring instrument
Measuring instrument represents all *instruments* that have a measurement function in a *plant* or a *process control system*.

Process control device
Process control device subsumes all the *devices* in *process control systems*.

[88] Currently, only the connection to the plant item is modeled in more detail. However, the extension to interfaces of information processing devices might be done analogously without difficulty.

8.4 Plant Equipment

The realization of a *chemical process system* is extensively covered by the preceding modules on a conceptual level. However, additional insight may be gained from the perspectives of specific applications. Hence, the partial model **plant_equipment** covers exemplary specifications of frequently applied equipment. In particular, the classes *fixtures*, *apparatus* and *machines* introduced previously are now taken as a basis for a further specification to be presented in the modules *fixture*, *apparatus* and *machine.*

8.4.1 Fixture

The *fixture* module introduces some exemplary specializations of the class *fixture* defined in the *plant* module. Some frequently applied *fixtures* are listed in Fig. 8.29.
Note that this list is not intended to be exhaustive but rather constitutes a starting point for further completion. As an example *stirrer* and *jacket* are typically installed in vessels to form specialized equipment such as continuous stirred tank reactors or heated tanks. For the description of shell and tube heat exchangers the combination of *shells* and *tubes* are required. Furthermore, *trays* are an important integral part of tray columns. Some applications examples of are presented in Sect. 8.4.2.

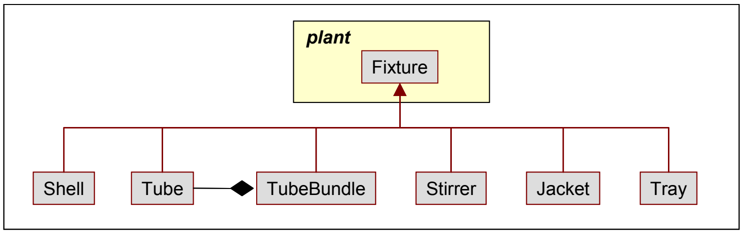

Fig. 8.29: Representation of the specializations of *fixture*

Often, properties of fictures are supposed to be specified for a more detailed description. These properties are linked with a fixture by special relations. Some examples include the heated length of a *tube,* or the hole diameter and the tray area of a *tray* (cf. Fig. 8.30).

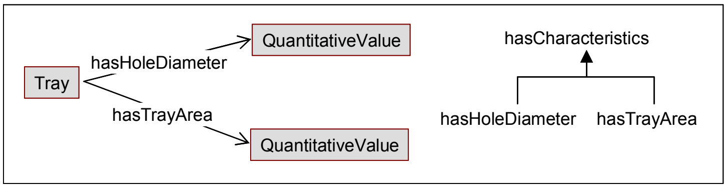

Fig. 8.30: Exemplary assignment of relevant properties of *fixtures*

For the description of the individual concepts of the module *fixture*, we refer to Wiesner et al. (2008a).

8.4.2 Apparatus

Basically, the module *apparatus* identifies some exemplary specializations of *apparatus*. In Fig. 8.31 some sample subclasses of *apparatus* are given.
This taxonomy has been based on DIN standards given for equipment of chemical plants (Graßmuck et al. 1994). Most of the presented apparatuses meet their particular function by means of the installation of an appropriate *fixture*. Apparatuses in OntoCAPE can be composed of either a simple *apparatus* and *fixtures* or a number of *fixtures*. As an example a *stirred tank* is a *vessel* with an embedded *stirrer* whereas a *shell tube apparatus* consists of a *shell* and a *tube bundle*.

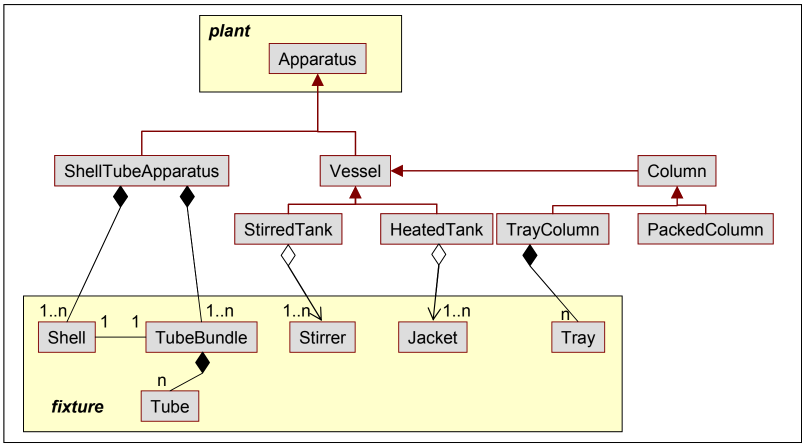

Fig. 8.31: Representation of the association between *apparatus* and *fixture*

The representation of multiple occurring *fixtures* of the same type in one *apparatus* is realized by means of the pattern provided by *multiset* defined in the meta ontology (cf. Sect. 4.5.2). A *tray column* can be considered as an example for such an *ap-*

paratus incorporating multiple *trays*. Fig. 8.32 illustrates an assignment of the *trays* to the corresponding *tray column* by means of the *multiset* directives.

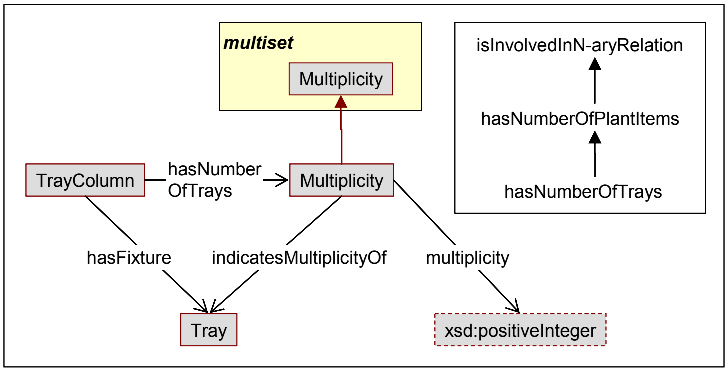

Fig. 8.32: Representation of assigning *multiplicity* in *apparatus*

For the description of the individual concepts of the module *apparatus*, we refer to Wiesner et al. (2008a).

8.4.3 Machine

Currently, the module *machine* comprises only one exemplary class namely *pump*. However, the module can be extended readily in the future to introduce other specific types of machines.
For the description of the individual concepts of the module *machine*, we refer to Wiesner et al. (2008a).

8.5 Process Control Equipment

In the area of process control engineering, some approaches on data models exist (see for example Polke (1994); Lauber (1996)). More recent activities initiated by consortiums like prolist (2009) or eClass (2009) have resulted in very detailed data models for the description of technical specifications of process control equipment as it is required for the procurement of such devices. However, in contrast to these very detailed models, this module intends to provide a conceptual description such that an easy integration of detailed models in the context of an entire chemical process system is attainable. Hence, detailed models such as prolist or eClass can be integrated as specializations into the model structure introduced in *process_control_system*[89]. In the following, we will briefly show how such an extension could be achieved using the class hierarchies for *measuring instrument* and *controlling instrument* given by Lauber (1996) and Polke (1994). A specialization with respect to some exemplarily chosen devices is demonstrated in this partial model in analogy to **plant_equipment**.

8.5.1 Measuring Instrument

Measuring instruments can be distinguished according to the type of process variable they detect. In Fig. 8.33 the major *measuring instruments* are given: *L-sensors* (level); *P-sensors* (pressure); *F-sensors* (flow rate), *T-sensors* (temperature), *Q-sensors* (some quality, like concentration, or conductivity). Note that this hierarchy is not intended to be complete.

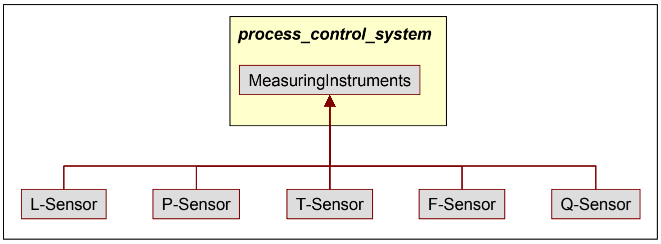

Fig. 8.33: Class diagram for some measuring device

For the different types of *measuring devices* further classifications can be given. This is shown in Fig. 8.34 for different *T-sensors*: *Pt100*, *expansion T-sensor*, *bimetal T-sensor*, *Seger cone*, *quartz crystal T-sensor*, *thermocouple* and *pyrometer* are different types of *measuring devices* for measuring temperatures.

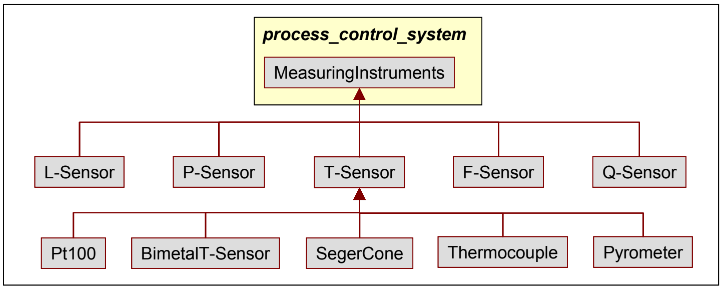

Fig. 8.34: Class diagram for some further specialization of T-Sensor

[89] Concepts about the integration of product data models have been reported by e.g. Bayer et al. (2000).

Furthermore, *measuring devices* can be distinguished according to their application and design features (e.g. online/offline, product property/process property, range, and manufacturer). This can be done by the introduction of corresponding attributes (not shown in Fig. 8.34).
For the description of the individual concepts of the module *measuring_instrument*, we refer to Wiesner et al. (2008a).

8.5.2 Controlling Instrument

Controlling instruments are first classified according to their mode of operation: The classification distinguishes whether a controlling instrument works mechanically or electrically. Hence, the classes *mechanical controlling instrument* and *electrical controlling instrument* are introduced (cf. Fig. 8.35).

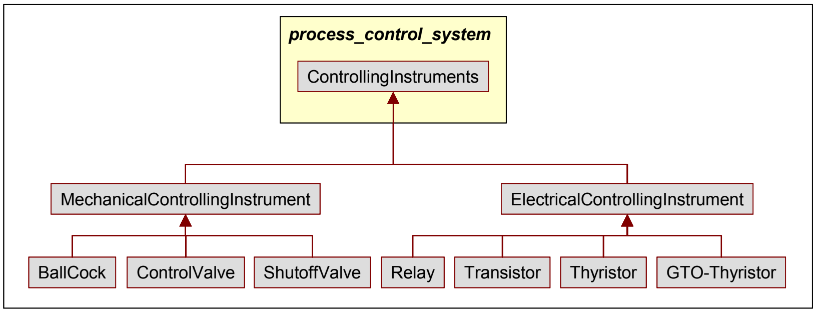

Fig. 8.35: Class diagram for some controlling instrument

Mechanical controlling instruments can be classified further into *control valves*, *screw conveyors*, *ball cocks*, and *shutoff valves,* which can be used to control material streams. *Relays*, *transistors*, and different types of *thyristors* can be used to control electrical current. Similar to the *measuring instruments*, con*trolling instruments* can also be characterized by application and design features.
For the description of the individual concepts of the module *controlling_instrument*, we refer to Wiesner et al. (2008a).

8.6 CPS Behavior

The preceding sections have focused on the **function** and the **realization** *aspects* of the *chemical process system*. The third major *aspect* to be considered is the **behavior** of a *chemical process system*. This aspect is represented in the partial model **CPS_behavior**. Currently, this partial model comprises only the module *behavior*.

In general, the behavior of any *system* may be described by means of mathematical models. This, however, is not the task of this partial model, but of the partial model **mathematical_model** (cf. Sect. 9.1). Instead, the description of *behavior*, as given here, is based on a phenomenological perspective. That is, this module mainly defines the chemical engineering concepts for directly representing *physicochemical phenomena*, which enable a qualitative description of the *system behavior* (cf. Sect. 5.3.3). In addition, *system behavior* can be quantitatively described by the explicit indication of property-value pairs (i.e., by means of a data-based description). To this end, appropriate *properties* will be introduced. By linking these *properties* to the corresponding *physicochemical phenomena*, a relation between the qualitative and the quantitative description can be established (cf. Fig. 5.30 in Sect. 5.3.5).
The description of material behavior, as it is considered in OntoCAPE, includes all material characteristics that depend on the shape, size, or amount of a particular occurrence of a material. Consequently, *extensive properties*[90] as well as the distribution of *intensive properties* in space and the distribution of any *physical quantity* in time are associated with material behavior. Such properties essentially describe the behavior of material demonstrated in a concrete spatio-temporal setting, such as being processed in a manufacturing process or acting as a construction material. Those data are usually considered as properties of the respective application or usage of the material (e.g., as properties of a chemical process), but not as properties of the material itself (a more detailed discussion of this issue can be found in Sect. 7.1.1).
It should be noted that, for the time being, the partial model **CPS_behavior** is confined to describing the physical behavior of processed matter. While this addresses the most important aspect of chemical system behavior, it neglects certain other aspects, such as the behavior of the control system.

8.6.1 Behavior

The ontology module *behavior* is based on the data model VEDA (Marquardt 1992b; 1995) and classical textbooks about reaction kinetics and transport phenomena (e.g. Bird et al. 2001; Wesselingh and Krishna 2000; Smith 1981; Froment and Bischoff 1990).
This section is structured as follows: First, the high-level concepts are defined, and their mereological (de)composition is discussed. Next, the topological considerations are depicted. Subsequently, the phenomenon-related concepts are introduced.

[90] An extensive property is a property that depends on the system size or the amount of material in the system.

8.6.1.1 High-Level Concepts

The module *behavio*r holds two major classes: *material amount* and *material amount connection* (cf. Fig. 8.36).

A *material amount* is an *aspect system* that represents the occurrence of matter in a concrete spatiotemporal setting – a typical example of a *material amount* would be the liquid holdup in a vessel. The *properties* of a *material amount* include all material characteristics that depend on the shape, size, or amount of a particular occurrence of a material, such as weight, volume, enthalpy, and so on. Consequently, *extensive properties* as well as the distribution of *intensive properties* in space and time are attributable to *material amount*. *Material amount* thus differs from a *phase system*, which does not cover any geometry-dependent and amount-dependent property-Consequently, *material amount* refers to a chunk of material in a process context in contrast to a phase system which takes a perspective independent of a particular process context (cf. Sect. 7.1.1).

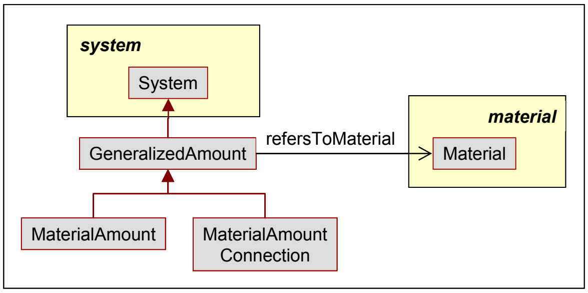

Fig. 8.36: Specializations of *generalized amount*

On the other hand, the fundamental characteristics of a material – e.g., its composition, molecular structure, or thermodynamic state – are not considered to be *properties* of a *material amount*. For their specification, a *generalized amount* refers to the corresponding *material* (Sect. 7.1).

A *material amount connection* represents a connection between two *material amounts* to refer to the transfer of matter and/or energy. Typical examples of *material amount connection* include a pipe that transports a fluid between two vessels or a wall between a hot and a cold reservoir that enables the transfer of heat energy by heat conduction.

For practical purposes, the class *generalized amount* is introduced as a superclass of *material amount* and *material amount connection*. It simply represents a generalization of *material amount* and *material amount connection*.

8.6.1.2 Aggregation and Decomposition

For the decomposition of *material amount (connections)*, all patterns defined for the decomposition of *systems* are applicable (cf. Sect. 5.1.3). Thus, a *material amount* can be composed of other (connected) *material amounts*. Also, a *material amount connection* can be composed of other *material amount connections* mediated by *material amount(s)*.

By convention, a *material amount connection* at the lowest level of decomposition (i.e., a *material amount connection* that qualifies as an *elementary system*, cf. Sect 5.1.4) is supposed to have a negligible volume and hence may not display a hold-up for *extensive quantities* (Marquardt 1992b; 1995). This is best illustrated by the example of a pipe connecting two process units: If the amount of material enclosed in the pipe is not considered to be relevant in the respective modeling context, the pipe is treated as an elementary *material amount connection*. If, on the other hand, the volume of material is significant for the problem at hand (e.g., because the flow delay or the spatial variation of the fluid properties along the pipe needs to be taken into account), the pipe must be treated as a composite *material amount connection*, as shown in Fig. 8.37. It is composed of a *material amount* representing the matter in the tube, and two adjacent *material amount connections*.

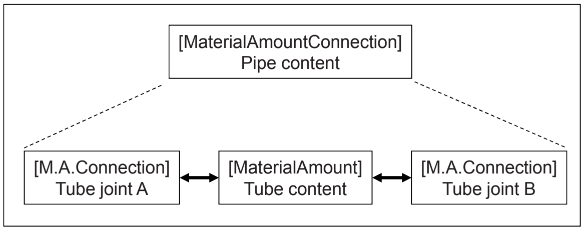

Fig. 8.37: Decomposition example for *material amount connection*

8.6.1.3 Representation of Granular and Dispersed Materials

The *material amount* may also be described, specialized, and categorized according to its dispersion state (cf. Fig. 8.38). Note that the concept of *phase system* (cf. Sect. 7.3) does not suggest anything about the dispersion state of material. In other words, it does not affect the *properties* of a *phase system* whether the material considered occupies an extended continuous region or whether it is dispersed. Instead, the concept of dispersion state is associated with *material amount*.

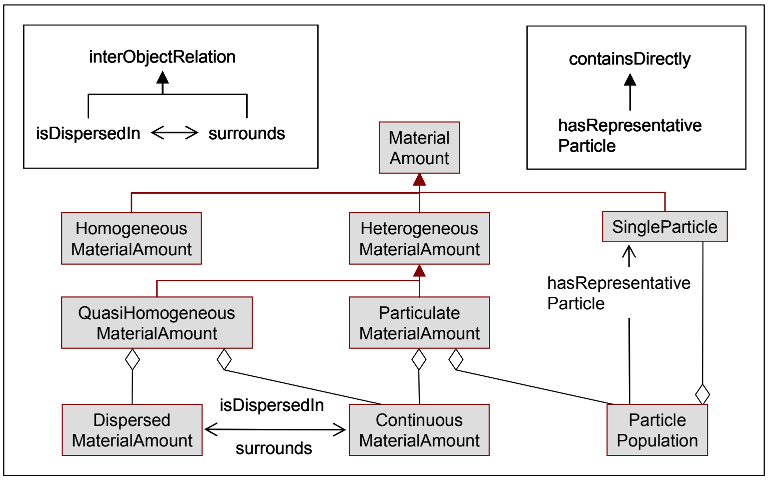
interObjectRelation
isDispersedIn
surrounds
containsDirectly
hasRepresentative Particle
Material Amount
Homogeneous MaterialAmount
Heterogeneous MaterialAmount
SingleParticle
hasRepresentative Particle
QuasiHomogeneous MaterialAmount
Particulate MaterialAmount
isDispersedIn
surrounds
Dispersed MaterialAmount
Continuous MaterialAmount
Particle Population

Fig. 8.38: Specialization categorization for *material amount*

To that end, in addition to a *homogeneous material amount*, two types of *heterogeneous material amounts* are distinguished: the *quasi-homogeneous material amount* and the *particulate material amount*.
The term 'quasi-homogeneous' refers to a *heterogeneous material amount* with the *dispersed material amount* assumed to be conceptually "dissolved" in the *continuous material amount*. Transfer resistances are neglected, and the *dispersed material amount* is treated exactly like any other chemical substance of the *continuous material amount*. The *physical quantities* (cf. Sect. 5.1.11) are suitably chosen averages of those of the *continuous* and the *dispersed material amounts*. As a prototype one may think of a simple abstraction of the contents of a three-phase slurry reactor, where the gaseous reactant and the solid catalyst are treated as "dissolved" in the continuous liquid phase.
Furthermore, two relations are introduced, as stated subsequently:

– A *continuous material amount* surrounds one or more *dispersed material amounts*. For example, consider the case of a slurry bubble-column reactor (e.g., Smith 1981): Gas bubbles (*dispersed material amount* gas) pass through a liquid (the *continuous material amount*), in which a number of catalyst particles (*dispersed material amount* solid) are suspended.
– A *dispersed material amount* isDispersedIn one ore more *continuous material amounts*. Take the example of a trickle-bed reactor (cf. Smith 1981), in which a catalyst particle (the *dispersed material amount*) might be covered partially by liquid (*continuous material amount* number one) and partially by gas (*continuous material amount* number two).

Note that, though not shown in Fig. 8.38 for the sake of simplicity, *dispersed material amount*, *continuous material amount* and *particle population* are also subclasses of *material amount*.

In contrast to *quasi-homogeneous material amount*, a *particulate material amount* refers to a heterogeneous system, where the dispersed material is described by a *particle population.* Such bulk material consists of a (possibly uncounted) number of *single particles*, which – in their entirety – can be characterized by means of property distributions and population balances (Ramkrishna 1985). The term 'particle' does not exclusively refer to solid particles, but includes vapor bubbles and liquid droplets, as well. In fact, a *heterogeneous material amounts* may consist of any combination of solid particles, liquid droplets, and/or vapor bubbles.
As an example for a *particulate material amount*, one can consider some crystalline particles that precipitate from a solution. Then the solution can be represented as a *continuous material amount*, and the crystalline particles can be described as a *particle population* characterized by a particle size distribution.
In most cases, it would probably not be of interest to describe all the individual *single particles* in a *particle population*. Instead, the particles are represented by means of a representative *single particle* with a few carefully selected properties that is part of a *particle population*. The relation hasRepresentativeParticle is introduced to formalize this concept. Usually geometric properties (e.g., characteristic size or volume) or other *extensive properties* (e.g., mass) or *intensive thermodynamic properties* (e.g., temperature or concentration) are selected.
For the specification of a *particulate material amount*, it is also of interest how a certain characteristic varies across the individual *single particles*. To this end, the class *fractional amount* is introduced. A *fractional amount* quantifies the amount of particles that show a certain characteristic. By means of the *fractional amount*, it can be said that a certain *single particle* represents a fraction of e.g. 50 % of all particles of the *particle population*. As an example, a distribution of *single particles* according to their volume is given.

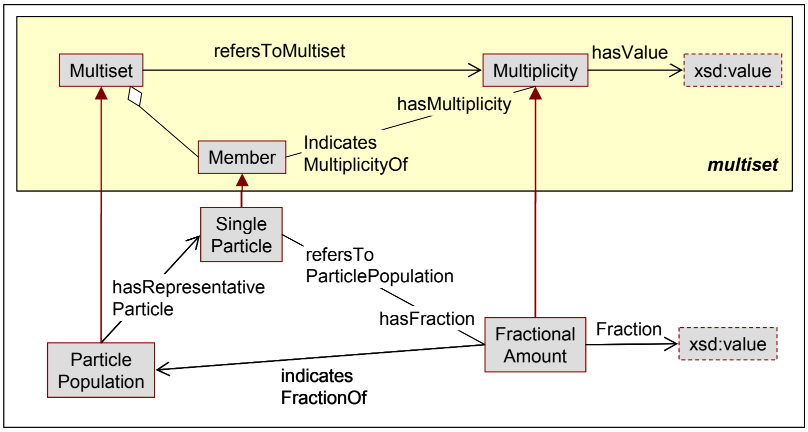

Fig. 8.39: Representation of a particle distribution of *particle population*

Fig. 8.39 illustrates, the assignment of a *fractional amount* to a *single particle*. Note that the *particle population amount* is considered to be a specialization of a *multiset*, and the fractional amount is derived from *multiplicity* (cf. Sect. 4.5.2).

8.6.1.4 Types of Material Amount Connection

There are three distinctive subclasses of *material amount connection*, namely *valve connection*, *film connection*, and *heat radiation connection* (cf. Fig. 8.40)[91]. With respect to mass transfer, convection and diffusion are the two major mechanisms to be considered. For heat transfer, in addition to heat convection and conduction corresponding to the stated mass transport mechanisms, heat radiation has to be taken into account, as well. Hence, the class *heat radiation connection* is introduced.

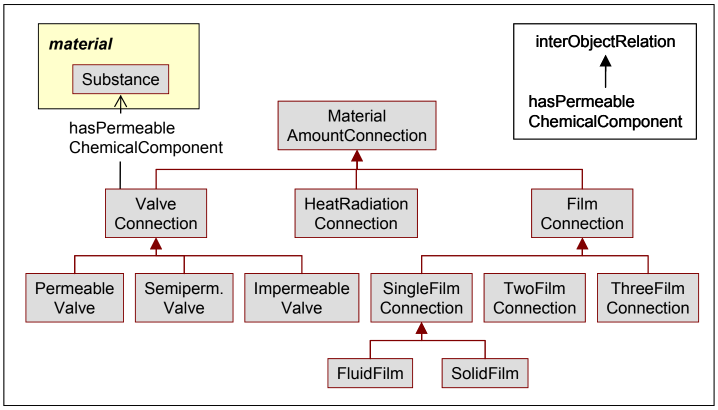

Fig. 8.40: Mereological considerations for *material amount connection*

The transport through *film connections* is dominated by conduction and diffusion processes, whereas *valve connections* represent the transport dominated by convection. A film represents a boundary layer occurring either in a solid or in a fluid phase; accordingly, the classes *fluid film* and *solid film* are introduced. Frequently, two or even three films are adjacent to each other at fluid-fluid or fluid-solid-fluid interfaces; this is accounted for by the classes *two-film connection* and *three-film connection*. *Film connections* include heat and mass transfer, such as the simultaneous heat and multi-component mass transfer at a liquid-vapor interface (e.g. Krishna and Taylor 1993).
The three types of *valve connections* displayed in Fig. 8.40 distinguish three different types of convective mass transport. For ease of modeling, three different states of *permeability* are assumed, namely **permeable**, **impermeable** and **semi-permeable.**

[91] The taxonomy shown is based on the work of Marquardt (1996) which itself relies on the conceptualization of Haase (1990).

In terms of connectivity, the states of **impermeable** and **permeable** represent the cases where either no material transport occurs at all or the entire material is transported. As an example, a pipeline which conveys all species in a mixture may be described by a *permeable valve*, whereas an *impermeable valve* typifies a solid wall that blocks any mass transfer. In case of a *semi-permeable valve*, e.g. a selective porous membrane, only some species in the mixture may pass. These selected permeable components may be identified and further specified by referring to a *substance* via the relation hasPermeableChemicalComponent. *Valve connections* and *film connections* are always permeable for heat.
In order to further clarify the above concepts, let us consider the wall of a tube in a shell and tube heat exchanger. If the heat capacity of the wall is negligible, the connection between the shell and tube side can be modeled as a *three-film connection*, which accounts for the heat transfer across the tube wall and the fluid boundary layers on both sides. On the other hand, if the heat capacity of the wall is significant in a certain modeling context, the connection between shell and tube side must be represented by a composite *material amount connection* The tube wall is then viewed as a *homogeneous material amount*, whereas the boundary layers on either side of the wall are treated as *material amount connections* of type *fluid film*.

8.6.1.5 Connectivity

In the following an overview on the concepts used for interconnecting *material amounts* and *material amount connections* is given (cf. Fig. 8.41). Again the principles defined in *network system* (cf. Sect. 5.2) are applicable. Thus, the *material amount* corresponds to a *device*, whereas *material amount connection* represents the function of *connection*. To represent a process more precisely, a *directed connection* may be introduced to explicitly indicate the direction of any exchange between two *material amounts*. Furthermore, *ports* and *connection points* may be introduced in order to better characterize the location (and possibly geometry) of the points for connection.

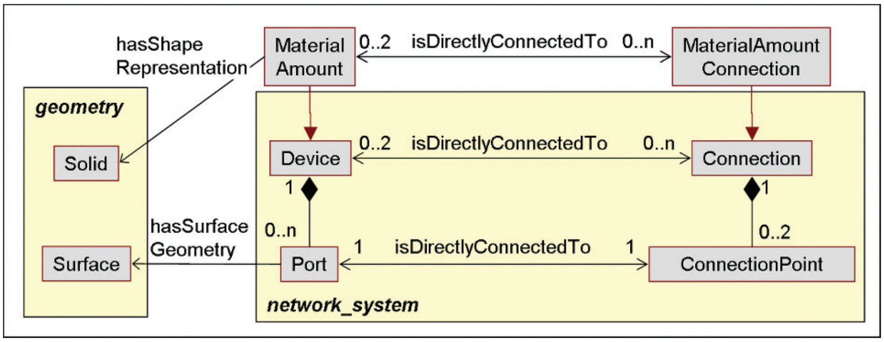

Fig. 8.41: Modeling of connectivity

Typically, the geometry of the *material amount* is abstracted by means of a simple geometrical shape, which can be represented through the concept of *solid* (cf. module *geometry*, Sect. 6.5, as well as Fig. 8.41). Additionally, the concept of a *surface* (cf. Sect. 6.5) may be used to denote the area of a *port* which may be associated with a specific *material amount connection*. In OntoCAPE, currently only simple geometries are defined (cf. Sect. 6.5); extensions covering complex geometric shapes would be required to model the geometric details of a *material amount* in an equipment of irregular geometry.
Moreover, a detailed account of a *port* of a *material amount* can be important in the case of a distributed parameter system. For example, the spatial distribution of nutrients in a non-ideally mixed, fed-batch fermenter can be sensitive to the concrete location of the feed point within the vessel. Furthermore, if the opening of the feeding pipe to the fermenter cannot be treated as a singular point, a detailed characterization of the feed zone, including the shape and the size of the opening, has to be taken into account. This essentially requires a characterization of the positioning of the relevant geometric objects (these are, in this example, the geometries of the *material amount* in the fermenter and the *port* corresponding to the feed pipe opening) within a particular *spatial coordinate system*. In principle, this could be achieved by specifying the orientation (via an appropriate vector quantity; cf. Sect. 6.5) and the location (via the coordinates of a characteristic point, e.g. the center of a sphere) of each geometric object involved, all against the same *spatial coordinate system*.

8.6.1.6 Phenomenological Description

As mentioned above, the *behavior* module describes a phenomenon-based perspective of behavior; the occurrence of the *phenomena*, in turn, influences the *properties* of a certain *material amount* or *material amount connection*. The idea is illustrated in Fig. 8.42.
The change of material properties due to the time-variant influence of a particular *phenomenon* is of special interest within the scope of *behavior*. This module is derived based on the ontological concept of *system behavior* previously defined in *technical system*. A *material amount* may undergo a certain *material amount phenomenon* and may be characterized by a *physical quantity*, which in turn may be influenced by *material amount phenomena*. The same pattern is applicable to *material amount connection*, resulting in a hasPhenomenon relation to *material amount connection phenomenon*. Clearly, from an engineering point of view, one is primarily interested in the changes of the *physical quantities* which describe a certain process instead of the underlying *phenomenon*. Hence, one may navigate from a particular *material amount (connection) phenomenon* to the corresponding *physical quantities* via the relation isInfluencedBy.

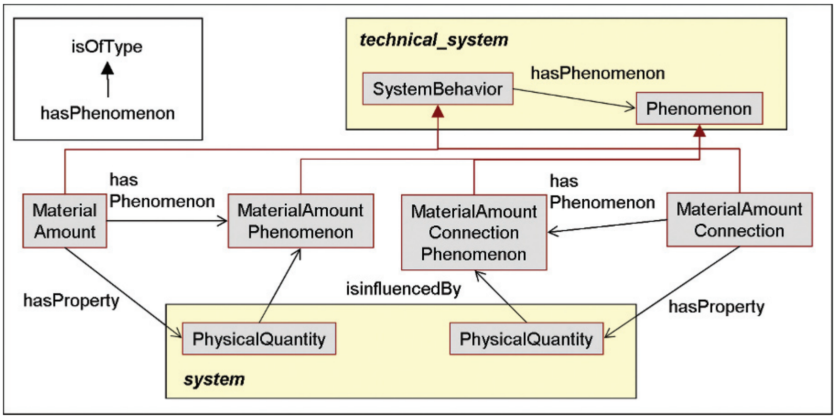

Fig. 8.42: Overview on the assignment of *phenomenon* and *properties* in *behavior*

A further specialization of the *material amount (connection) phenomena* is of interest since the concrete effect on the *material amount* may vary significantly depending on the nature of the particular *phenomenon* at hand[92]. Basically, all *phenomena* considered in *behavior* may be subsumed as *physicochemical phenomena*. Moreover, as already introduced earlier, a further classification into *material amount phenomena* and *material amount connection phenomena* is established. A graphical representation is given in Fig. 8.43.

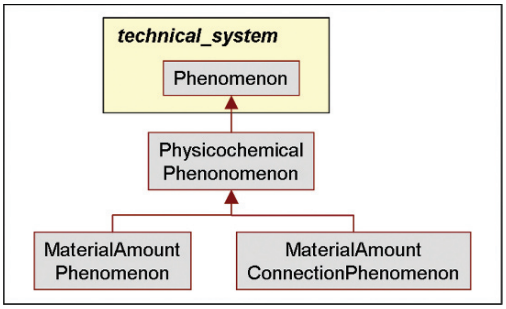
technical_system
Phenomenon
Physicochemical Phenonomenon
MaterialAmount Phenomenon
MaterialAmount ConnectionPhenomenon

Fig. 8.43: Representation of the specialization of *phenomena* employed in *behavior*

First, we consider the classification of the *phenomena* that can occur within a *material amount* (cf. Fig. 8.44-Fig. 8.45).
On the highest level, as represented in Fig. 8.44, there are (i) *phenomena* associated with chemical reactions (*chemical reaction phenomenon*), (ii) accumulation of *extensive properties* (*accumulation*) as well as (iii) *flow patterns*, which essentially give indications about convective transport phenomena.

[92] Within the scope of OntoCAPE 2.0 the description and classification provided for phenomena is kept at a very generic level; the essential directives are presented to allow for a further elaboration as required.

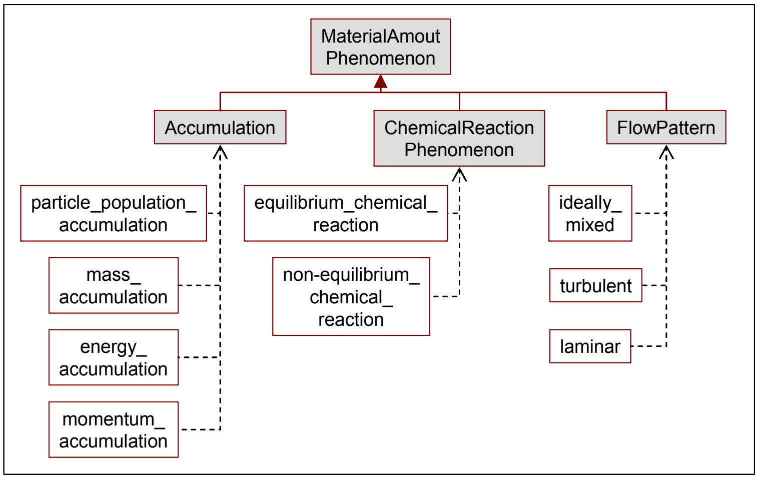

Fig. 8.44: Representation of some *material amount phenomena*

Furthermore, there are (iv) *particle phenomena*, (v) *molecular transport phenomena*, and (vi) *physical equilibrium phenomena* (cf. Fig. 8.45).
With regard to (vi), it should be noted that, although the *behavior* module does not intend to describe the intensive, equilibrium properties of material (cf. Sect. 7.1.1), this does not exclude the possibility to consider that a *material amount* reaches equilibrium state at some point in time – it is simply one of the possible states in the behavior of that *material amount*.

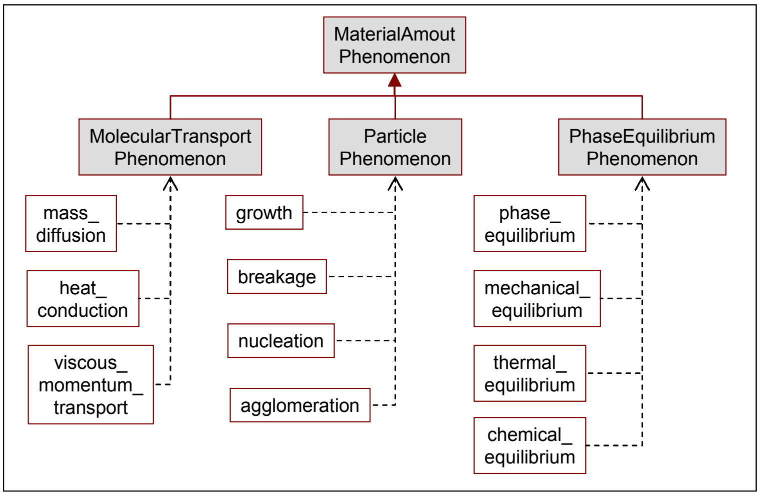

Fig. 8.45: Representation of some *material amount phenomena*

Additionally, the major individuals for a further refinement of the *material amount phenomenon* are introduced in Fig. 8.44 and Fig. 8.45 as well. Note that the proposed individuals are not considered to be exhaustive such that an adequate adaptation and expansion is recommended.

It is possible that more than one of the aforementioned phenomena occur in a certain *material amount* – either simultaneously or sequentially. However, some of the aforementioned phenomena are mutually exclusive; thus, to guarantee feasibility, these phenomena must not be modeled concurrently. To this end, four axiomatic classes are introduced to ensure a consistent use of these features. These classes are *ideally mixed material amount*, *material amount with molecular transport phenomenon*, *material amount in phase equilibrium,* and *material amount with spatially distributed intensive property*.

– An *ideally mixed material amount* is defined as a *material amount* in which the phenomenon of **ideally_mixed** prevails.
– A *material amount with molecular transport phenomenon* is defined as a *material amount* in which some kind of *molecular transport phenomenon* occurs.
– A *material amount in phase equilibrium* is defined as a *material amount* that is governed by the phenomenon of **phase_equilibrium**.
– A *material amount with spatially distributed intensive property* has some *intensive properties* that are distributed on a certain spatial domain.

As an *ideally mixed material amount* has spatially uniform properties, it cannot have any *properties* that indicate a distribution over a spatial domain; nor must it be associated with *molecular transport phenomena*, which require such a spatial distribution. Therefore, the class *ideally mixed material amount* is declared to be disjoint from the classes *material amount with spatially distributed intensive property* and *material amount with molecular transport phenomenon*.

Similarly, a *material amount in phase equilibrium* must neither have *properties* nor *phenomena* indicating a spatial distribution or some kind of molecular transport. Therefore, the class *material amount in phase equilibrium* is declared to be disjoint from the classes *material amount with spatially distributed intensive property* and *material amount with molecular transport phenomenon*.

The definition of *material amount connection phenomena* has been influenced by the thermodynamics of irreversible processes (e.g., Haase 1990). According to this theory, the flux of any extensive quantity (like mass, heat, etc.) is caused by one or more generalized forces, which are determined by the differences of some *physical quantity* in the adjacent material amount (such as pressure or temperature). In addition, surface reactions, surface diffusion, and other surface phenomena may occur in the *material amount connection*.

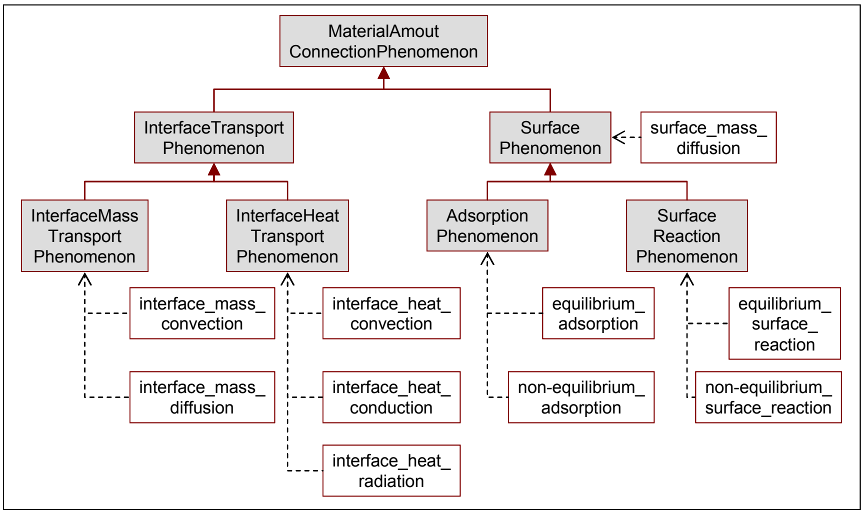
MaterialAmout ConnectionPhenomenon
InterfaceTransport Phenomenon
Surface Phenomenon
surface_mass_ diffusion
InterfaceMass Transport Phenomenon
InterfaceHeat Transport Phenomenon
Adsorption Phenomenon
Surface Reaction Phenomenon
interface_mass_ convection
interface_heat_ convection
equilibrium_ adsorption
equilibrium_ surface_ reaction
interface_mass_ diffusion
interface_heat_ conduction
non-equilibrium_ adsorption
non-equilibrium_ surface_reaction
interface_heat_ radiation

Fig. 8.46: Representation of *material amount connection phenomena*

Fig. 8.46 shows the classification of *material amount connection phenomena* as provided by OntoCAPE. *Surface phenomenon* is distinguished from *interface molecular transport phenomenon*. For the former, two specializations are introduced, namely *adsorption phenomenon* and *surface reaction phenomenon*. Again, some exemplary individuals are introduced to represent common phenomena.

8.6.1.7 Properties of Material Amount

Following the presentation of the *material amount (connection) phenomena*, the high-level classes of the resulting *properties* are introduced. The main focus here is on the description of geometry- and amount-specific *properties* of materials.
There are six major groups of *physical quantities*[93] to be distinguished for a behavioral description within this context (Marquardt 1996; Bogusch 2001). This comprises (i) *generalized fluxes*; (ii) *extensive properties*; (iii) geometry-dependent *intensive properties*[94]; (iv) *phenomenological coefficients*; (v) the spatial gradients of some *intensive thermodynamic state variables*, which are subsumed by the class *state variable gradients*; and finally (vi) spatial *velocity gradients*.

[93] Note that a more fine-grained perspective of the properties might be necessary for particular applications and that further specialization is recommended in such cases.

[94] Examples of geometry-dependent intensive properties would be the rates of adsorption and reaction that occur on a solid surface with specific porosity, or the growth rate of a face of a crystal that possesses a specific shape.

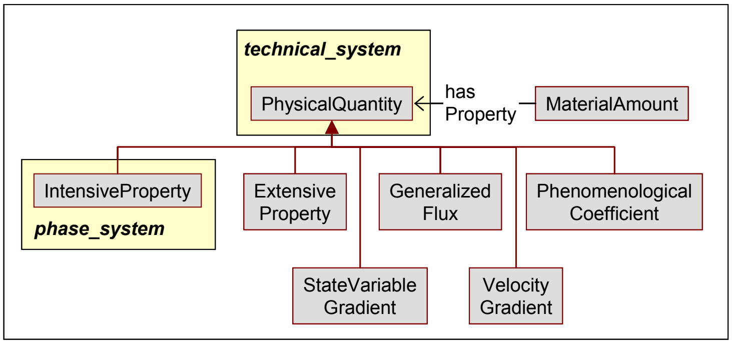

Fig. 8.47: Overview on *material amount* related *physical quantities*

Generalized fluxes (cf. Fig. 8.47) include (i) the transport within a phase (*intra-phase transport*), (ii) the transport across a phase boundary (*exchange*), (iii) the sources and sinks caused by chemical reaction or some external potential field (*source*), and finally (iv) the time-dependent holdup of material amount in a system (*holdup variation*). Consequently, any *properties* that describe the rates of chemical reaction or transport phenomena or the spatial/temporal accumulation of physical quantities are specializations of *generalized fluxes*. Some examples are shown in Fig. 8.48 such as *convective transport rate* or *conductive transport rate* for (i), *convective exchange* or *radiation exchange* for (ii), and *reaction rate* for (iii).

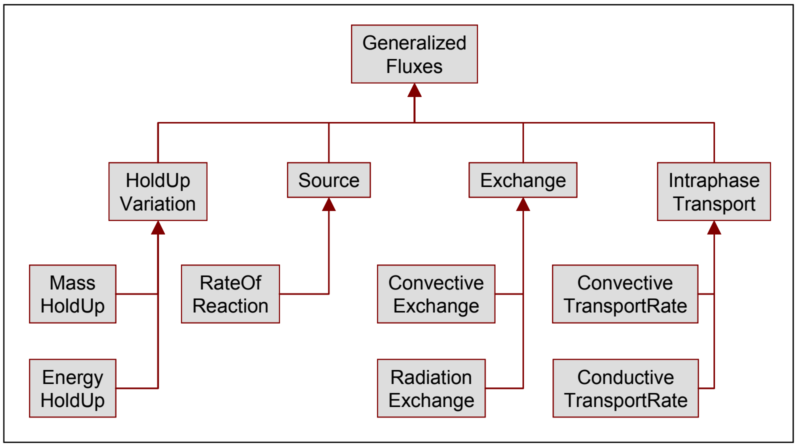

Fig. 8.48: *Generalized fluxes*

Phenomenological coefficients are distinguished into (i) *inter-phase transport coefficient*, (ii) *dynamic viscosity of non-Newtonian fluids,* and (iii) *reaction rate coefficients* (cf. Fig. 8.49). The second class is included in this partial model rather than the **material** partial model, because it addresses the non-equilibrium nature of non-Newtonian fluids. Some typical refinements of the above classes are introduced, such as *mass* *transfer coefficient*, *heat transfer resistance* for (i). Note that, not shown in Fig. 8.48 or Fig. 8.49, *rate of reaction* and *reaction rate coefficient* refer to *single reaction* (defined in the *reaction_mechanism* module; cf. Sect. 7.2.8) to denote which chemical reaction these two properties describe in a particular circumstance.

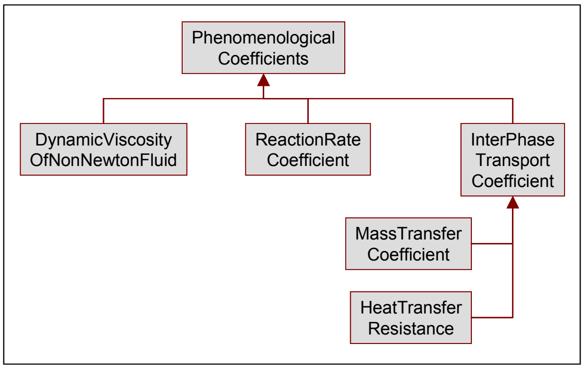
Phenomenological Coefficients
DynamicViscosity OfNonNewtonFluid
ReactionRate Coefficient
InterPhase Transport Coefficient
MassTransfer Coefficient
HeatTransfer Resistance

Fig. 8.49: *Phenomenological coefficients*

Finally, the class *convective transport rate* may be further refined (cf. Fig. 8.50). To that end, the classes (i) *convective mass flowrate*, (ii) *convective energy flowrate*, (iii) *convective momentum flowrate,* (iv) *convective mass flux*, (v) *convective energy flux*, and (vi) *convective momentum flux* are introduced.

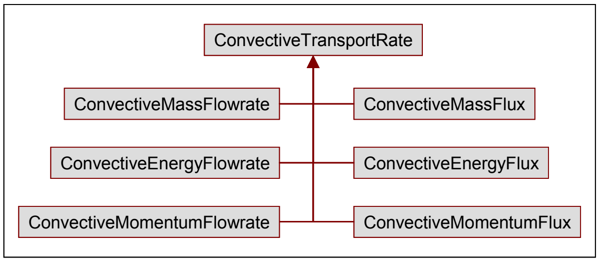

Fig. 8.50: *Convective transport*

8.6.1.8 Application Examples

In the following, two application examples are discussed: The first example illustrates the construction of a complex behavior model from *material amounts* and *material amount connections* of different types. The second focuses on the interrelations between the modules *behavior* and *phase_system*. To keep the examples easy to understand, it was decided to separate the two topics, although the modeling can certainly be combined if required.

The first example assumes a chemical reactor holding a liquid product, with a deactivated catalyst distributed in the liquid product. For catalyst reactivation, a separation of the catalyst from the liquid by means of filtration is assumed. For the sake of simplicity, this filtration is assumed to take place in the same reactor which holds the reaction product, such that reaction product denotes the entire material amount considered in the chemical reactor. The resulting pure catalyst stream is fed through a pipe to the reactivation process step. The catalyst reactivation is then achieved in a fluidized bed reactor. The fluidized bed itself consists of the dispersed catalyst in a gaseous medium. Fig. 8.51 gives an overview on the assumed process.

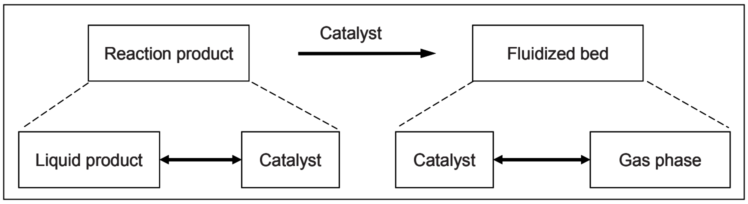

Fig. 8.51: Example 1: Catalyst reactivation after a reaction

The representation corresponding to the principles defined in *behavior* yields the diagram shown in Fig. 8.52. First of all, the **reaction product** is represented as an instance of *particulate material amount,* the **fluidized bed** as an instance of *quasi-homogeneous material amount,* and the **catalyst stream** (representing the flow through the pipe) is depicted as an instance of *valve connection*. The connectivity on this level of abstraction is realized by means of a subclass of *directed connections* indicating the proper sequence of process steps.

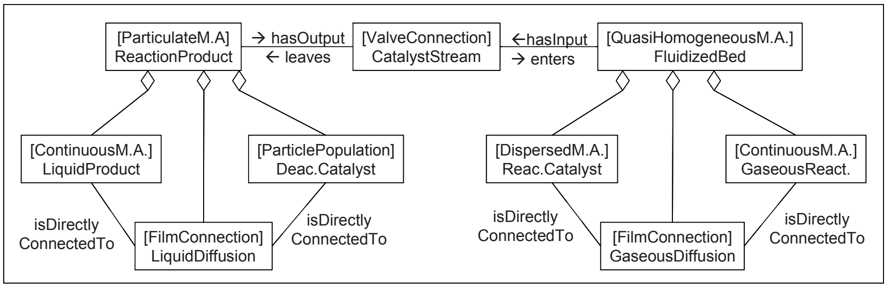
[ParticulateM.A] ReactionProduct
→ hasOutput
← leaves
[ValveConnection] CatalystStream
←hasInput
→ enters
[QuasiHomogeneousM.A.] FluidizedBed
[ContinuousM.A.] LiquidProduct
[ParticlePopulation] Deac.Catalyst
[DispersedM.A.] Reac.Catalyst
[ContinuousM.A.] GaseousReact.
isDirectly ConnectedTo
[FilmConnection] LiquidDiffusion
isDirectly ConnectedTo
isDirectly ConnectedTo
[FilmConnection] GaseousDiffusion
isDirectly ConnectedTo

Fig. 8.52: Example 1: Representation in OntoCAPE

Fig. 8.53 displays an extension of the application example: It demonstrates how to further characterize a *material amount* by concepts from the **material** partial model. In this particular case, we demonstrate how the catalyst is identified as a *chemical species* of type **silver oxide** via the relation refersToMaterial.

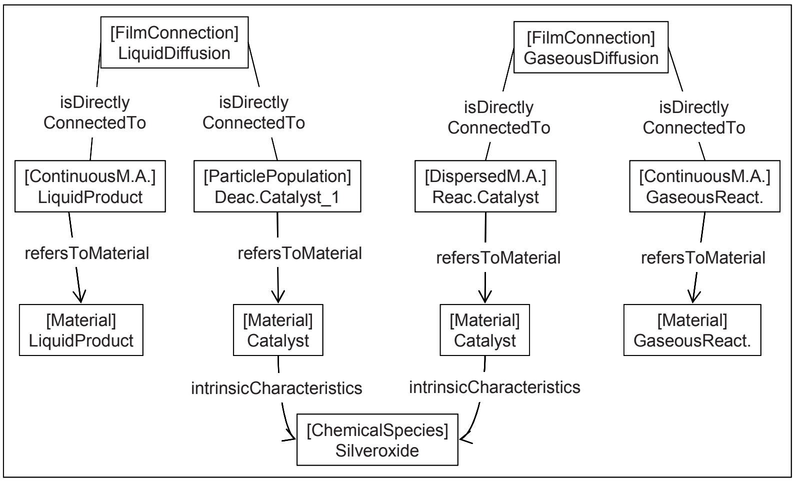

Fig. 8.53: Example 1: The catalyst is further characterized as a *chemical species* of type **silver oxide**

The second example considers the behavior in a jacket-cooled CSTR (cf. Fig. 8.54). Two instances of *material amount* are introduced, namely the **reactor content** representing the material inside the reactor vessel, and the **jacket content** representing the material inside the jacket. The **reactor content** again consists of two *homogeneous material amounts* - the **liquid content** and the **vapor content**.

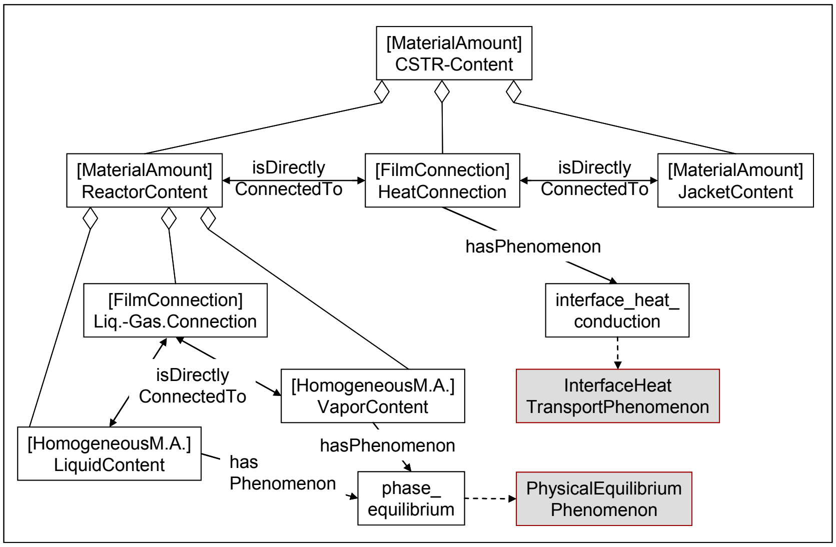
[MaterialAmount] CSTR-Content
[MaterialAmount] ReactorContent
isDirectly ConnectedTo
[FilmConnection] HeatConnection
isDirectly ConnectedTo
[MaterialAmount] JacketContent
hasPhenomenon
interface_heat_ conduction
InterfaceHeat TransportPhenomenon
[FilmConnection] Liq.-Gas.Connection
isDirectly ConnectedTo
[HomogeneousM.A.] VaporContent
[HomogeneousM.A.] LiquidContent
hasPhenomenon
has Phenomenon
phase_ equilibrium
PhysicalEquilibrium Phenomenon

Fig. 8.54:Example 2: Material behavior in a jacket-cooled CSTR

The connections between (i) the **liquid content** and the **vapor content** and (ii) the **reactor content** and the **jacket content** are realized by means of *film connections* of type *three film connection*. The heat transfer between jacket and vessel may be described by the *phenomenon* of **interface_heat_conduction**, i.e. heat is led from the vessel to the jacket.
In the following we focus on the representation of the reactor content description. However, *material amount phenomena* and *properties* in the jacket may be described analogously in the way the **reactor content** is modeled.
The **liquid content** and the **vapor content** are assumed to be in phase equilibrium; this is modeled by linking the two individuals to the *material amount phenomenon* of **phase_equilibrium**. Moreover, it is assumed that a **non-equilibrium_chemical_reaction** phenomenon occurs in the **liquid content**; namely the chemical reaction A + B → C. Thus, both the **liquid content** and the **vapor content** involve a mixture of chemical components A, B, and C (cf. Fig. 8.57). Note that a detailed representation of the chemical reaction is not shown in Fig. 8.57. However for an application example of chemical reaction modeling, we refer to Sect. 7.2.8.1.
The state of aggregation of the **liquid content** and the **vapor content** can be specified by using concepts from the partial model **material** (cf. Fig. 8.55) : First, a relation to the abstract *material* is established; second, the *materials* are characterized as two *single phases* of the **liquid** and the **gaseous** *state of aggregation*.

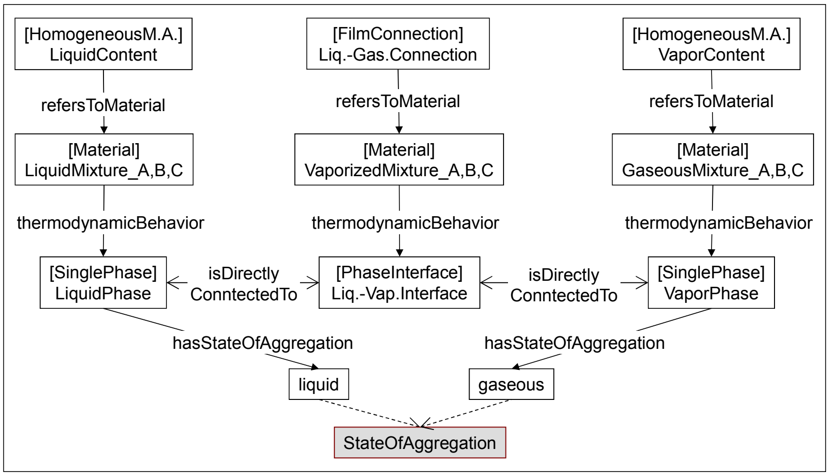

Fig. 8.55: Example 2: Interrelations between concepts from *behavior* and *phase_system*

The connections to the (not yet specified) environment of the **reactor content** (two input *valve connections* and one output *valve connection*) and the **jacket content** (one input and output *valve connection*, not displayed in Fig. 8.56) are realized by means of the *environment connection* (Sect. 5.2), which also may be characterized by means of *phenomena*. In particular, each *valve connection* is associated with the *phenomenon* of **interface mass convection** (cf. Fig. 8.56).

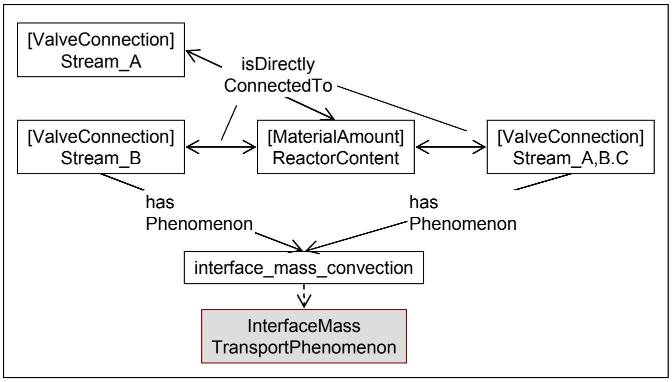
[ValveConnection] Stream_A
isDirectly ConnectedTo
[ValveConnection] Stream_B
[MaterialAmount] ReactorContent
[ValveConnection] Stream_A,B.C
has Phenomenon
has Phenomenon
interface_mass_convection
InterfaceMass TransportPhenomenon

Fig. 8.56: Example 2: *Environment connection* for **reactor content**

Finally, the assignment of *physical quantities* influenced by particular phenomena occurring in the **reactor content** is illustrated in Fig. 8.57. In particular, the *mass hold-up* of the liquid compound is exemplarily described. For the liquid reactor content, **mass accumulation** and **energy accumulation** phenomena are considered. This implies the time-varying values of some of its properties.

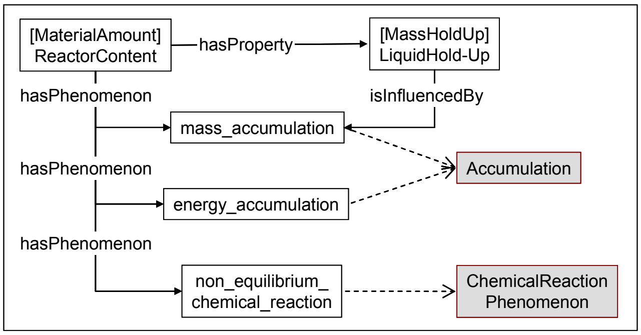
[MaterialAmount] ReactorContent
hasProperty
[MassHoldUp] LiquidHold-Up
hasPhenomenon
isInfluencedBy
mass_accumulation
hasPhenomenon
Accumulation
energy_accumulation
hasPhenomenon
non_equilibrium_ chemical_reaction
ChemicalReaction Phenomenon

Fig. 8.57: Example 2: *Phenomena* occurring in **reactor content**

8.6.1.9 Concept Descriptions

Individual concepts of the module *behavior* are defined below. For the description of the introduced *physicochemical phenomena*, we refer to Wiesner et al (2008a) and the appropriate textbooks, respectively.

Class Descriptions

Accumulation
Accumulation is a *material amount phenomenon* which denotes the accumulation of a certain extensive quantity of the *material amount* considered.

Adsorption phenomenon
An *adsorption phenomenon* is an increase in the concentration of a dissolved substance at the interface of a solid and a liquid phase due to the operation of surface forces. Adsorption can also occur at the interface of a solid and a gaseous phase (McNaught and Wilkinson 1997).

Chemical reaction phenomenon
A *chemical reaction phenomenon* is a *material amount phenomenon* in which some components are converted into some other components by molecular transformation.

Conductive transport rate
A *conductive transport rate* is the transfer of heat by direct contact of particles of matter within a phase per unit time.

Continuous material amount
A *continuous material amount* in a *heterogeneous material amount* is the *material amount* in which the disperse phase is distributed, corresponding to the solvent in a true solution.

Convective energy flow rate
A *convective energy flow rate* is the energy of a *material* which passes through a given *surface* per unit time.

Convective energy flux
A *convective energy flux* is defined as the amount of energy that passes through an interface per unit area and unit time.

Convective exchange
A *convective exchange* in the most general terms refers to the movement of molecules across the boundary of fluid phases which is the sum of advective and diffusive transport.

Convective mass flow rate
A *convective mass flow rate* is the mass of a *material* which passes through a given *surface* per unit time.

Convective mass flux
A *convective mass flux* is defined as the amount of mass that passes through an interface per unit area and unit time.

Convective momentum flow rate
A *convective momentum flow rate* is the momentum which passes through a given *surface* per unit time.

Convective momentum flux
A *convective momentum flux* is defined as the momentum that passes through an interface per unit area and unit time.

Convective transport rate
A *convective transport rate* in the most general terms refers to the movement of molecules within fluid phases which is the sum of advective and diffusive transport.

Dispersed material amount
A *dispersed material amount* is the discontinuous portion of a heterogeneous system, corresponding to the solute in a true solution. The *dispersed material amount* is assumed to be "dissolved" in the *continuous material amount*.

Dynamic viscosity of non-Newtonian fluid
Dynamic viscosity of non-Newtonian fluid is a property of non-Newtonian fluids which relates shear stress to shear rate. For a non-Newtonian fluid, this property is not a constant and can be dependent on shear rate or even time.

Energy hold-up
Energy hold-up refers to the accumulation of the overall energy in a *material amount*.

Exchange
An *exchange* represents the transport of matter or energy across a phase boundary.

Extensive property
An *extensive property* of a system depends on system size or on the amount of material in the system.

Film connection
A *film connection* is a *material amount connection* dominated by an *interface molecular transport phenomenon*. It is used to describe molecular transport across the phase interface between two contiguous *material amounts* or molecular transport through some media with negligible capacity which separates two *material amounts* (e.g. a membrane between two fluids).

Flow pattern
A *flow pattern* is a *material amount phenomenon* which refers to the flow condition of a *material amount*.

Fluid film
A *fluid film* represents a boundary layer occurring in a fluid phase which allows for conductive or diffusive transport.

Fractional amount
A *fractional amount* represents the percentage of *single particles* possessing some common characteristics.

Generalized amount
A *generalized amount* represents a generalization of *material amount* and *material amount connection*, as it could be required for the definition of quantity-related properties at a stage of the functional description.

Generalized fluxes
Generalized fluxes include the variation of holdup caused by the transport in a phase or across a phase boundary as well as sources of an extensive property caused by chemical reaction or some external potential field.

Heat radiation connection
A *heat radiation connection* is a *material amount connection* whose dominating phenomenon is *heat radiation*.

Heat transfer resistance
A *heat transfer resistance* is described by the ratio between the temperature difference and the average heat flow across the interface.

Heterogeneous material amount
A *heterogeneous material amount* is a composite *material amount* that involves material amounts with different dispersion states due to phases or particle size.

Hold-up variation
Hold-up variation refers to the accumulation of extensive properties over time and is influenced by *accumulation phenomena*.

Homogeneous material amount
A *homogeneous material amount* represents a *material amount* with a continuous, single phase which is not part of another more complex *material amount*.

Ideally mixed material amount
An *ideally mixed material amount* may neither possess an *intensive material amount property* that indicates a distribution over a spatial domain nor might it provide *properties* for *molecular transport phenomenon*.

Impermeable valve
An *impermeable valve* refers to a convective mass transport where no material transport occurs at all due to a given local condition, e.g. a blockage in a pipeline.

Interface heat transport phenomenon
An *interface heat transport phenomenon* is the transfer of thermal energy or simply heat from a hotter *material amount* to a cooler *material amount* driven by the temperature difference.

Interface mass transport phenomenon
An *interface mass transport phenomenon* is any mechanisms by which particles or quantities move from one *material amounts* to another.

Interface transport phenomenon
Interface transport phenomenon subsumes transport phenomena that occur at the interface between two connected *material amounts.*

Inter phase transport coefficient
An *inter phase transport coefficient* is any *physical quantity* that is forced by an *interface transport phenomenon.*

Intraphase transport
Intraphase transport considers all variants of heat and mass transfer that can occur with a particular phase.

Mass hold-up
Mass hold-up refers to the accumulation of the overall mass covered in a *material amount.*

Mass transfer coefficient
A *mass transfer coefficient* is a constant that relates the mass transfer rate to the product of mass transfer area and an appropriate driving force such as the concentration gradient (Seader and Henley 1998).

Material amount
A *material amount* characterizes the time-variant behavior of a chunk of material.

Material amount connection
A *material amount connection* is a *connection* that connects two *material amounts* or connects one *material amount* and one *environment connection.*

Material amount connection phenomenon
A *material amount connection phenomenon* is a *physicochemical phenomenon* which occurs at a *material amount connection.*

Material amount in phase equilibrium
A *material amount in phase equilibrium* must neither have *properties* indicating molecular transport nor spatial distribution.

Material amount phenomenon
A *material amount phenomenon* is a *physicochemical phenomenon* which occurs in a *material amount.*

Molecular transport phenomenon
A *molecular transport phenomenon* is a *material amount phenomenon* in which physical quantities such as mass, energy, and momentum are transported among different locations through molecular motion in the *material amount.*

Material amount with molecular transport phenomenon
A *material amount with molecular transport phenomenon* may not possess properties of a *ideally mixed material amount.*

Material amount with spatially distributed intensive properties
A *material amount with spatially distributed intensive properties* may not have any properties indicating an ideal mixing.

Particle phenomenon
A *particle phenomenon* is a *material amount phenomenon* which occurs with one or more *particles.*

Particle population
A *particle population* consists of a (possibly uncounted) number of *single particles,* which are all present in the same state of aggregation and – in their entirety – can be characterized by means of distribution curves or population balances (Ramkrishna 1985).

Particulate material amount
A *particulate material amount* represents the dispersed material amount in a heterogeneous system which is composed of *single particles* which typically hold uneven characteristics.

Permeability
Permeability denotes a set of chemical components which are permeable in a *film connection* or a *valve connection*.

Permeable valve
A *permeable valve* represents convective transport of mass, where all material compounds, energy, and momentum are transported.

Phase equilibrium phenomenon
A *phase equilibrium phenomenon* is a *material amount phenomenon* that denotes a certain equality of properties within the *material amount* considered, which does not involve chemical reactions.

Phenomenological coefficient
A *phenomenological coefficient* summarizes various coefficients employed to characterize fluxes.

Physicochemical phenomenon
A *physicochemical phenomena* is a *phenomenon* that can be described by physics or chemistry.

Quasi-homogeneous material amount
A *quasi-homogeneous material amount* assumes at least two parts, a *dispersed material amount* and a *continuous material amount*, where one is dispersed in the other. It is characterized by average *physical quantities* of both parts.

Radiation exchange
A *radiation exchange* is defined as the emission of heat by one body which travels through a medium or through space and which is ultimately absorbed by another body.

Rate of reaction
The *rate of reaction* of a general *phase reaction* $aA + bB + \ldots \rightarrow pP + qQ + \ldots$ is defined as

$$r = -\frac{1}{a}\frac{d[A]}{dt} = -\frac{1}{b}\frac{d[B]}{dt} = \frac{1}{p}\frac{d[P]}{dt} = \frac{1}{q}\frac{d[Q]}{dt},$$

where symbols placed inside square brackets denote the concentrations of the species involved in the reaction. Thus, the *rate of reaction* is defined as the change in concentration per unit time. Different measures of concentration may be chosen, such as *phase component fraction* and *volume-based concentrations*. When a catalyst is used, the reaction rate may also be stated on a catalyst weight (mol g^{-1} s^{-1}) or surface area (mol m^{-2} s^{-1}) basis.

Reaction rate coefficient
A *reaction rate coefficient* of any reaction is a constant that relates the reaction rate to the concentration-dependent term in the reaction rate expression. This constant is thus independent of concentration and time (McNaught and Wilkinson 1997).

Semi-permeable valve
A *semi-permeable valve* represents convective transport of mass in which only some selective species in a mixture are transported in addition to energy and momentum.

Single film connection
A *single film connection* represents diffusive transport processes across a single boundary layer, e.g. fluid or solid.

Single particle
A *single particle* is a *material amount* which can be modeled by means of population balances. It is usually of a small size.

Solid film
A *solid film* represents a boundary layer occurring in a solid phase which allows for diffusive transport.

Source
A *source* is caused by chemical reaction or some external potential field within a *material amount*.

State variable gradient
A *state variable gradient* is the spatial gradient of an *intensive thermodynamic state variable*.

Surface phenomenon
A *surface phenomenon* is a *material amount connection phenomenon* that occurs on a *surface*.

Surface reaction phenomenon
A *surface reaction phenomenon* is a *surface phenomenon* that denotes a chemical reaction which takes place on a surface.

Three film connection
A *three film connection* represents diffusive transport processes across a boundary layer in which three films are adjacent to each other, e.g. fluid-solid-fluid.

Two film connection
A *two film connection* represents diffusive transport processes across a boundary layer in which two films are adjacent to each other, e.g. fluid-fluid.

Valve connection
A *valve connection* refers to convective mass transport which is governed by different states of *permeability*.

Velocity gradient
A *velocity gradient* is the partial derivatives of velocity with respect to the spatial coordinates.

Relation Descriptions

hasFraction
The relation hasFraction indicates the fractional amount a *representative particle* may have in a *distributed material amount*.

hasPermeableChemicalComponent
The relation hasPermeableChemicalComponent indicates which *chemical component* may pass or diffuse through a certain *material amount connection*.

hasPermeability
The relation hasPermeability indicates whether or not a *material amount connection* has a selectivity for certain *chemical components*.

hasRepresentativeParticle
The relation hasRepresentativeParticle assigns a particular *representative particle* to a *distributed material amount* to represent the common features of the particles in this amount.

indicatesFraction
The relation indicatesFraction refers a certain fraction to the corresponding *member*.

isDispersedIn
The relation isDispersedIn indicates that a *dispersed material amount* is partly or totally dispersed in a *continuous material amount*.

refersToParticlePopulation
The relation refersToParticlePopulation indicates which *particle population* a particular *fractional amount* refers to.

refersToMaterial
The relation refersToMaterial indicates which *material* a *material amount* or a *material amount connection* refers to.

surrounds
The relation surrounds describes that a *continuous material amount* surrounds a *dispersed material amount*.

8.7 CPS Performance

The performance of a *chemical process system* may be assessed from different perspectives such as costs, environmental impact (e.g. product life cycle analysis), safety and many more. However, the current version of OntoCAPE only considers the *economic performance* during the design tasks, although other performance indicators (e.g. sustainability) in different lifecycle phases (e.g. operation) may be elaborated when required. The concepts provided in this section are related to cost estimation. Cost estimations are understood to form the basis for company management to decide if (further) capital should be invested in a project. The partial model proposed refers to engineering economics concepts compiled by the textbook of Peters and Timmerhaus (1991). This partial model is by far the least extensive one among those established in **chemical_process_system**.

8.7.1 Economic Performance

This section deals with the concepts for analyzing costs and profits, which are typically used to (i) predict whether capital should be invested in a particular project and (ii) to check the actual financial results typically be done by an accountant. Hence the major classes introduced are the *costs* and *earnings* which have to be considered for a thorough analysis of the *economic performance* of a *chemical process system*.
The overall costs of a *chemical process system* comprise the costs related to processing and control systems as well as general costs that cannot be related directly to one particular *aspect system* or their elements. General costs include, for

example, the expenditure for research and development and that for sales. This module considers primarily costs that are related to the *chemical process system* in general. However, product costs are also introduced for the sake of generality whereas all costs pertaining to labor are not considered. Costs are incurred in all lifecycle phases: the design of the *chemical process system*, the building and commissioning of the plant, the operation and maintenance, and finally the decommissioning. Thus, the different costs are usually not categorized primarily according to the different *subsystems* of a *chemical process system* they can be associated to, but rather according to their type – that is, whether they are related to the capital investment or to the manufacturing process, maintenance or financial services. At a secondary level, however, the different costs can be linked to the *subsystems* where these costs arise. The following description of the *costs* is taken from (Bayer et al. 2001).

8.7.1.1 High-Level Concepts

Costs are considered as a *property* of the *chemical process system performance*. Therefore, *economic performance* is introduced as a subclass of *system* (cf. Fig. 8.58).
The *costs* of a *chemical process system* include all expenses that are related to the design, construction, and maintenance of the *chemical process system*, and the production and sales of the *end products*.

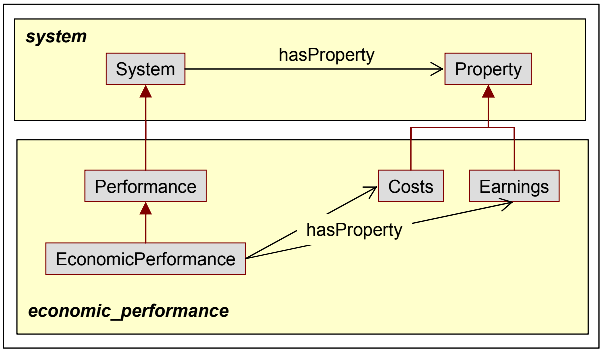

Fig. 8.58: Assignment of chemical process performance

Generally, the emerging *costs* may consist of several specific costs related to a special function or realization of the *chemical process system*. These specific costs may represent either *individual costs*, which are generated by a certain task, or *accumulated costs*, which constitute the sum of several *individual costs* (cf. Fig. 8.59). Hence, *accumulated costs* are only summed up from specific costs that emerge on the next lower level of the cost hierarchy. As an example: the *total CPS costs* represents the sum of the costs of the *total capital investment* and the *production costs*. Therefore, the relation addsUp as a specialization of hasDirectPart is introduced. As an example, *total CPS costs* is introduced which adds up the *total capital investment* and *production costs* in Fig. 8.59. Note that the introduced relation addsUp is used exclusively to relate *costs*.

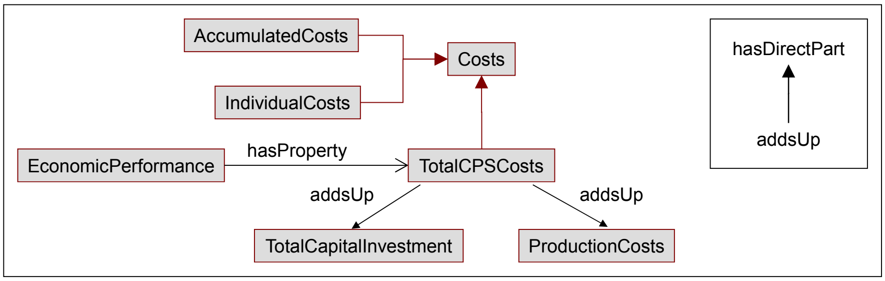

Fig. 8.59: Representation of costs

Earnings refer to the total amount of capital received as a result of the sales of goods (and also services). The earnings obtained from the sales of goods depend on the value of the products and their amount. Here, *product earnings* and *virtual intermediate earnings* are distinguished (cf. Fig. 8.60).

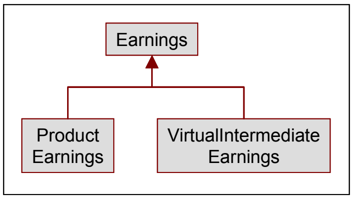

Fig. 8.60: Representation of costs

Product earnings are the earnings that can be obtained from the sales of the *end products* processed in a *chemical process system*. The *virtual intermediate earnings* are used to characterize the value of a *feed* or an *intermediate product* within the process. These earnings could be assessed by the assumed sale of those products at the market. Usually no such sale is attempted, since the earnings that can be obtained from the sale of the end product produced from the intermediates is much higher.

8.7.1.2 Modeling of Decomposition

A comprehensive introduction and categorization of costs related to chemical process systems is given by Peters and Timmerhaus (1991). Following their categorization, different types of *costs* are exemplarily introduced, as shown in

Fig. 8.61-Fig. 8.63. For a comprehensive overview we refer to Bayer et al. (2001). A detailed description of these *costs* is provided below.

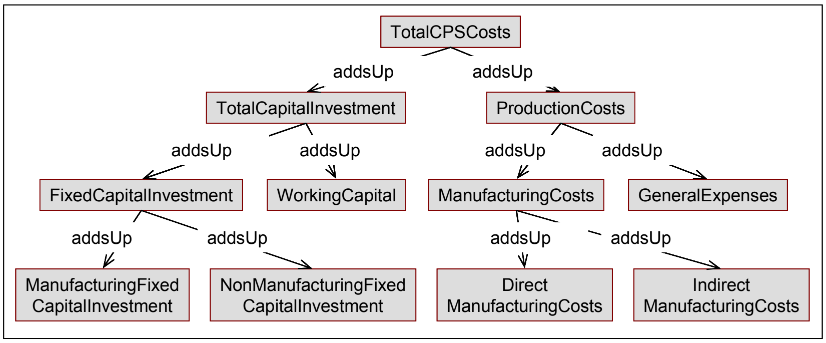

Fig. 8.61: Representation of the specific costs involved in *total CPS costs*

Fig. 8.61 gives an overview on the specific costs that are considered for the *economic performance*. The *total capital investment* and the *production costs* are shown together as parts of the *total CPS costs*. They are in fact both part of the *total CPS costs*, but there is one fundamental difference between them: while the capital investment needs to be supplied prior to production in order to design and build a chemical process system, the production costs are incurred during production over a long period of time. The *earnings* are also obtained during production period. These different periods or timeframes need to be taken into consideration in economic evaluations.

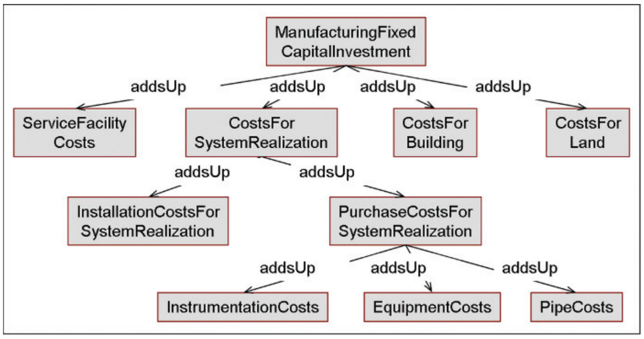

Fig. 8.62: Exemplary extension of *individual costs* for *manufacturing fixed capital investment*

The left most branch of Fig. 8.61 apparently refers to the design perspective only. Fig. 8.62 details a specific type of costs related to **CPS_realization**.

The right branch in Fig. 8.61 refers to the costs incurred during production. These costs may be applied for a rather rough cost estimation at the design stage, e.g. covering *raw material costs* or *catalyst and solvent costs*, but they may also be used for the evaluation of the operating cost during the later stages of the plant lifecycle (cf. Fig. 8.63).

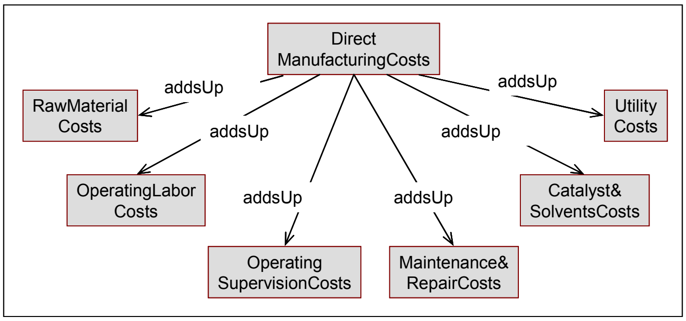

Fig. 8.63: Exemplary extension of *individual costs* for *direct manufacturing costs*

In the following, both branches of the classes related to costs and the earnings are investigated in more detail, demonstrating the dependencies to the other partial modules of **chemical_process_system**.

8.7.1.3 Dependencies between the Aspect Systems for Economic Performance Evaluation

The different costs, expenses, and earnings introduced in the previous section are related to the *chemical process system*, its (sub-)systems, and the processed material. These relations and dependencies can be modeled by refining the dependency relation between the *aspect systems* of *chemical process system*.
The total *costs* and the overall *earnings* can directly be associated to the *chemical process system* itself (cf. Fig. 8.64).

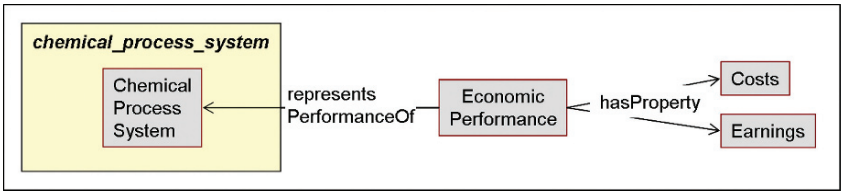

Fig. 8.64: Relation between *economic performance* and *chemical process system*

Furthermore, the following associations between the different cost items and the parts of the *aspects systems* of the *chemical process system* can be identified: *earn-*

ings are related to the *end products* of the chemical process; *manufacturing costs* refer to processed chemical products, (i.e. *end products* (cf. Fig. 8.10)) to the functional aspects of the *chemical process system* (i.e. *process steps* (Sect. 8.2.1) and to *control* (Sect. 8.2.2)); investment costs, especially purchase costs, are related to the realization of the *chemical process system* (i.e. the *plant* (Sect. 8.3.1) and the *process control system* (Sect. 8.3.2)). The most important of these associations will be identified and discussed in the following.
The main purpose of a *chemical process system* is the production of some specific chemical compounds from raw materials. The different chemical compounds involved in this production process are important cost factors. The purchase of the *raw materials* and the sale of the *end products* are related to major cost items within the lifecycle of a *chemical process system* (Fig. 8.65): the *product earnings* are obtained from the sales of the *end products*; the *virtual intermediate earnings* can be related to *feeds* and *intermediate products* whose values they represent; the *raw material costs* are the expenses needed for purchasing the *raw materials*. Further important *feeds* are auxiliary materials like catalysts and solvents; their purchase costs are represented by the *catalysts and solvents costs*.

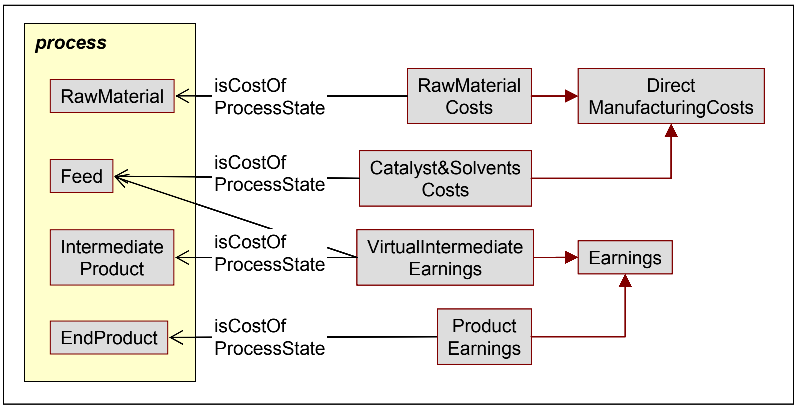

Fig. 8.65: Relation between *economic performance* and *chemical process system*: material costs.

The *direct manufacturing costs* can be further related to the functional description of the chemical process system (cf. Fig. 8.66). This overall function is implemented by the processing part (i.e. *process step*) and the operating part (i.e. *control loop*). These different functions cause different costs. The *utility costs* and the *maintenance and repair costs* are induced by the *process step*. Characteristic properties of a *process step* like design temperature and design pressure influence the *utility costs* tremendously; the *maintenance and repair costs* depend among other things on the processing and auxiliary materials used and on the operating conditions of the *process step*. The main*tenance and repair costs* also depend on the chosen control strategy (i.e. *control loop*).

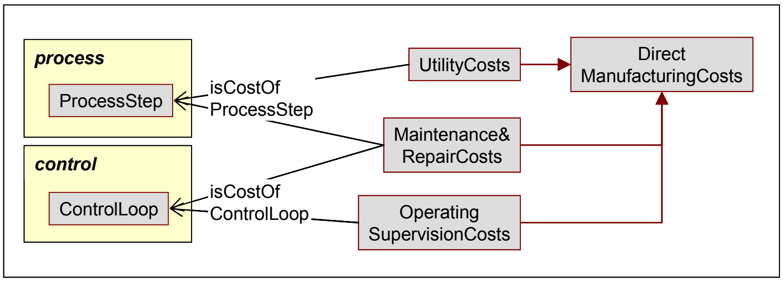

Fig. 8.66: Relation between *economic performance* and *chemical process system*: *direct manufacturing costs*

As shown in Fig. 8.67, the *purchase costs for systems realization* refer to the chemical process realization. The single items of the purchase costs for systems realization can be related to the single *plant items*. *Pipe costs* correlate with the *pipe*; *equipment costs*, like cost for purchasing apparatuses or machines are related to the respective equipment; and, *instrumentation* costs refer to the *instrumentation* as well as the *process control system*. The design of the apparatuses and the machines also influences the *maintenance and repair costs*.

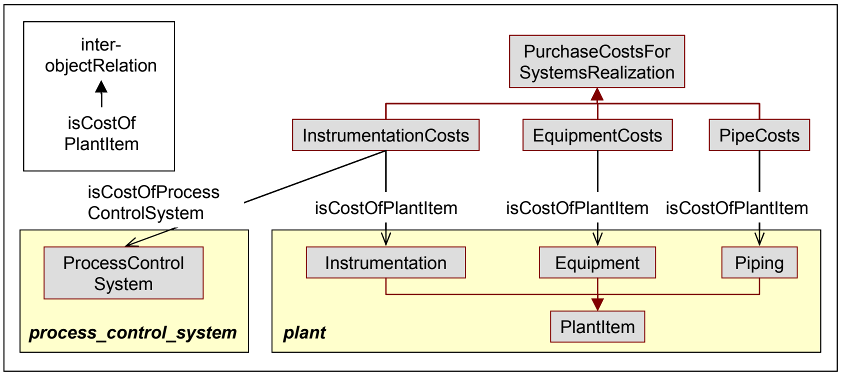

Fig. 8.67: Relation between *economic performance* and *plant*

8.7.1.4 Concept Descriptions

Individual concepts of the module *economic_performance* are defined below. For an exhaustive description of all involved concepts, we refer to Wiesner et al. (2008a).

Class Descriptions

Accumulated costs
Accumulated costs is the sum of a number of specific costs, which itself may be *individual costs* or *accumulated costs*, occurring in a *chemical process system.*

Catalyst and solvent costs
Catalyst and solvent costs are *direct manufacturing costs* which are related to the particular catalysts and solvents used in the manufacturing process.

Costs
Costs describe all kinds of costs that may arise with respect to the economic evaluation of a *chemical process system*.

Costs for buildings
Costs for buildings incur during the construction of all buildings in a *plant*.

Costs for land
Costs for land cover the real-estate costs price and the yard improvement costs.

Costs for systems realization
Costs for systems realization are *costs* directly associated with the *processing subsystem* and the *operating subsystem*.

Direct manufacturing costs
Direct manufacturing costs include expenses directly associated with manufacturing. Besides, it may also involve secretarial work directly related to the manufacturing process, laboratory charges and patents.

Earnings
The *earnings* refer to the total amount of capital received as a result of the sales of goods and services.

Economic performance
Economic performance evaluates the performance of a *chemical process system* from an economic perspective.

Equipment costs
Equipment costs are *costs* incurred by the purchase of *equipment.*

Fixed capital investment
Fixed capital investment is the capital that is permanently invested in the manufacturing facilities.

General expenses
General expenses include, for example, general research and development expenses or the costs for a sales office within a company, where the products of more than one *chemical process system* are brought to the market.

Indirect manufacturing costs
Indirect manufacturing costs include plant overhead costs and fixed charges.

Individual costs
Individual costs represents *costs* that are not added up from any other *costs.*

Installation costs for systems realization
Installation costs for systems realization include, for example, the costs for labor, platforms, and construction.

Instrumentation costs
Instrumentation costs are the *costs* related to the *process control system.*

Maintenance and repair costs
Maintenance and repair costs are related to maintenance and repair work on production facilities.

Manufacturing costs
Manufacturing costs are expenses that are directly connected with the manufacturing.

Manufacturing fixed capital investment
Manufacturing fixed capital investment represents the capital necessary for the installed process equipment with all auxiliaries that are needed for complete process operation (Peters and Timmerhaus 1991).

Non-manufacturing fixed capital investment
Non-manufacturing fixed capital investment represents fixed capital required for construction overhead and for all plant components that are not directly related to process operation (Peters and Timmerhaus 1991).

Operating labor costs
Operating labor costs are related to work force in production, including related taxes and benefits.

Operating supervision costs
Operating supervision costs are related to systems and procedures supervising the manufacturing process.

Pipe costs
Pipe costs are the costs related to the installation of piping network in the *plant.*

Product earnings
The *product earnings* are the earnings that can be obtained from the sales of the *end products* of the processing system.

Production costs
Production costs cover the costs for operating the *chemical process system* and selling the *core products.*

Purchase costs for systems realization
Purchase costs for systems realization are the costs related to the acquisition of the *chemical process system*.

Raw material costs
Raw material costs are related to the purchase of raw material including freight and transportation. This type of costs is among the major costs in the chemical process industry and its ratio to total plant cost varies considerably for different types of chemical process systems.

Service facility costs
Service facility costs are the costs related to utilities (power, cooling water, etc.), waste disposal, fire protection, and other services.

Total CPS costs
Total CPS costs of a *chemical process system* include all costs that are related to design, construction, and maintenance of the *chemical process system*, as well as the production and sales of the *end products*.

Total capital investment
Total capital investment is the sum of *fixed capital investment* and *working capital*.

Utility costs
Utility costs are related to the operaton of the plant. It typically includes costs for electricity, process steam, refrigerants, compressed air, cooling water, heated water, hot oil, etc.

Virtual intermediate earnings
The *virtual intermediate earnings* are used to characterize the value *feed* or an *intermediate product* within the process.

Working capital
Working capital is the capital needed for operating a plant.

Relation Descriptions

addsUp
The relation addsUp indicates that specific *costs* belong to particular *accumulated costs*.

8.8 Process Units

Summarizing the content of the previous sections, a *chemical process system* can be conveniently described according to the aspects: **function**, **realization**, **behavior** and

performance[95]**.** For each of the four aspects, generic concepts have been provided such that arbitrary kinds of *process units* can be represented using those concepts. However, the evolution of the discipline of chemical engineering has resulted in a number of "typical" *process units*. Frequently the term "unit operations" is used synonymously. However, here we adopt the term "process units" instead of "unit operations", since the latter is often (though implicitly) intended to refer to the functional aspect of a *chemical process system*, in the same manner as in OntoCAPE.

As already introduced in Sect. 8.1.1, a *process unit* is a partial *chemical process system*, reflected through its aspects of **function**, **realization**, **behavior** and **performance**. Particular constraints are imposed on the *process unit* and on its aspects in order to distinguish the *process unit* from others.

In this section a number of *process units* of well-established types will be exemplarily defined (see Fig. 8.68 below). Naturally, for each type of *process units*, there can be a class hierarchy according to the classification employed in chemical engineering practice. For each individual class, a large amount of information may be provided if a rather complete description is to be attempted for each aspect (i.e. **function**, **behavior** or **realization**). Due to the limited level of detail provided by the current version of OntoCAPE, these partial models have actually been developed only for serving the following two purposes instead of rendering a rather complete ontology: (i) demonstrating how *process units* can be defined on the basis of the concepts of generic *chemical process system*, and (ii) providing the concepts necessary for those particular projects in which OntoCAPE has yet been applied (cf. Chap. 12).

For the time being, six classes are introduced representing the very basic process units most frequently employed. These *process units* include *mixing unit*, *splitting unit*, *flash unit*, *chemical reactor*, *heat transfer unit* and *distillation system*, as shown in Fig. 8.68.

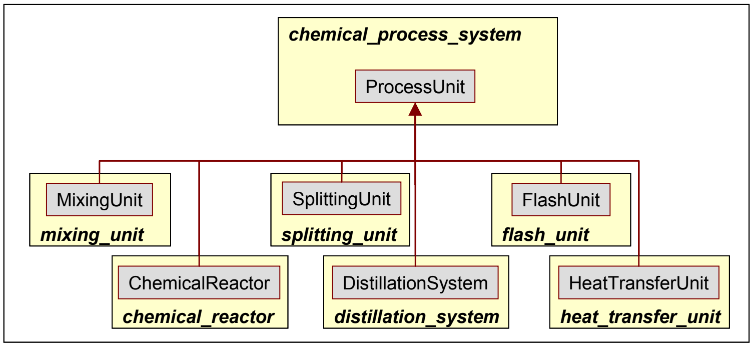

Fig. 8.68: Overview on exemplary modules contained in *process unit*

[95] The aspect of operation is clearly missing in the list. This however, will be included in future versions of OntoCAPE to be able to describe all phase of a plant's lifecycle.

Among these classes, *heat transfer unit* has been the one worked out in the greatest level of detail. It will thus serve as an example in this chapter to represent the specification of a particular process unit. However, as stated before, extensions to other classes and the introduction of new classes may be undertaken on demand at a later stage.

8.8.1 Heat Transfer Unit

This module basically defines *process units* which perform heat transfer. A general *heat transfer unit* is defined next. It is essentially a *process unit* which has the function of *enthalpy change* and is realized in an *apparatus*. Its behavioral aspect comprises two (or more) *heat transfer material amounts* connected via *heat transfer connection(s)*. A graphical representation is given in Fig. 8.69.

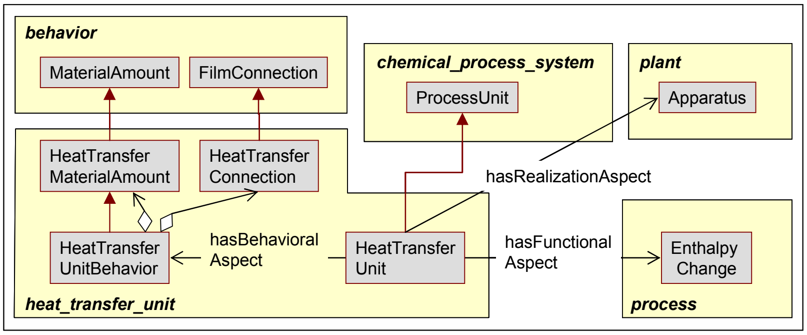

Fig. 8.69: Overview on the interactions of *heat transfer unit* with other modules

8.8.1.1 High-Level Concepts

In chemical engineering practice different types of *heat transfer units* are employed for the particular tasks at hand. Therefore, some exemplarily specializations in terms of realization and function are introduced subsequently. With respect to functionality, the most frequently applied *heat transfer units* are *heat exchangers* for a *temperature change* and *condensers* and *reboilers* for a *phase change* (cf. Fig. 8.70)

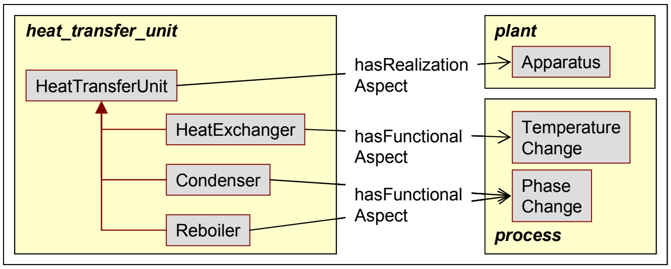

Fig. 8.70: Exemplary specializations of a *heat transfer unit*

Additionally, a further specialization in terms of realization may be undertaken. According to the apparatuses stated previously, *heat exchangers* are frequently built as *shell tube apparatuses*. Hence, the class *shell tube heat exchanger* including the corresponding class of *shell tube heat exchange behavior* is introduced (cf. Fig. 8.71).

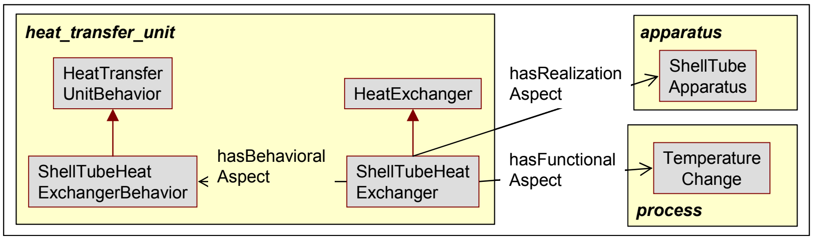

Fig. 8.71: Exemplary specialization of *heat exchanger*

8.8.1.2 Modeling of Decomposition

For a realistic description of a *heat transfer unit* a further elaboration of the *heat transfer unit behavior* is addressed subsequently (cf. Fig. 8.72). To start with, the behavior may be described by a *hot side material amount* and a *cold side material amount,* which are both specializations of *heat transfer material amount*. The latter class is introduced primarily for the users' convenience. If a user wants to assign some *property* or *phenomena* which are relevant to the *hot side material amount* as well as the *cold side material amount* simultaneously, only the superclass *heat transfer material amount* has to be used to cover the assignments. Moreover, *heat transfer connection* is introduced as a specialization of *film connection* for linking the aforementioned *material amount* classes. Finally, *heat leak* and *heat loss* are established as *connections* to the (not explicitly defined) environment which may act as a heat source or a heat sink.

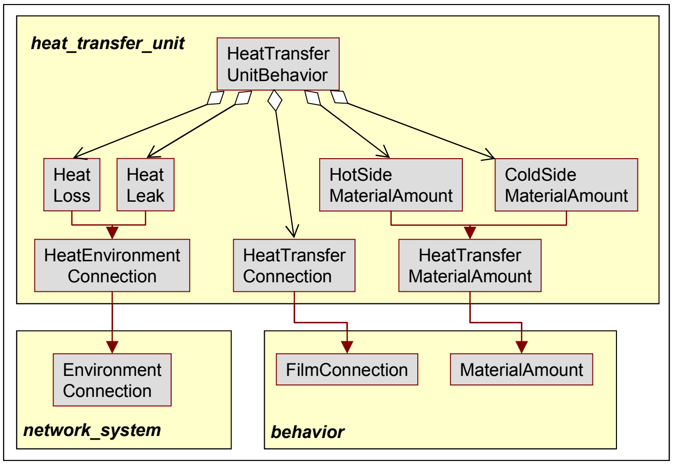

Fig. 8.72: Exemplary elaboration of *heat transfer unit behavior*

For the elaboration of *shell tube heat exchange behavior* only a few specializations are further considered and displayed in Fig. 8.73.

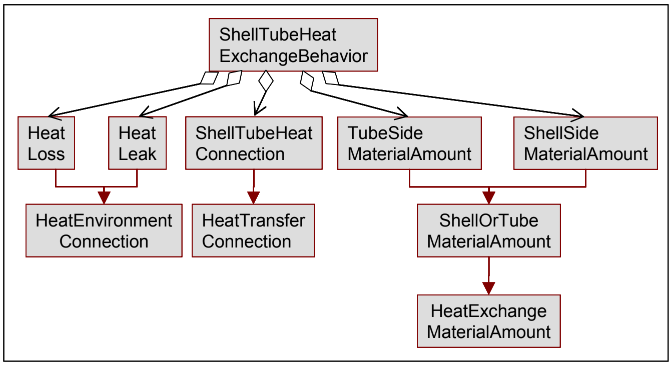

Fig. 8.73: Description of a *shell tube heat exchange behavior*

To start with, the general *shell or tube material amount* class is introduced as a specialization of *heat exchange material amount*. Again, this class may be used for covering properties or phenomena assignments which refer to the subclasses. *Tube side material amount* and *shell side material amount* are introduced to represent either the cold or the hot side of the *heat exchanger*. Next, the specialization comprises a *shell tube heat connection* for describing the particular phenomena occurring at the connection between the tube and the shell side material amount. Finally, *heat loss* and *heat leak* are considered in the way they are already defined.

8.8.1.3 Modeling of Connectivity

The connectivity of the concepts introduced so far builds on the primitives defined in *behavior* (Sect. 8.6.1). The simplest, non-specific connection between the classes, e.g. the case where only two *heat transfer material amounts* without a distinct heat transfer direction have to be modeled, is shown in Fig. 8.74.

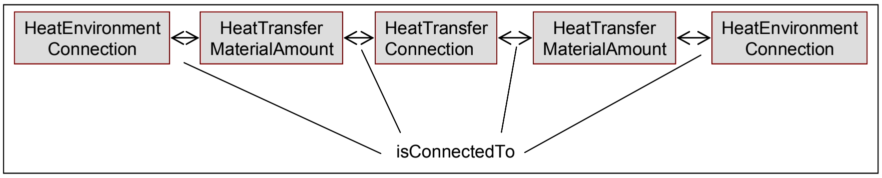
HeatEnvironment Connection
HeatTransfer MaterialAmount
HeatTransfer Connection
HeatTransfer MaterialAmount
HeatEnvironment Connection
isConnectedTo

Fig. 8.74: Connectivity of *heat transfer material amount*

8.8.1.4 Properties of Heat Transfer Unit

Some exemplary properties for *heat transfer connection* are introduced (cf. Fig. 8.75). Clearly, some of the *physical properties* defined in *behavior* may be used for the description, such as e.g. *heat transfer coefficient*, *conductive transport* or *heat transfer resistance*.

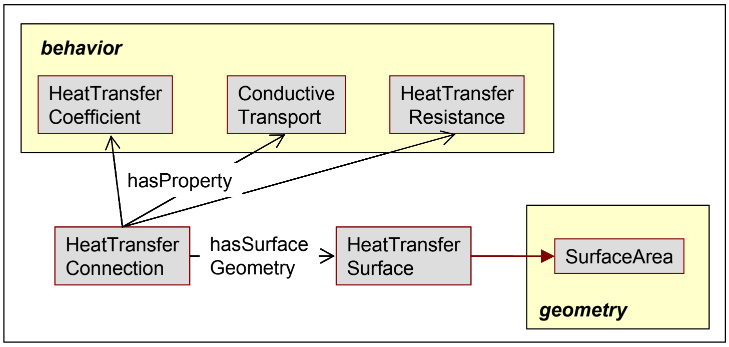

Fig. 8.75: Exemplarily relations for *heat transfer connection*

These properties may be further specialized if required. Furthermore, geometrical considerations may be important to describe the interface to the *heat transfer material amount*. Thus a *heat transfer surface* may be depicted if any non-trivial geometry has to be considered. Therefore the relation hasSurfaceGeometry is introduced.
For an exhaustive description of all concepts involved in the module *process_units*, we refer to Wiesner et al. (2008a).

8.9 References

Bayer B, Schneider R, Marquardt W (2000) Integration of data models for process design – first steps and experiences. *Comput. Chem. Eng.* **24**:599-605.

Bayer B, Krobb C, Marquardt W (2001) *A Data Model for Design Data in Chemical Engineering – Information Models*. Technical Report LPT–2001–15. Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bird RB, Stewart WE, Lightfoot EN (2001) *Transport Phenomena*. John Wiley, New York.

Bogusch R (2001) *A Software Environment for Computer-Aided Modeling of Chemical Processes*. Fortschritt-Berichte VDI: Reihe 3, Nr. 705, VDI-Verlag, Düsseldorf.

EClass (2009) *eCl@ss*. Online available at: http://www.eclass.de.

Encyclopedia Britannica (2009) *Unit process*. Online available at: http://www.britannica.com/EBchecked/topic/615307/unit-process.

Föllinger O (1992) *Regelungstechnik – Einführung in die Methoden und ihre Anwendung*. Hüthig, Heidelberg.

Froment GF, Bischoff, KB (1990) *Chemical Reactor Analysis and Design.* John Wiley and Sons, New York.

Früh KF (ed.) (2000) *Handbuch der Prozessautomatisierung*. Oldenbourg, München.

Graßmuck J, Houben KW, Zollinger RM (1994) *DIN-Normen in der Verfahrenstechnik*. Teubner, Stuttgart.

Green DW, Perry RH (deceased) (1997) *Perry's Chemical Engineers' Handbook*. 7th Edition. McGraw-Hill, New York.

Haase R (1990) *Thermodynamics of Irreversible Processes*, Dover Publications, New York.

Incropera FP, De Witt DP (1990) *Fundamentals of Heat and Mass Transfer* (3rd ed.). John Wiley, New York.

ISO 10303, Part 231 (1998) *Process Engineering Data: Process Design and Process Specifications of Major Equipment*. ISO TC 184/SC4/WG3 N740.

Krishna R, Taylor R (1993) Multicomponent mass transfer: theory and applications. In Cheremisino NP (ed): *Handbook of Heat and Mass Transfer*, Gulf Publishing Company, **2**:259-432.

Lauber J (1996) *Methode zur funktionalen Beschreibung und Analyse von Produktionsprozessen als Basis zur Realisierung leittechnischer Lösungen*. Dissertation, Lehrstuhl für Prozessleittechnik, RWTH Aachen University.

Marquardt W (1992b) Rechnergestützte Erstellung verfahrenstechnischer Prozessmodelle. *Chem. Ing. Tech.* **64**:25-40

Marquardt W (1995) Towards a Process Modeling Methodology. In: R. Berber: *Methods of Model-Based Control*. NATO-ASI E, Applied Sciences, **293**, Kluwer, Dordrecht:3-41.

Marquardt W (1996) Trends in computer-aided process modeling. *Comput. Chem. Eng.* **20** (6/7):591–609.

Matthes F (1959) Zur Systematik der chemischen Technologie – Teil 2, *Chem. Tech.*:536-543.

McCabe WL, Smith JC, Harriott P (2004) *Unit Operations in Chemical Engineering*. 7th Edition, McGraw-Hill, New York.

McNaught AD, Wilkinson A, eds. (1997) *Compendium of Chemical Terminology*, 2nd Edition. Blackwell Science, Oxford, UK. Electronic version online available at http://goldbook.iupac.org/.

Miller DC, Josephson JR, Elsass MJ, Davis JF, Chandrasekaran B (1997) Sharable engineering databases for intelligent system applications. *Comput. Chem. Eng.* **21**:77-82.

Patzak G (1982) *Systemtechnik – Planung komplexer innovativer Systeme*. Springer, Berlin.

Peters MS, Timmerhaus KD (1991) *Plant Design and Economics for Chemical Engineers*. McGraw-Hill, New York.

Polke M, ed. (1994) *Process Control Engineering*, VCH, Weinheim.

Prolist (2009) *Prolist*. Online available at: http://www.prolist.org.

Ramkrishna D (1985) The status of population balances, *Rev. Chem. Eng.* **3**:49-95.

Sargent RWH (1998) A functional approach to process synthesis and its application to distillation systems. *Comput. Chem. Eng.* **22**:31-45.

Schuler H, ed. (1999) *Prozessführung*. Oldenbourg, München.

Seader JD, Henley EJ (1998) *Separation Process Principles*. John Wiley, New York.

Shreve RN (1978) *Chemical Process Industries*. McGraw-Hill, New York.

Smith JM (1981) *Chemical Engineering Kinetics.* 3rd Edition, McGraw-Hill, New York.

TGL 25000 (1974) *Chemical Engineering Unit Operations – Classification.* Departmental Standard of the German Democratic Republic.

Unbehauen H (1989) *Regelungstechnik I.* Vieweg, Braunschweig.

Wesselingh JA, Krishna R (2000) *MassTransfer in Multicomponent Mixtures.* Delft University Press.

Wiesner A, Morbach J, Bayer B, Yang A, Marquardt W (2008a) *OntoCAPE 2.0 – Chemical Process System.* Technical Report (LPT-2008-29), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer =LPT-2008-29.

Wilhelm Jr RG (1996) In search of the Cheshire cat: the invisible automation system paradox. *ISA Transactions* **35**:321-327.

9 Mathematical Models

The partial model **mathematical_model** is concerned with the description of mathematical models. Fig. 9.1 gives an overview of the ontology modules of **mathematical_model** and their interrelations. The main module, *mathematical_model* (cf. Sect. 9.1), introduces the basic concepts for mathematical modeling, including model variables as well as items pertaining to sub-models and their connections. CapeML (von Wedel 2002) was taken as an important source. The ontology module *equation_system* (cf. Sect. 9.2) further specifies the characteristics of the model equations that constitute a *mathematical model*. Based on these characteristics, an appropriate numerical solver can be selected, which is the concern of the ontology module *numerical_solution_strategy* (cf. Sect. 9.3). The modules *process_model* (cf. Sect. 9.5) and *cost_model* (cf. Sect. 9.4) describe two particular types of mathematical models: *process models* model the behavior of *process units* (cf. Sect. 8.8) and *materials* (cf. Sect. 7.1), while *cost models* predict the *costs* (cf. Sect. 8.7.1) of *chemical process systems*.

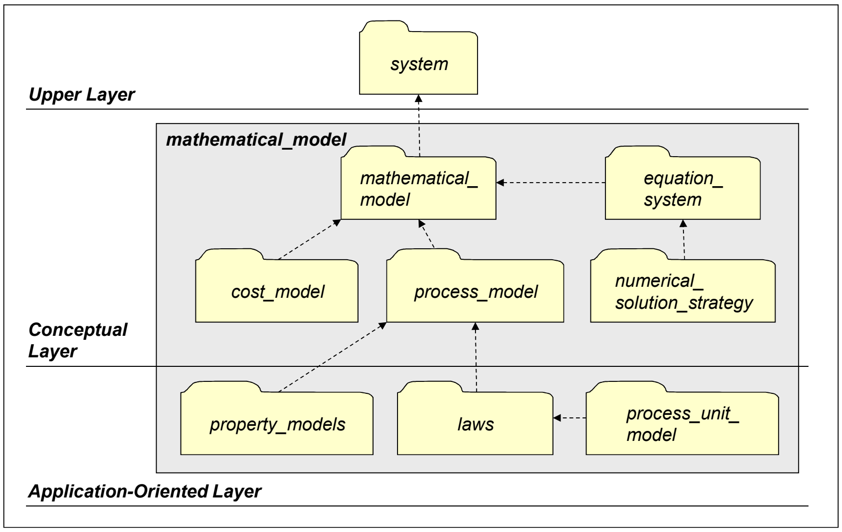

Fig. 9.1: Overview on partial model **mathematical_model**

The *process_model* module is extended on the Application-Oriented Layer: The ontology module *laws* (cf. Sect. 9.6) establishes models for a number of physical laws that are common in the context of chemical engineering (e.g., the law of energy conservation). Thus, laws may be associated with the phenomena introduced and described in *behavior* (Sect. 8.6.1). *Property_models* (cf. Sect. 9.7) provides correlations for designated *physical quantities*, such as vapor pressure correlations or activity coefficient models. Finally, the module *process_unit_model* (cf. Sect. 9.8) establishes customary mathematical models for process units, such as ideal reactor models or tray-by-tray models for distillation columns.

9.1 Mathematical Model (Ontology module)

A *mathematical model* is a special type of *model* (cf. Sect. 5.1.6), which uses mathematical expressions to describe the behavior of the modeled *system,* for example by means of simulation.

9.1.1 High-Level Concepts

A *mathematical model* has a number of *properties*, the most important of which are *model quantities*. As indicated in Fig. 9.2, a *model quantity* is a subclass of *physical quantity* that is linked to the model via the relation hasVariable (a specialization of hasProperty, cf. upper right corner of Fig. 9.2). Like any *physical quantity*, a *model quantity* has a particular *physical dimension* and can be either a *scalar quantity* (cf. Sect. 5.1.11) or a *tensor quantity* (cf. Sect. 5.5). The *value* of a *model quantity* is represented by the class *model quantity specification*; each *model quantity* has exactly one *model quantity specification*.

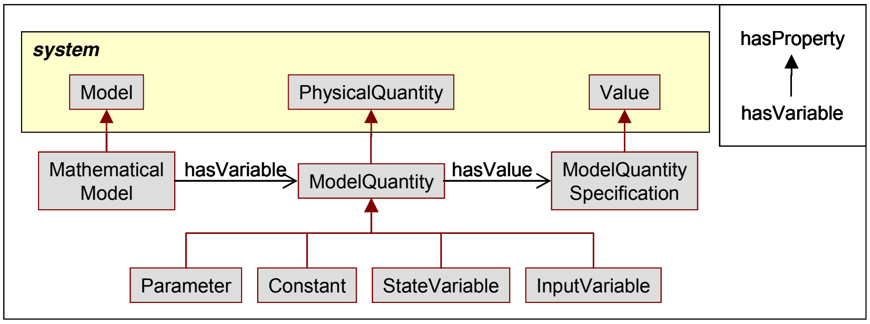

Fig. 9.2: *Mathematical model*, *model quantity*, and *model quantity specification*

A *model quantity* can be one of the following types: a *constant*, a *parameter*, a *state variable,* or an *input variable*, depending on the intended specification of its value: *Constants* and *parameters* constitute the fixed set of specified variables. *Input variables* represent time or spatially dependent inputs, which have to be specified for dynamic and/or spatially distributed systems. Finally, *state variables* constitute the fixed set of unknown variables, which have to be computed by the model. The *model quantity specification* indicates the numericalValue (cf. Sect. 5.1.11) of the *model quantities*. Unlike *constants*, *parameters* and *input variables* may have different *model quantity specifications* in different simulation runs. If the *model quantity* is of type *parameter* or *state variable*, the *model quantity specification* may indicate their *upper limit* and *lower limit* (cf. Fig. 9.3).

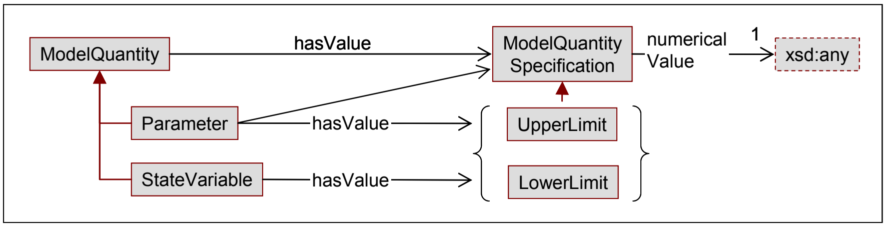

Fig. 9.3: Assignment of particular *model quantity specifications* to *model quantites*

A *system* which is the target of a models relation is classified as a *modeled object*. The correspondences between a *model quantity* and a *physical quantity* of the *modeled object* can be explicitly represented by means of the relation corresponds-ToQuantity (cf. Fig. 9.4).

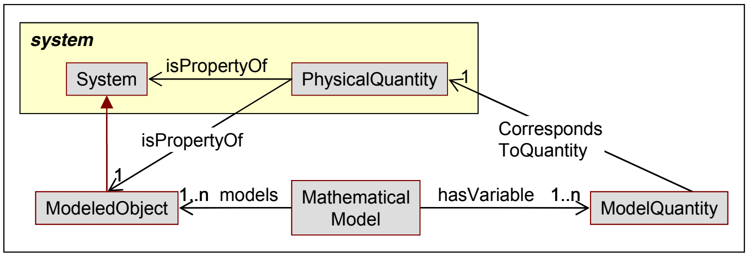

Fig. 9.4: Correspondence between a *model quantity* and a *physical quantity* of the *modeled object*

9.1.2 Modeling of Decomposition

Like any *system*, a *mathematical model* can be decomposed into *subsystems*, which are called *submodels*. The *submodel* models the same *system* as its superordinate *mathematical model*. Consequently, there is no need to specify the models relation between *submodel* and *system* explicitly. However, such a relation may be indicated if the *submodel* models a designated *subsystem* of the overall *system*.

9.1.3 Modeling of Connectivity

The different *submodels* of a *mathematical model* are coupled via their *model quantities* as explained in the following.
At first, the concept of a *model port* is introduced. A *model port* is a special type of *property set*, which comprises *model quantities* that can participate in a connection with another model. Thus, a *model port* has the function to identify and to bundle the "public" variables of a *mathematical model*.
Next, the concept of a *coupling* is established. A *coupling* is a property of the overall *mathematical model*, which defines a connection between two of its *submodels* by linking their respective *model ports*[96]. The *coupling* implicitly defines equality constraints between the *model quantities* in the two *model ports* and must be treated as such (e.g., during a degrees-of-freedom analysis of a complex model). It may be used to connect *mathematical models* both 'horizontally' (i.e., on the same level of decomposition) and 'vertically' (i.e., across levels of decomposition).
The order of the *model quantities* within a *model port* can be specified by a *port index*, as shown in Fig. 9.5. The *port index* is used to identify corresponding *model quantities* in a *coupling*: Two *model quantities* of different *model ports* are coupled if and only if their *port indices* have the same indexValues. The specification of a *port index* may be omitted if the correspondence between *model quantities* is evident from the context (e.g., if each of the coupled *model ports* comprises only a single *model quantity*, or if corresponding *model quantities* can be uniquely identified through their *physical dimension*.)

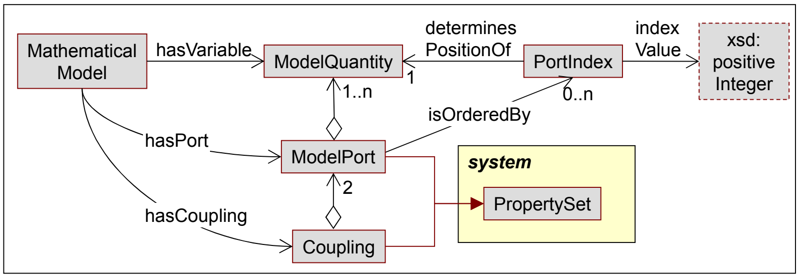

Fig. 9.5: Variables, ports, couplings

Fig. 9.6 shows exemplarily the definition of a *mathematical model* **M**, which consist of two *submodels*, **M1** and **M2**. **M1** has the *model quantities* **a**, **b**, and **c**, while **M2** has the *model quantities* **x**, **y**, and **z**. Model **M1** owns the *model port* **P1**, which comprises the quantities **b** and **c**. Similarly, the *model port* **P2** of model **M2** comprises the

[96] Please note that this way of modeling does not contradict the principles stated in *network_system* (cf. Sect. 5.2) since *model port* and *model quantity* are subclasses of *property* and are not considered as subsystems.

quantities **y** and **z**. **P1** and **P2** are coupled via the *coupling* **C**, which is a property of the overall model **M**. The corresponding quantities of the *coupling* are identified via their *port indices*: **b** and **y** have the same indexValue and are thus linked by an equality constraint. The same holds true for quantities **c** and **z**.

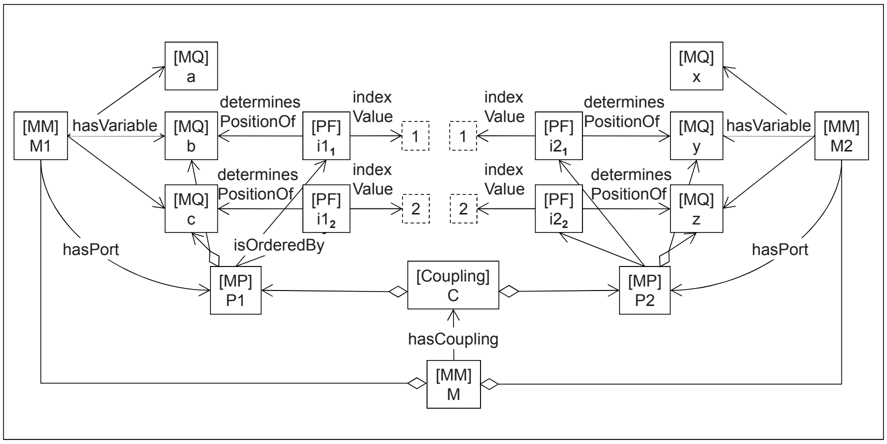

Fig. 9.6: Exemplary decomposition of model **M** into submodels **M1** and **M2**

9.1.4 Usage

The ontology module *mathematical_model* provides only the basic concepts for the description of mathematical models. For practical applications, further concepts may be required, which would typically be supplied by additional ontology modules located on the Application-Oriented Layer. Some possible extensions of the *mathematical_model* module are discussed next.

One possible extension would be the introduction of concepts suitable for representing model equations. Such an extension could be realized easily by reusing the concepts of the *mathematical_relation* module (cf. Sect. 6.1). However, such an extension is not required in practice, since specialized representation formats for mathematical equations are available, such as MathML (Ion and Miner 1999), CapeML (von Wedel 2002), or CellML (Lloyd et al. 2004).

Another possible extension would be the definition of different types (i.e., subclasses) of *model ports*. A particular *model port* type could, for example, prescribe the number of *model quantities* comprised in a *model port*, their types (i.e., *constant*, *parameter*, *input variable* or *state variable*), their *physical dimensions*, etc. Moreover, a *model port* type could be further characterized through attributes (e.g., assigning a direction to a *model port*, thus turning it into either an *inlet port* or an *outlet port*). That way, standardized model interfaces can be defined – for instance, one may

define a standard *energy port*, which contains a single scalar *model quantity* with the *physical dimension* of an energy flow and must furthermore be tagged as an *inlet* or *outlet port*. Such standardization facilitates checking the feasibility of a *coupling*: A *coupling* of two *mathematical models* will be feasible if their *model ports* (a) are of the same type (e.g., *energy port*) and (b) have matching attributes (e.g., an *inlet port* can only be coupled to an *outlet port*).
In practice, a *mathematical model* often consists of several interconnected *submodels* of the same type – for example, the model of a distillation column contains several models of distillation column trays. An application-oriented extension of *mathematical_model* could apply the *loop* design pattern introduced in the Meta Model (Sect. 4.5.5) to define such repetitive model structures. An example is given in Fig. 9.7 and Fig. 9.8.

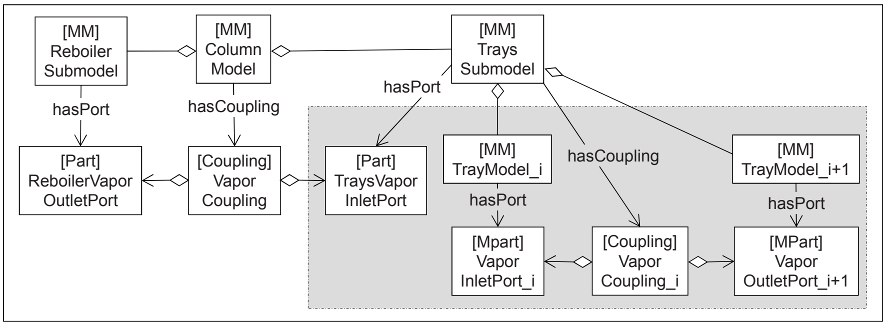
[MM] Reboiler Submodel
[MM] Column Model
[MM] Trays Submodel
hasPort
hasCoupling
hasPort
[Part] ReboilerVapor OutletPort
[Coupling] Vapor Coupling
[Part] TraysVapor InletPort
[MM] TrayModel_i
hasCoupling
[MM] TrayModel_i+1
hasPort
hasPort
[Mpart] Vapor InletPort_i
[Coupling] Vapor Coupling_i
[MPart] Vapor OutletPort_i+1

Fig. 9.7: Specification of the overall model

Fig. 9.7 specifies the overall structure of a **Column Model**. It consists of a **Reboiler Submodel** and a **Trays Submodel**, which are coupled via a **Vapor Coupling** (to simplify matters, the liquid phase is not considered in this example). The **Trays Submodel** is defined iteratively (see grey-shaded area in Fig. 9.7). It consists of several *submodels* of the same type, which are represented by the individual **TrayModel_i**. Each **TrayModel_i** has a **VaporInletPort_i**, which is coupled to the **VaporOutletPort_i+1** of the next **TrayModel_i+1**. This connectivity statement is included in a **ForLoop** that counts from 1 to 20, as shown in Fig. 9.8, to define a structure of 20 interconnected tray models. The vapor inlet port of the 20th tray model corresponds to the previously defined **TraysVaporInletPort**.

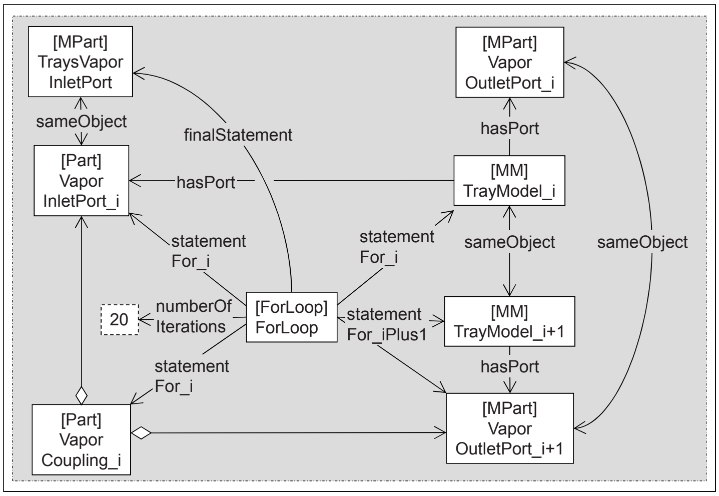

Fig. 9.8: Specification of the repetitive submodel structure

9.1.5 Concept Descriptions

Individual concepts of the module *mathematical_model* are defined below.

Class Descriptions

Constant
A *constant* is a specific *model quantity*, the *model quantity specification* of which has a constant numericalValue in all simulation runs.

Coupling
A *coupling* connects two *model ports* of different *submodels*, thereby defining equality constraints between *model quantities* comprised in the two *model ports*.

Input variable
Input variables represent time- or space-depenent inputs, which have to be specified for dynamic and/or spatially distributed systems.

Lower limit
An *lower limit* is *model quantity specification* which defines an lower bound for the numericalValue of a *model quantity specification*.

Mathematical model
A *mathematical model* is a *model* that uses mathematical language to describe the modeled *system*.

Modeled object
A *system* that is modeled by means of a *model* is denoted as a *modeled object.*
Formal definition: A *modeled object* is a *system* that isModeledBy a *model.*

Model port
A *model port* is a collection of *model quantities* that can participate in a connection with another *mathematical model.* Thus, a *model port* has the function to identify and to bundle the "public" variables of a *mathematical model.* Optionally, a *model port* can be ordered by a *port index.*

Model quantity
A *model quantity* represents a *physical quantity* involved in a *mathematical model*, the *value* of which can be either supplied by the modeler or a computed from an evaluation of the *mathematical model.*
Formal definition: A *model quantity* is either a *state variable* or a *parameter* or a *constant.*

Model quantity specification
A *model quantity specification* specifies a *model quantity* in terms of its numerical value (or limits of its numerical value) and its unit of measurement.

Parameter
A *parameter* is a specific *model quantity* (i.e., an input variable), the *model quantity specification* of which may take different numericalValue in different simulation runs.

Port index
A *port index* orders the *model quantities* comprised in a *model port* by assigning each of them an indexValue. In a *coupling*, *model quantities* with the same indexValue are coupled to each other.

Submodel
A *mathematical model* can be decomposed into *submodels.*
Formal definition: A *submodel* is a direct subsystem of a *mathematical model.*

State variable
State variables constitute the fixed set of unknown variables which have to be computed by the model.
Its *model quantity specification* either indicates the upperLimit and lowerLimit of the *model quantity* (before solving the model) or its numericalValue (after solving the model).

Upper limit
An *upper limit* is a *model quantity specification* which defines an upper bound for the numericalValue of a *model quantity specification.*

Relation Descriptions

correspondsToQuantity
The relation denotes a one-to-one correspondence between a *model quantity* and a *physical quantity* of the *modeled object*.

hasCoupling
The relation indicates a *coupling* between two *submodels* of a *mathematical model*.

hasModelPort
The relation identifies the *model port* of a *mathematical model*.

hasModelVariable
The relation indicates the *model quantities* of a *mathematical model*.

determinesPositionOf
The one-to-one relation between a *port index* and the corresponding *model quantity*.

isIndexOf
The relation isIndexOf points from a *port index* to the associated *model port*.

isOrderedBy
The relation isOrderedBy points from a *model port* to its sorting *port index*.

Attribute Descriptions

indexValue
The attribute indexValue indicates the numerical value of a *port index*.

9.2 Equation System

The ontology module *equation_system* provides concepts for the description of the model equations that constitute a *mathematical model*. The model equations are not explicitly represented, only their *equation system characteristics* are specified. Moreover, the scope of *equation system characteristics* is confined to those characteristics that are of relevance for selecting an appropriate solver and/or solution strategy for the *mathematical model* (cf. Sect. 9.3).
A *mathematical model* can be classified according to different criteria including equation system type, *variables type*, *model representation form*, etc. Following the recommendations for ontology normalization[97] given by Rector (2003), *equation systems* are classified along a two axes (cf. Fig. 9.9): (1) using the equation system type as a differentiating criterion and (2) referring to the linearity of mathematical

[97] More details on this issue can be found in Sect. 4.2.

models, as described in Sect. 4.2.1. The other possible criteria are explicitly modeled as *equation system characteristics*, which are linked to a *mathematical model* via the relation hasCharacteristic (or one of its specializations, cf. upper left corner of Fig. 9.9). Note that some of the *equation system characteristics* can only be assigned to special types of *equation systems*; for instance, *DAE type* only applies to *differential algebraic equation systems*.

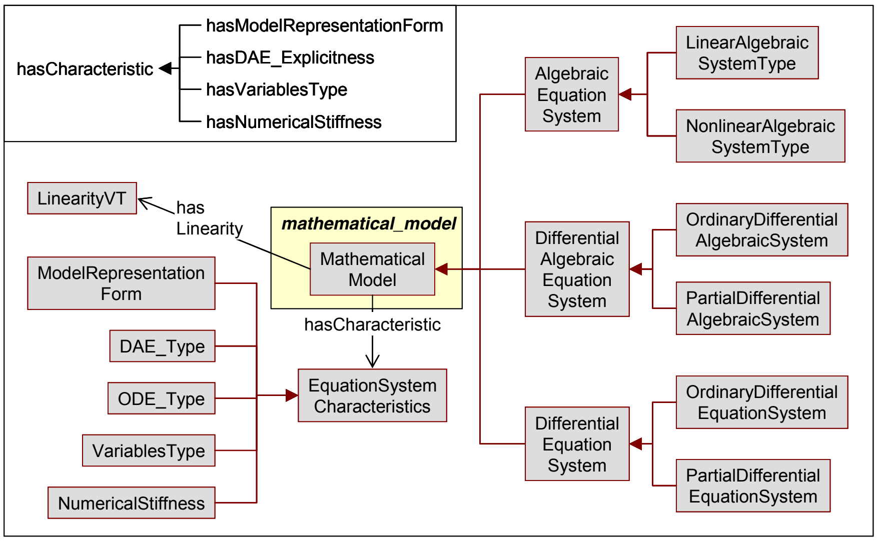
hasCharacteristic
hasModelRepresentationForm
hasDAE_Explicitness
hasVariablesType
hasNumericalStiffness
LinearityVT
has Linearity
mathematical_model
Mathematical Model
ModelRepresentation Form
DAE_Type
ODE_Type
VariablesType
NumericalStiffness
hasCharacteristic
EquationSystem Characteristics
Algebraic Equation System
LinearAlgebraic SystemType
NonlinearAlgebraic SystemType
Differential Algebraic Equation System
OrdinaryDifferential AlgebraicSystem
PartialDifferential AlgebraicSystem
Differential Equation System
OrdinaryDifferential EquationSystem
PartialDifferential EquationSystem

Fig. 9.9: Equation system characteristics

While the meaning of most concepts displayed in Fig. 9.9 should be evident from their names, the concept of *model representation form* requires some explanation. A *mathematical model* may appear in two representation forms, which are termed **open-form** and **closed-form**.

An **open-form** model does not provide a solution method to solve its model equations. A numerical solver needs to be applied to the model to obtain a solution solving the simulation. Hence, the **open-form** model must provide all the information required by the external numerical algorithm to solve the model. For example, a model representing a set of algebraic equations may provide equation residuals and derivatives to a Newton solver. Before an **open-form** model can be successfully solved, it has to be "squared", meaning that the number of its unknown variables must be the same as that of its equations. Among all the *model quantities* of an **open-form** model, those declared as *constants, input variables*, or *parameters* have to be given values (i.e., they need to be assigned a *model quantity specification* with a definite numericalValue) before the model can be evaluated (cf. Sect. 9.1). The other variables are *state variables*. If there are still more *state variables* than equations in a model, it is necessary to assign values to some selected variables (i.e., turn them into *parameters* or *input variables*). Generally, one can freely choose the set of

model variables of an **open-form** model to be specified, as long as the model remains solvable. The values of the remaining variables can be obtained by solving the model.
A **closed-form** model includes an underlying numerical algorithm, which solves its model equations. Thus, it does not require any external solver for obtaining the values of its unknown variables. The "execution" of the **closed-form** model yields the values of a set of selected unknown variables, the so-called outputs, based on the given values of the specified variables. In this process, the algorithm of a **closed-form** model accepts only a fixed set of input variables, and consequently returns a fixed set of output variables. No choice for specifying additional variables is available, as in the case of **open-form** models. Reflected in *model quantity* types, *constants* and *parameters* constitute the fixed set of specified variables, while the *state variables* constitute the fixed set of unknown variables (i.e. output variables).

9.2.1 Usage

The ontology module *equation_system* provides the basic concepts for the identification of mathematical models from a mathematical point of view (e.g., whether it is an ODE or DAE) This identification was primarily of concern in the COGents project (cf. Sect. 12.1.1), where this module was applied to specify the type of mathematical model to search for in various libraries.
Typically, mathematical models may be classified either by means of content (e.g., a mathematical model for a polyethene reactor) or simply by mathematical features, as it is done here. As an example, consider a process engineer who searches for a particular mathematical model, which is supposed to be applied for the calculations of a reactor. Depending on the software (e.g., he might have only a solver for ODEs available), a classification with respect to the characteristics (e.g., ODE type) is extremely helpful to identify the suitable mathematical model.

9.2.2 Concept Descriptions

Individual concepts of the module *equation_system* are defined below. For an extensive description of the introduced individuals, we refer to Morbach et al. (2008j).

Class Descriptions

Algebraic equation system
An *algebraic equation system* is a *mathematical model* which solely consists of algebraic equations.
Formal definition: An *algebraic equation system* is either a *linear algebraic system* or a *nonlinear algebraic system*.

DAE type
Characterizes the explicitness of a *differential algebraic equation system*.
Formal definition: The class *DAE type* is an exhaustive enumeration of the individuals **fully_implicit** and **semi-explicit**.

Differential algebraic equation system
A *differential algebraic equation system* (DAE system) is a *mathematical model* that comprises both algebraic and differential equations.
Formal definition: A *differential algebraic equation system* is either an *ordinary differential algebraic system* or a *partial differential algebraic system*.

Differential equation system
A *differential equation system* is a *mathematical model* that solely consists of differential equations.
Formal definition: A *differential equation system* is either an *ordinary differential equation system* or a *partial differential equation system*.

Equation system characteristics
The *equation system characteristics* characterize the model equations of a *mathematical model*.

Linear algebraic system type
A *linear algebraic system type* is an *algebraic system* which contains only linear equations.
Formal definition: A *linear algebraic system type* is an *algebraic system* that is characterized as **linear**.

Linearity VT
Linearity VT characterizes the linearity of a mathematical model.
Formal definition: Linearity is an exhaustive enumeration of the individuals **linear** and **nonlinear**.

Model representation form
A *mathematical model* may appear in two forms, as indicated by the *model representation form*:

- An **open-form** model is solved by an external algorithm. One can freely choose the inputs and outputs of the **open-form** model.

- A **closed-form** model includes an underlying numerical algorithm that solves the model equations. The algorithm accepts only a fixed set of input variables, and consequently returns only a fixed set of output variables.

Formal definition: The class *model representation form* is an exhaustive enumeration of the individuals **open-form** and **closed-form**.

Nonlinear algebraic system type
Formal definition: A *nonlinear algebraic system type* is an *algebraic equation system* that is characterized as **nonlinear**.

Numerical stiffness
In mathematics, stiff equations are equations where certain implicit methods, in particular BDF, perform better, usually tremendously better, than explicit ones (Hairer and Wanner 1996).
Formal definition: The class *numerical stiffness* is an exhaustive enumeration of the individuals **stiff** and **nonstiff**.

ODE_type
Characterizes the explicitness of an *ordinary differential equation system*, which can be given in **implicit_formulation** or **explicit_formulation**.
Formal definition: *ODE_types* is an exhaustive enumeration of the individuals **implicit_formulation** and **explicit_formulation**.

Ordinary differential algebraic system
An *ordinary differential algebraic system* comprises algebraic equations as well as ordinary differential equations, but no partial differential equations.

Ordinary differential equation system
An *ordinary differential equation system* (ODE system) is a *differential equation system* which solely consists of ordinary differential equations.

Partial differential algebraic system
A *partial differential algebraic system* is a *differential algebraic equation system* which comprises both partial differential equations and algebraic equations.

Partial differential equation system
A *partial differential equation system* (PDE system) is a *differential equation system* which consists of partial differential equations.

Variables type
A *variables type* indicates whether the *model quantities* of a *mathematical model* are all continuous, all discrete, or partly continuous and partly discrete.
Formal definition: The class *variables type* is an exhaustive enumeration of the individuals **continuous**, **discrete**, and **mixed**.

Relation Descriptions

hasDAE_Type
Indicates an *equation system characteristic* of type *DAE type.*

hasLinearity
Refers from a *mathematical model* to a *linearity value type*.

hasModelRepresentationForm
Indicates an *equation system characteristic* of type *model representation form*.

hasNumericalStiffness
Indicates an *equation system characteristic* of type *numerical stiffness*.

hasODE_Type
Indicates an *equation system characteristic* of type *ODE type.*

hasVariablesType
Indicates an *equation system characteristic* of type *variables type*.

Attribute Descriptions

differentialIndex
The attribute represents the differential index of an ordinary *differential algebraic equation system*, as defined by Gear and Petzold (1984) or of a partial differential algebraic system, as defined by Martinson and Barton (2000).

differentialOrder
The attribute differentialOrder denotes the order of a differential equation, which is defined as the order of the highest derivative of a *model quantity* appearing in the differential equation.

9.3 Numerical Solution Strategy

In this ontology module, strategies for solving *mathematical models* are defined. At present, it is confined to numerical solution strategies only. A classification of numerical solution techniques is given, and the ability of a strategy to solve a particular type of *mathematical model* is explicitly specified. The major concepts are shown in Fig. 9.10. A *model solution strategy* solves a *mathematical model*; the subclasses of *model solution strategy* represent different types of numerical algorithms, which are specifically designed to solve a certain type of *mathematical model* with certain *equation system characteristics*. To this end, a *model solution strategy* may apply some other, specialized *model solution strategy*. So far, only numerical solution strategies have been considered in OntoCAPE, but symbolic/analytical solution methods could be added in an analogous manner.

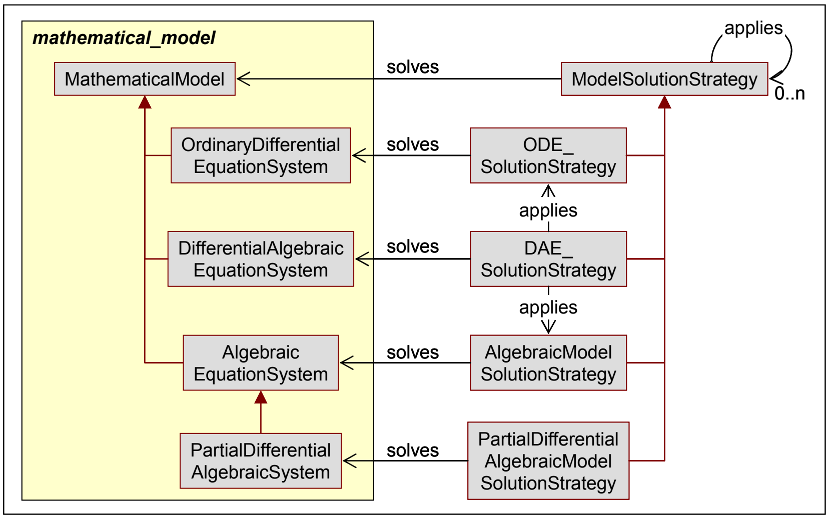
mathematical_model
MathematicalModel
OrdinaryDifferential EquationSystem
DifferentialAlgebraic EquationSystem
Algebraic EquationSystem
PartialDifferential AlgebraicSystem
solves
ModelSolutionStrategy
applies
0..n
ODE_ SolutionStrategy
DAE_ SolutionStrategy
AlgebraicModel SolutionStrategy
PartialDifferential AlgebraicModel SolutionStrategy

Fig. 9.10: Numerical solution strategy.

Fig. 9.11 shows the refinement of class *algebraic model solution strategy*. An exemplary *linear algebraic model solution strategy* is Gauss-elimination, an example of a *nonlinear algebraic model solution strategy* is Newton's method.

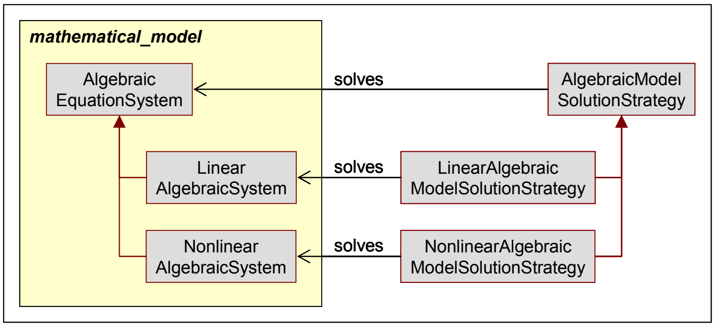
mathematical_model
Algebraic EquationSystem
Linear AlgebraicSystem
Nonlinear AlgebraicSystem
solves
AlgebraicModel SolutionStrategy
LinearAlgebraic ModelSolutionStrategy
NonlinearAlgebraic ModelSolutionStrategy

Fig. 9.11: Types of *algebraic model solution strategies*

An *ODE solution strategy* can be further characterized by indicating if the algorithm is a **one-step_method** (e.g., the classical Runge-Kutta methods) or a **multi-step_method** (e.g., the Adams-Bashforth methods). Moreover, it can be specified whether the algorithm is a *solution strategy for explicit ODEs* or *implicit ODEs* (cf. Fig. 9.12.

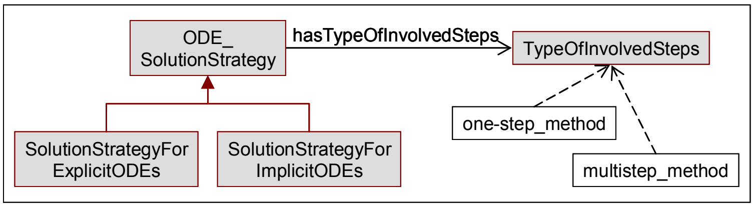

Fig. 9.12: Further specification of *ODE solution strategy*

9.3.1 Concept Descriptions

Individual concepts of the module *numerical_solution_method* are defined below.

Class Descriptions

Algebraic model solution strategy
An *algebraic model solution strategy* is a *model solution strategy* for solving *algebraic equation systems*.

DAE solution strategy
A *DAE solution strategy* is a *model solution strategy* for solving *differential algebraic equation systems*. Examples are implicit Runge-Kutta, BDF, etc.

Linear algebraic model solution strategy
A *linear algebraic model solution strategy* is a *model solution strategy* for solving *linear algebraic systems*. An example is Gauss elimination.

Model solution strategy
A *model solution strategy* is a (typically numerical) algorithm that can be used to solve *mathematical models*.

Nonlinear algebraic model solution strategy
A *nonlinear algebraic model solution strategy* is a *model solution strategy* for solving *nonlinear algebraic systems*. An example is Newton's method.

ODE solution strategy
An *ODE solution strategy* is a *model solution strategy* for solving *ordinary differential equation systems*. An example is the Euler method.

Partial differential algebraic model solution strategy
A *partial differential algebraic model solution strategy* is a *model solution strategy* for solving *partial differential algebraic systems*. An example is a finite element method.

Solution strategy for explicit ODEs
A *solution strategy for explicit ODEs* is used to solve *ordinary differential equation systems* that are given in an **explicit_formulation**. Examples are explicit Euler, explicit Runge-Kutta, etc.

Solution strategy for implicit ODEs
A *solution strategy for implicit ODEs* is used to solve *ordinary differential equation systems* that are given in an **implicit_formulation**. Examples are implicit Euler, implicit Runge-Kutta, etc.

Type of involved steps
A *type of involved step* denotes whether an *ODE solution strategy* is a **one-step_method** or a **multi-step_method**.

- A **one-step_method** characterizes an *ODE solution strategy* that uses information of one integration step. Examples are various Runge-Kutta methods.
- A **multi-step_method** characterizes an *ODE solution strategy* that uses information of multiple integration steps. Examples are Adams, BDF, etc.

Formal definition: Exhaustive enumeration of the individuals **one-step_method** and **multi-step method**.

Relation Descriptions

applies
A *model solution strategy* may apply some other, specialized *model solution strategy* (e.g., for initialization, solving corrector equation, solution of a subproblem, etc.).

hasTypeOfInvolvedSteps
Indicates the *type of involved steps* of an *ODE solution strategy*.

solves
The relation indicates the type of *mathematical model*, for the solution of which a particular *model solution strategy* is designated.

Attribute Descriptions

handlesDifferentialIndexUpTo
A *DAE solution strategy* can only solve *differential algebraic equation systems* up to a certain differentialIndex. This restriction is specified through the attribute handlesDifferentialIndexUpTo.

9.4 Cost Model

The ontology module *cost_model* establishes some *cost models* for predicting the (investment) *costs* of chemical plants. A *cost model* is a special type of *economic performance model*, which models the *economic performance* of a *chemical process system*.
At present, the module merely holds a number of models for the estimation of the *fixed capital investment* (cf. Sect. 8.7.1.2); in the future, further types of *cost models* are to be added, and the existing ones are to be specified in detail. Fig. 9.13 gives an overview on the *cost models* defined so far. For an explanation of the individual classes, we refer to the concept definitions below.

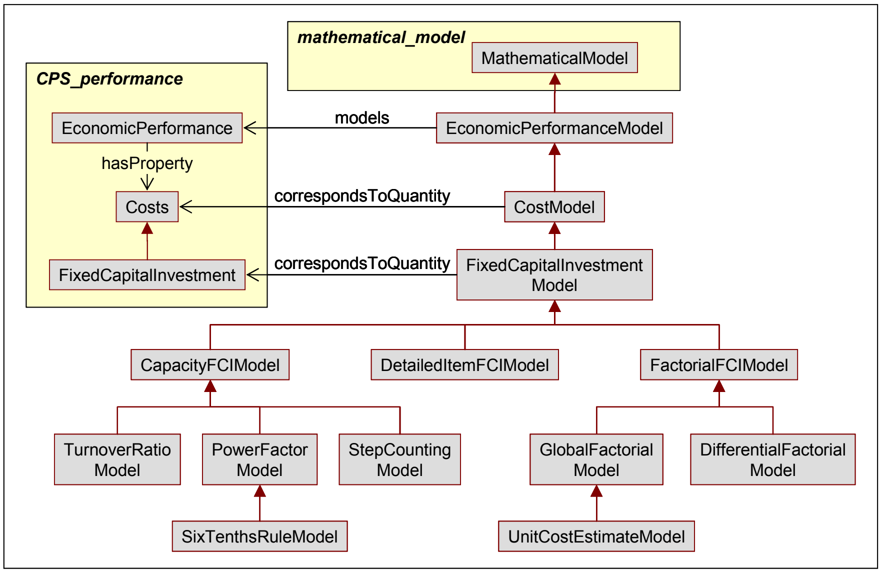

Fig. 9.13: Models for estimating the fixed capital investment

9.4.1 Concept Descriptions

Individual concepts of the module *cost_model* are defined below.

Class Descriptions

Capacity FCI model
Capacity FCI models are based on *fixed capital investments* of past design projects that are similar to the current *chemical process system*. Besides, some relating factors (e.g., the turn-over ratio), exponential power ratios, or more complex relations are given.

Cost model
A *cost model* is a *mathematical model* to estimate the investment costs of a *chemical process system*.
Formal definition: A *cost model* is an *economic performance model* that has a *model quantity* which corresponds to the quantity of *costs*.

Detailed-item FCI model
A *detailed-item FCI model* requires careful determination of all individual direct and indirect cost items. For such models, extensive data and large amounts of engineering time are necessary. Therefore, this type of estimate is almost exclusively prepared by contractors bidding on complete and all-inclusive work from finished drawings and specifications.

Differential factorial model
Within *differential factorial models*, different factors are used for estimating the costs of the *fixed capital investment*. Examples are modular estimate models, where individual modules consisting of a group of similar items are considered separately, and their costs are then summarized (Guthrie 1969).

Economic performance model
An *economic performance model* models the *economic performance* of a *chemical process system*.
Formal definition: An *economic performance model* is a *mathematical model* that models some *economic performance*.

Factorial FCI model
Factorial FCI models rely on the fact that the percentages of the different costs within the *fixed capital investment* are similar for different *chemical process systems*. Based on one or several known costs (for example the *equipment costs*), the *fixed capital investment* is estimated using some factors that are derived from cost records, published data, and experience.

Fixed capital investment model
Fixed capital investment models (FCI models) are *mathematical models* that are used to estimate the *fixed capital investment* of a chemical process system.
Formal definition: A *fixed capital investment model* is a *cost model* which has a *model quantity* that correspondsToQuantity of *fixed capital investment*.

Global factorial model
A *global factorial model* estimates the *fixed capital investment* by multiplying the basic equipment cost by some factor. This factor depends, among other things, on the type of chemical process involved, on the required materials of construction, and on the location of the *chemical process system realization*. Examples for global factors are the ones proposed by (Lang 1947). This model can be extended to calculate the *total capital investment*.

Power factor model
The *power factor model* relates the *fixed capital investment* of a new *chemical process system* to the one of similar, previously constructed systems by an exponential power ratio (cf. Peters and Timmerhaus 1991).

Six-tenths rule model
The *six-tenths rule model* is a *power factor model* with x=0.6.

Step counting model
Step counting models are based on the assumption that the *fixed capital investment* can be estimated from the number of *process steps* (depending on the specific approach, *composite process steps* or *unit operations* and *reactions* are used), multiplied with the costs per *process step* and some correcting factors. The costs of the *process steps* are estimated from their capacity and some other factors (Vogt 1996).

Turnover ratio model
The *turnover ratio model* is a fast evaluation method for order-of-magnitude estimates. The turnover ratio is defined as the ratio of gross annual sales to *fixed capital investment*. Values of turnover ratios for different types of chemical processes are for example given by Schembra (1991) and Vogt (1996).

Unit-cost estimate model
Unit-cost estimate models are based on detailed estimates of the main *purchase costs for system realization* (either obtained from quotations or from cost records and published data).

9.5 Process Model

As an extension to *mathematical_model*, the ontology module *process_model* enables the definition of specialized *mathematical models* for the domain of chemical engineering. Such models, which model either *process units* (cf. Sect. 8.1.1) or *materials* (cf. Sect. 7.1) or subsystems of these, are called *process models* (cf. Fig. 9.14). The *modeling principle* based on which a *process model* is developed may also be indicated.

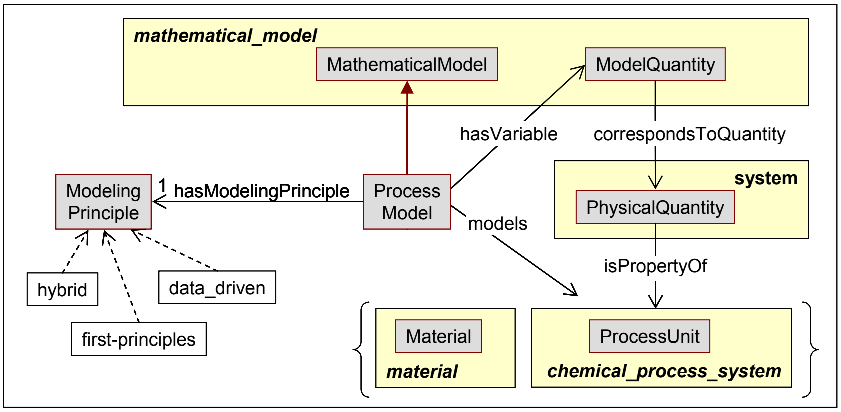

Fig. 9.14: Overview on *process_model*

A *process model* may contain other *process models*, particularly the established *laws* and *property models* (cf. Fig. 9.15). Neither *laws* nor *property models* are self-contained models, but form part of an overall *process model*, where they represent mathematical correlation between designated *model quantities*.

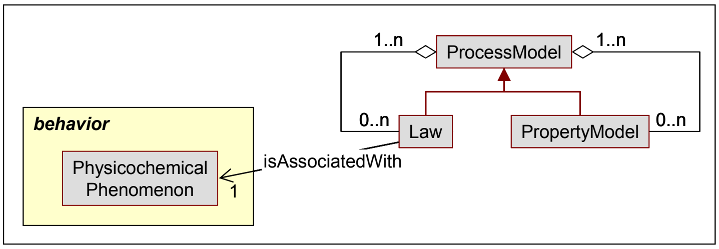

Fig. 9.15: Laws and property models

A *law* constitutes the mathematical representation of a scientific law, such as the law of energy conservation (cf. Sect. 9.6). Each *law* can be associated with a *physicochemical phenomenon* (cf. Sect. 8.6.1.6). The former gives a quantitative, the latter a qualitative description of a certain physical behavior. The correspondence between a *law* and a *physicochemical phenomenon* can be stated via the relation isAssociatedWith, as indicated in Fig. 9.15. Moreover, the *model quantities* of the *law* correspond to the *physical quantities* that are influenced by the *physicochemical phenomenon*, as exemplarily shown in Fig. 9.16.

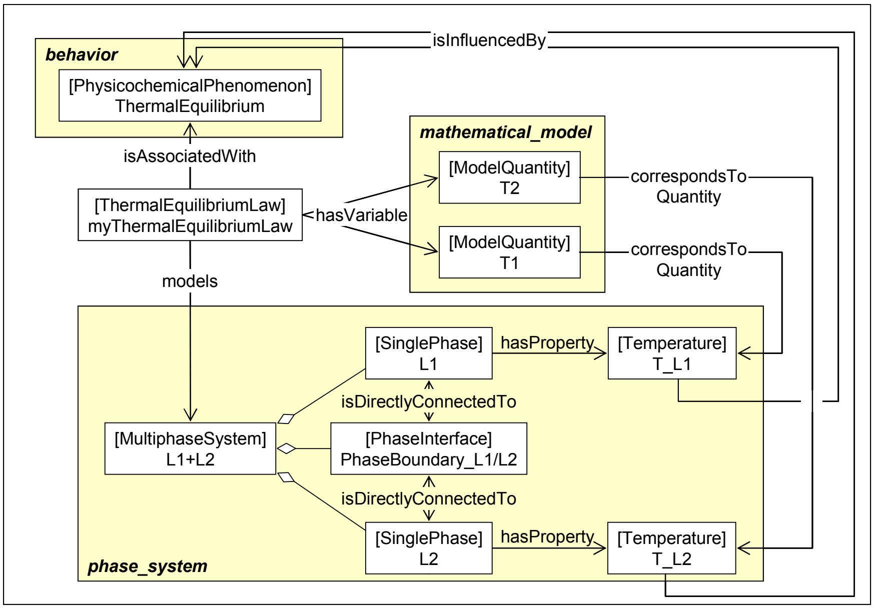

Fig. 9.16: Exemplary law modeling thermal equilibrium

A *property model* represents a mathematical correlation for the computation of one designated *model quantity*, which corresponds to one specific *physical quantity*. An example is given in Fig. 9.17: An *activity coefficient model* constitutes a correlation for the computation of *activity coefficients*. Consequently, an *activity coefficient model* comprises, among others, a *model quantity* which corresponds to an *activity coefficient*.

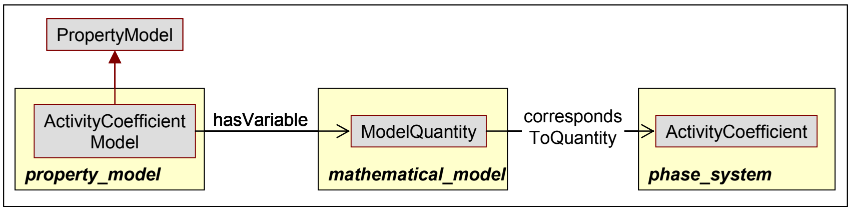

Fig. 9.17: Exemplary property model

9.5.1 Concept Descriptions

Individual concepts of the module *process_model* are defined below. For a description of the instances of *modeling principle*, we refer to Morbach et al. (2008j).

Class Descriptions

Law
A *law* constitutes the mathematical representation of a scientific law. It usually forms part of an overall *process model*.

Modeling principle
A *modeling principle* represents the principle on which the development of *process model* is based.

- Following the **data_driven** *modeling principle*, a *process model* is derived from the *values* of the *properties* of a *modeled object*. Examples of this type of models are neural network models.
- Following the **first-principles** *modeling principle*, the *process model* is based on established physical laws and mechanisms.
- A **hybrid** *modeling principle* applies both the **first-principles** and the **data_driven** approach.

Formal definition: *Modeling principle* is defined by an exhaustive enumeration of the individuals **data_driven**, **first-principles**, and **hybrid**.

Process model
A *process model* is a *mathematical model* that models a *process unit* or *material* (or *subsystems* of these).

Property model
A *property model* forms part of an overall *process model*. It represents a mathematical correlation for the computation of a designated *model quantity*, which corresponds to a specific *physical quantity*. Examples are vapor pressure correlations or activity coefficient models.

Relation Descriptions

hasModelingPrinciple
Indicates the *modeling principle* on which a *process model* is based.

isAssociatedWith
The relation denotes a correspondence between a *law* and a *physicochemical phenomenon*. The former gives a quantitative, the latter a qualitative description of a certain physical behavior.

9.6 Laws

The ontology module *laws*, located on the Application-Oriented Layer of OntoCAPE, introduces a hierarchical collection of *laws* that are frequently used in process modeling. The law hierarchy shown was originally presented by Marquardt (1995). A selection of the taxonomy related to physicochemical laws is given in Fig. 9.18 - Fig. 9.22. The high-level concepts include *balance laws*, *constitutive laws*, and *constraints* as shown in Fig. 9.18.

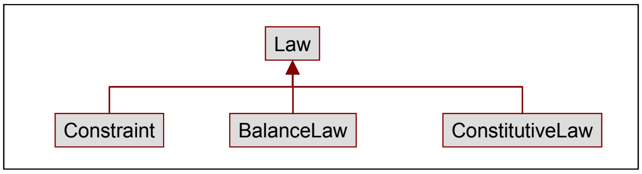

Fig. 9.18: High-level classification of *laws*

Balance laws generally represent the change of an extensive quantity in *process models*. This typically includes balances for total mass and mass of species in a mixture (*mass balance law*), for momentum (*momentum balance law*), for total or any other kind of energy (*energy balance law*), and for the particle number in case of a particulate system (*population balance laws*), as depicted in Fig. 9.19.

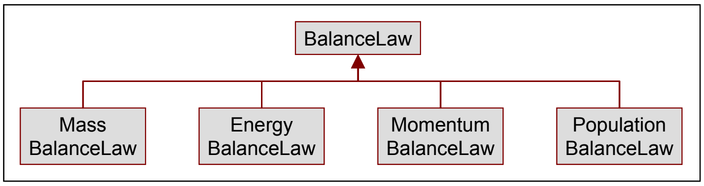

Fig. 9.19: Specialization of *balance laws*

However, balance equations do not suffice to describe the behavior related to a *process model*. Thus, *constitutive laws* have to be added in order to determine the *process model* completely. Three types of *constitutive laws* may be distinguished (compare Sect. 8.6.1.7 and Fig. 9.20), including *generalized flux laws*, *phenomenological coefficient law*, and *thermodynamic state function law*. *Generalized flux laws* describe the contribution to any kind of *balance law*. These laws are typically composed of a phenomenological coefficient and a driving force determined by some thermodynamic state function which are modeled by *phenomenological coefficient laws* and *thermodynamic state function laws*.

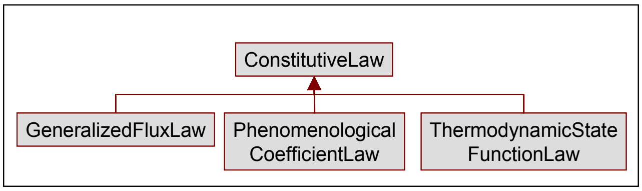
ConstitutiveLaw
GeneralizedFluxLaw
Phenomenological CoefficientLaw
ThermodynamicState FunctionLaw

Fig. 9.20: Specialization of *constitutive law*

In Fig. 9.21, the specializations of *generalized flux law* are presented, including some further specializations associated to transport and exchange phenomena. These specific *laws* have to be considered before a concrete *process model* can be generated.

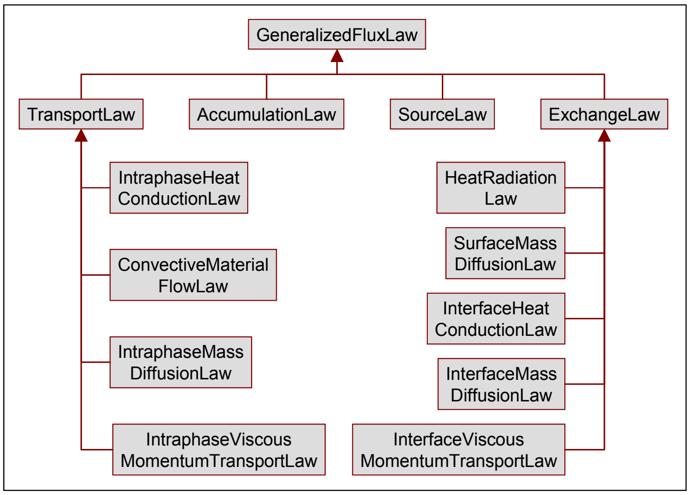
GeneralizedFluxLaw
TransportLaw
AccumulationLaw
SourceLaw
ExchangeLaw
IntraphaseHeat ConductionLaw
ConvectiveMaterial FlowLaw
IntraphaseMass DiffusionLaw
IntraphaseViscous MomentumTransportLaw
HeatRadiation Law
SurfaceMass DiffusionLaw
InterfaceHeat ConductionLaw
InterfaceMass DiffusionLaw
InterfaceViscous MomentumTransportLaw

Fig. 9.21: Specialization of *generalized flux laws*

Finally, *constraints* describe all kinds of (algebraic) relations between *process quantities* which – literally or by assumption – have to hold at any time. Typical examples are *volume constraints* or *equilibrium constraints*, which specialize the class *constraint* in Fig. 9.18.

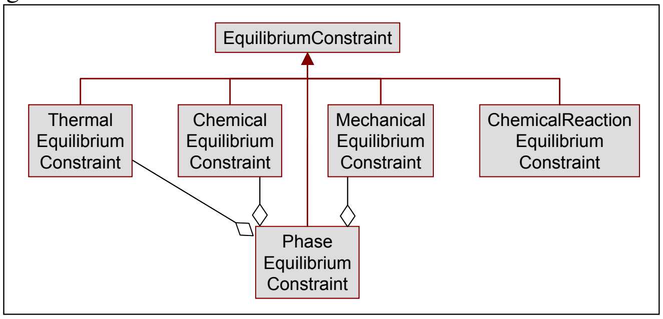
EquilibriumConstraint
Thermal Equilibrium Constraint
Chemical Equilibrium Constraint
Mechanical Equilibrium Constraint
ChemicalReaction Equilibrium Constraint
Phase Equilibrium Constraint

Fig. 9.22: Specialization of *equilibrium constraints*

In Fig. 9.22, *equilibrium constraints* are specialized into *thermal equilibrium*, *chemical equilibrium*, and *mechanical equilibrium* on the one hand, which refer to equal temperature, pressure, or chemical potential in adjacent phases. *Phase equilibrium* and *chemical reaction equilibrium* are considered on the other hand. It shows that *phase* *equilibrium* is just an aggregation of thermal, chemical, and mechanical equilibrium. *Chemical reaction equilibrium* refers to a network of chemical reactions residing in a *single phase*, where all forward reaction rates equal the backward reaction rates. The constraint *phase equilibrium* can also be formulated in various alternative ways which are fully equivalent[98].

Currently, only the hierarchy of *laws* and the associated *physicochemical phenomena* (cf. Sect. 8.6.1.6) are modeled. In future extensions of this ontology module, one may add further definitions and constraints in order to specify a *law*'s *model quantities* and their corresponding *physical quantities*.

For an exhaustive description of all concepts used in the module *laws*, we refer to Morbach et al. (2008j).

9.7 Property Models

The ontology module *property_models*, which is located on the Application-Oriented Layer of OntoCAPE, provides a hierarchically ordered collection of frequently used *property models*. As indicated in Fig. 9.23, a *property model* might be one of the following:

– A *chemical kinetics model*, which specifies how to calculate the rate coefficient of a homogenous or heterogeneous reaction.
– A *phase interface transport property model*, which provides a correlation for computing certain *phase interface transport properties*.
– A *thermodynamic property model*, which indicates the correlation between certain *intensive thermodynamics state variables* (cf. Sect. 7.3.3) and *intra-phase transport properties*.

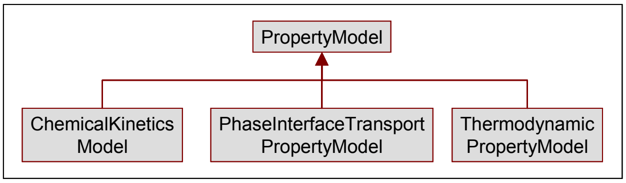
PropertyModel
ChemicalKinetics Model
PhaseInterfaceTransport PropertyModel
Thermodynamic PropertyModel

Fig. 9.23: High-level classification of *property models*

The classification of these specialized *property models* is given in Fig. 9.24 - Fig. 9.27:

[98] For example, a number of alternative formulations exist for chemical equilibrium, such as the equality of chemical potentials between two phases and the equality of fugacities between two phases. The equality of fugacities can further be written in different forms depending on what property models are to be used in conjunction with the law for the chemical equilibrium.

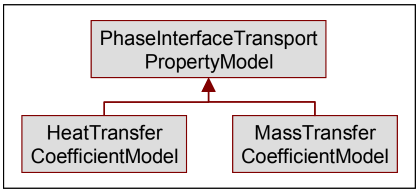
PhaseInterfaceTransport PropertyModel
HeatTransfer CoefficientModel
MassTransfer CoefficientModel

Fig. 9.24: Some *phase interface transport property models*

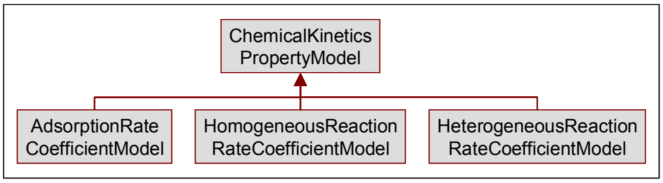
ChemicalKinetics PropertyModel
AdsorptionRate CoefficientModel
HomogeneousReaction RateCoefficientModel
HeterogeneousReaction RateCoefficientModel

Fig. 9.25: Some *chemical kinetics models*

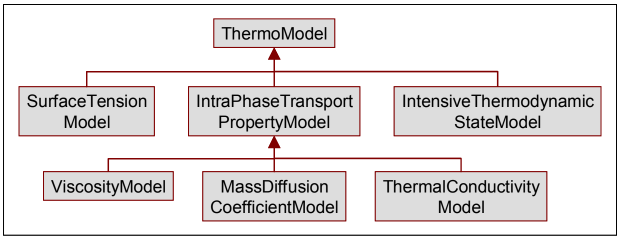
ThermoModel
SurfaceTension Model
IntraPhaseTransport PropertyModel
IntensiveThermodynamic StateModel
ViscosityModel
MassDiffusion CoefficientModel
ThermalConductivity Model

Fig. 9.26: Some *thermodynamic property models*

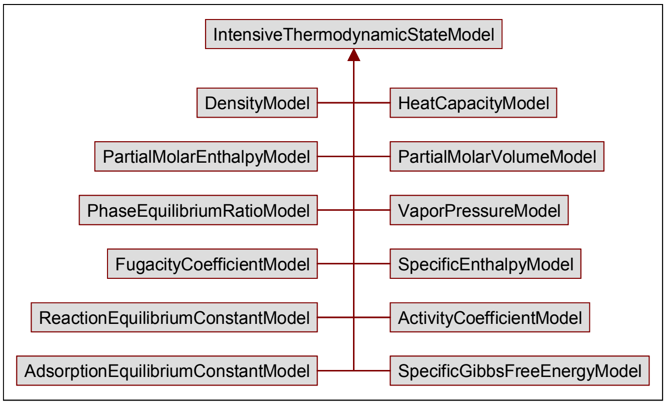
IntensiveThermodynamicStateModel
DensityModel
HeatCapacityModel
PartialMolarEnthalpyModel
PartialMolarVolumeModel
PhaseEquilibriumRatioModel
VaporPressureModel
FugacityCoefficientModel
SpecificEnthalpyModel
ReactionEquilibriumConstantModel
ActivityCoefficientModel
AdsorptionEquilibriumConstantModel
SpecificGibbsFreeEnergyModel

Fig. 9.27: Some *intensive thermodynamic state models*

Exemplarily, the definition of the class *density model* is shown in Fig. 9.28: A *density model* has some *model quantities*, one of which corresponds to a *physical quantity* of type *density*. The other *property models* are defined analogously.

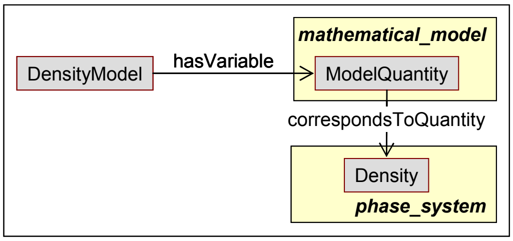

Fig. 9.28: Definition of the class *density model*

For an exhaustive description of all concepts used in the module *property_models*, we refer to Morbach et al. (2008j).

9.8 Process Unit Models

The ontology module *process_unit_models*, located on the application-oriented level of OntoCAPE, provides a collection of *mathematical models* that model the behavioral aspect of *process units*.
Please note that this module is introduced not for the purpose of providing a full account on this topic, but rather for suggesting a principle for defining various types of *process unit models* and illustrating the principle by means of only a few examples.
These exemplary *property unit models* are classified according to the modeled *process units* (Wiesner et al. 2008a): a *chemical reactor model* models a *chemical reactor behavior*, a *flash unit model* models a *flash unit behavior*, etc. (cf. Fig. 9.29).

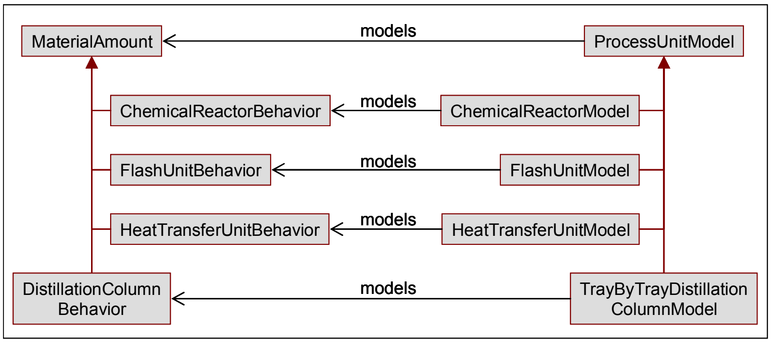

Fig. 9.29: High-level classification of *process unit models*

Beyond this high-level classification, the ontology module comprises some special types of *process unit models*. Fig. 9.30 exemplarily shows the definition of a *CSTR model*: A *CSTR model* is a *chemical reactor model* that models a *chemical reactor behavior* with the *physicochemical phenomenon* of phenomenon **ideally_mixed**. Furthermore, the *CSTR model* is a **first-principles** model and incorporates the following *laws*: *energy conservation law*, *mass conservation law*, and *reaction kinetics law*.

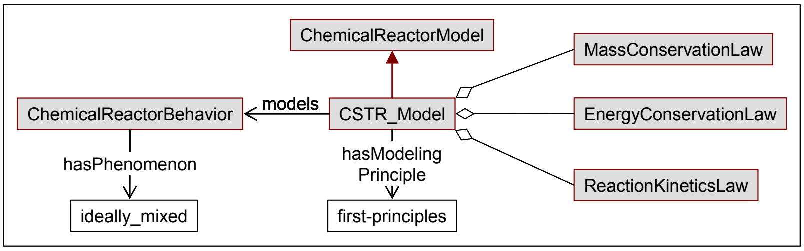

Fig. 9.30: Definition of the class *CSTR model*

Currently, the ontology module provides only a few of such specialized *process unit models*. In the future, it is to be extended to offer a substantial library of *process unit models*. As for now, the module merely provides the framework for establishing such a library.

9.9 References

Gear CW, Petzold L (1984) ODE methods for the solution of differential/algebraic systems. *Trans. Society Computer Simulation* **1**:27–31.

Guthrie K (1969) Data and techniques for preliminary capital cost estimation. *Chem. Eng. (New York)* **24** (3):114-142.

Hairer E, Wanner G (1996) *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*, Springer, Berlin.

Ion P, Miner R, eds. (1999) *Mathematical Markup Language (MathML) 1.01 Specification*. W3C Recommendation, revision of 7 July 1999. Online available at http://www.w3.org/TR/REC-MathML/. Accessed April 2007.

Lang HJ (1947) Engineering approach to preliminary cost estimates. *Chem. Eng. (New York)*:130-133.

Lloyd CM, Halstead MDB, Nielsen PF (2004) CellML: its future, present and past. Progress in Biophysics and Molecular Biology **85** (2-3):433-450.

Martinson WS, Barton PI (2000) A differentiation index for partial differential-algebraic equations. SIAM *J. Sci. Comput.* **21**:2295–2315.

Morbach J, Yang A, Marquardt W (2008j) *OntoCAPE 2.0 – Mathematical Models*. Technical Report (LPT-2008-28), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Peters MS, Timmerhaus KD (1991) *Plant Design and Economics for Chemical Engineers*, McGraw-Hill, New York.

Schembra M (1991) *Daten und Methoden zur Vorkalkulation des Anlagekapitalbedarfs von Chemieanlagen*. PhD thesis, Technische Universität Berlin.

Vogt M (1996) *Neuere Methoden der Investitionsrechnung in der Chemischen Industrie*. Diploma thesis, Technische Universität Berlin.

von Wedel L (2002) *CapeML – A Model Exchange Language for Chemical Process Modeling*. Technical Report (LPT-2002-16), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Wiesner A, Morbach J, Bayer B, Yang A, Marquardt W (2008a) *OntoCAPE 2.0 – Chemical Process System*. Technical Report (LPT-2008-29), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer =LPT-2008-29.

10 Design Principles of OntoCAPE

Concluding the description of OntoCAPE, we will subsequently present the major principles according to which the ontology has been designed. Basically, design principles are objective criteria for guiding and evaluating the design decisions made during ontology development (Gruber 1995). A number of design principles for information modeling in general, and ontology engineering in particular, have been suggested in the literature (e.g., Gruber 1995; Fox and Grüninger 1998; Arpírez et al. 1998; Chandrasekaran et al. 1999; Gómez-Pérez et al. 2004; Rector et al. 2004; Smith 2006; and others). Compliance with these acknowledged principles is a credible indicator for the quality of an ontology.

The design of OntoCAPE has been guided by the following major principles: *coherence*, *conciseness*, i*ntelligibility*, *adaptability*, *minimal ontological commitment*, and *efficiency*. These six principles subsume the plethora of recommendations stated in the literature. In the subsequent sections, these principles will be addressed individually: We will define the meaning of each principle, discuss its general implication on ontology design, and describe its realization in OntoCAPE.

10.1 Coherence

The principle of *coherence*, in the literature also known as *soundness* or *consistency*, stipulates that the ontological definitions (i) are individually sound and (ii) do not contradict each other. This principle applies to both the formal and the informal specification of the ontology.

The coherence of the formal specification can be checked by means of computer programs:

- Ontology editors like Protégé (Stanford 2008) provide functionality for *syntax checking* and *sanity testing*, through which most of the inadvertent inconsistencies can be detected and resolved. A number of such tests have been run on OntoCAPE; typical errors found this way are relation properties that do not match the properties of the relation's inverse, or local range restrictions on a superclass that are narrower than the range restrictions assigned to its subclasses.
- Reasoners like RacerPro (Racer Systems 2007) or FaCT++ (Tsarkov and Horrocks 2007) allow for more sophisticated consistency testing: They do not only detect inconsistencies between the stated axioms, but also check for contradictory conclusions that can be inferred from these axioms. However, there are limitations with respect to the size of the ontology: At present (as of 2009), neither FaCT++ nor RacerPro are able to test Onto-

CAPE as a whole[99]. To enable testing, we had to reduce the problem size by splitting OntoCAPE into parts of about half the size of the ontology. Subsequently, each part has been individually tested for consistency. By creating different overlapping partitions, consistency between the individual parts could, at least to some extent, be validated. Note that the need to partition the ontology does not necessarily contradict the applicability of the ontology for real world applications. Typically, only a subset of the ontology modules is required for particular tasks. These subsets of modules have been tested successfully.

However, the computerized proof of coherence is only conclusive if a sufficient number of the ontological definitions are formally stated as axioms; otherwise, essential information is inaccessible to the reasoner, and consequently the reasoner cannot come to a significant conclusion when processing the ontology. Hence, one should strive to axiomatize as many ontological definitions as possible[100]. In this respect, three types of axiomatizations are of particular importance and should be applied wherever appropriate:

- Firstly, defined classes are preferred over primitive classes (Gruber 1995; Rector et al. 2004) since the latter do not explicitly state the conditions for membership, and thus the reasoner lacks vital information for evaluating their consistency.
- Secondly, siblings[101] should generally be declared to be mutually disjoint; otherwise, the classes are assumed to overlap, which often causes unwanted effects and leads to false conclusions (Rector et al. 2004). For the same reasons, the instances of a common class should be stated to be mutually distinct.
- Thirdly, due to the open world assumption made by DL reasoners, definitions must be explicitly "closed off" (Rector et al. 2004) in order to tighten their possible interpretations. The definition of a relation is closed off by means of local range restrictions; the definition of a class may be closed off by declaring the class to be an exhaustive enumeration of its siblings or instances (Arpírez et al. 1998; Gómez-Pérez et al. 2004; Rector 2005).

Compliant with the above recommendations, the ontological terms of OntoCAPE are formally defined by more than a thousand[102] axioms. As a rule, siblings have

[99] Testing was performed on a machine with a 2.66 GHz dual quad-core processor and 8 GB CPU memory. When the ontology was processed as a whole, either the computer ran out of memory, or the computation was aborted after several hours without any concluding result.

[100] However, if the ontology is too tightly constrained by axiomatic definitions, it violates the principle of minimal ontological commitment (cf. Sect. 10.5).

[101] Siblings are the direct subclasses of a common parent class.

[102] Cf. 12.4.1 for some statistical data about the type and number of axioms in OntoCAPE. These data clearly show the benefits resulting from the axiomatization.

been declared to be mutually disjoint, and the instances of a common class are declared to be different from each other. Closure axioms have been defined wherever appropriate – either in form of local or global range restrictions on relations, or by exhaustive decomposition of classes into siblings or instances. The ratio of defined to primitive classes is roughly 2:5, which constitutes a reasonable trade-off between the principles of coherence and intelligibility on the one hand, and minimal ontological commitment on the other (cf. Sects. 10.3 and 10.5).

Still, there will always be some aspects of an ontology that cannot be formally represented – either due to a lack of expressiveness of the modeling language or due to efficiency considerations – and thus must be described informally. According to Gruber (1995), "coherence should also apply to the concepts that are defined informally, such as those described in natural language documentation and examples. If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent". Obviously, incoherencies of this type can only be found by manual inspection. A number of them were resolved during a peer review of the informal specification of OntoCAPE 1.0 (cf. Sect. 11.1.3); further inconsistencies were uncovered when the document was revised to create the informal specification for OntoCAPE 2.0. Nevertheless, it is highly probable that a number of inconsistencies currently remain undetected; they will be revealed eventually through the continuous reuse of the ontology in new fields and applications.

10.2 Conciseness

The principle of *conciseness*, a.k.a. *minimality* or *minimization*, demands (i) to reduce the number of vocabulary terms to the necessary minimum and (ii) to avoid redundancy with respect to axiomatic definitions. A concise ontology is easier to understand, easier to apply, and easier to maintain; thus, conciseness enhances the intelligibility (cf. Sect. 10.3), usability (cf. Sect. 1.3), and adaptability (cf. Sect. 10.4) of an ontology.

As for case (ii), it includes both the explicit redundancies between definitions and the implicit redundancies that can be inferred from the explicitly stated axioms (Gómez-Pérez et al. 2004). Several test criteria have been suggested for detecting redundant axioms (e.g., Gómez-Pérez 2001; Seipel and Baumeister 2004). Some of them are implemented in ontology editors: For instance, the aforementioned ontology editor Protégé checks for class-embedded axioms, which re-implement a restriction that has already been defined on a superclass; moreover, it searches for cardinality constraints that specify a minimal cardinality of zero. Thanks to the testing functionality provided by Protégé, these types of redundancies have been eliminated in OntoCAPE 2.0.

However, the above tests are only capable of detecting the rather obvious cases of redundancies. The majority of redundancies in an ontology, particularly those of

case (i), are modeling issues, which can only be found by manual inspection. Hence, conciseness in the sense of (i) can only be achieved gradually, by continuous revision and reengineering of the ontology. This matter has been one of the major motivations for the development of version 2.0 of OntoCAPE. Accordingly, the number of vocabulary terms required to represent certain concepts could be significantly reduced, compared to previous versions of the ontology (cf. Sect. 12.4). For the ontology as a whole, the reduction cannot be quantified since there are other, counteracting effects that influence the total number of terms when moving from version 1.0 to 2.0 (e.g., the extension of the ontology to new application areas). Yet we may give some examples, which demonstrate the progress made in selected areas:
The first example is about the conceptualization of topological connectivity between *connections* and *devices* (cf. Fig. 10.1).

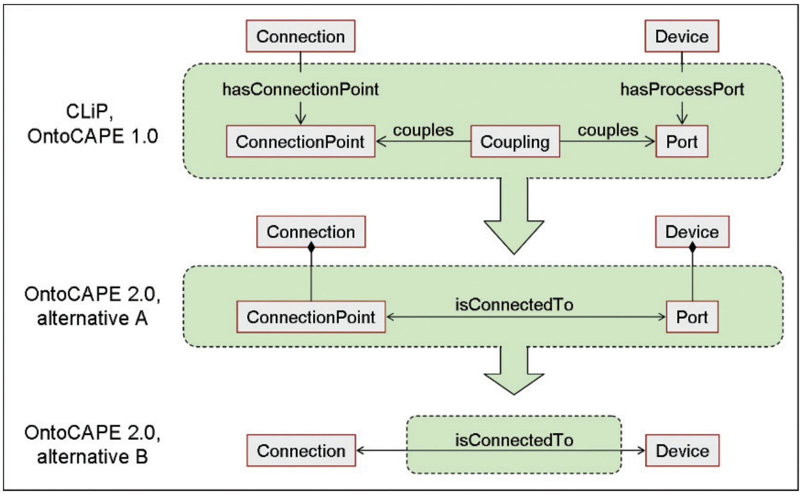

Fig. 10.1: Conceptualization of connectivity in CLiP, OntoCAPE 1.0, and OntoCAPE 2.0

In CLiP and OntoCAPE 1.0, such connectivity is modeled through three classes (*port*, *connection point*, *coupling*) plus three relations[103] (hasPort, hasConnectionPoint, couples), as shown in the upper part of Fig. 10.1. An evaluation of this conceptualization yielded that the *coupling* class and the couples relation could be replaced by a single relation (isConnectedTo) without any loss of expressivity, thus saving one vocabulary term. Another two terms could be saved by realizing that the relations hasConnectionPoint and hasPort, respectively, are redundant with the previously defined hasDirectPart relation. Finally, it was recognized that the usage of *ports* and

[103] For sake of simplicity, the inverses of the respective relations are not included in the count.

connection points is optional, not mandatory; consequently, in some cases, connectivity may simply be represented by the relation isConnectedTo.
Fig. 10.2 shows a second example, regarding the representation of the backdrop concept (cf. Sect. 5.1.10): When this concept was first introduced in CLiP, it was represented through a total of four vocabulary terms (two classes plus two relations) as shown in the upper part of Fig. 10.2. In OntoCAPE 1.0, the conceptualization of the backdrop was already reduced by one class without loosing expressiveness (i.e., the concept of backdrop was only respresented by the relation hasBackdrop). Finally, the assignment of values to certain properties was further condensed in OntoCAPE: Now the respresentation requires only a single relation called isObservedAgainstBackdrop, by which the backdrop concept is modeled very efficiently.

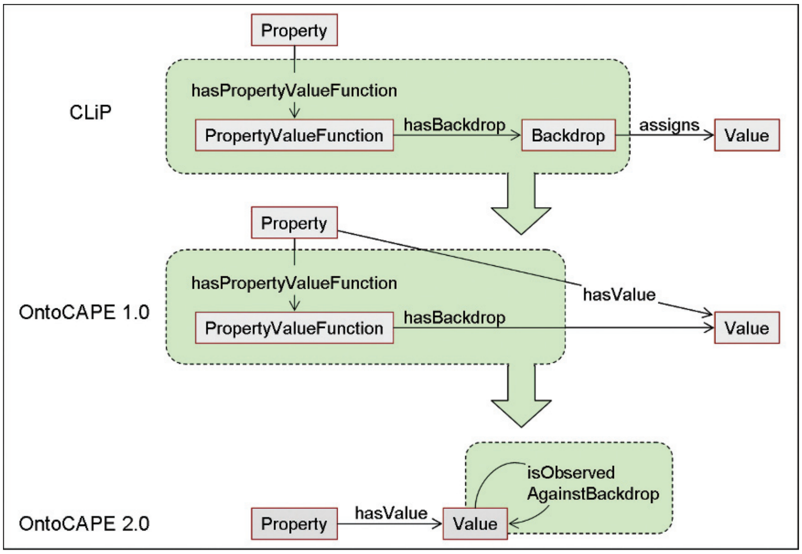

Fig. 10.2: Conceptualization of backdrop in CLiP, OntoCAPE 1.0 and 2.0

10.3 Intelligibility

According to the principle of *intelligibility* (a.k.a. *understandability*), an ontology should be easily understandable to users “who are willing to invest a reasonable amount of effort in mastering its documentation” (Smith 2006). This principle applies to the informal as well as to the formal specification; its significance should be obvious, since an incomprehensible ontology is unlikely to find any users.

The intelligibility of an ontology is improved by abiding the principle of conciseness, as explained in Sect. 10.2. Also, modularizing the ontology, as described in Sect. 3.1.2, facilitates the understanding of the ontology, since the user does not need to grasp the entire ontology at once, but gradually by accessing one module after another.
In addition to these measures, the principle of intelligibility of an ontology can be further enhanced by following three major (sub-)principles: *clarity*, *homogeneity*, and *thorough documentation*. Subsequently, issues pertaining to these sub-principles will be discussed.

10.3.1 Clarity

The first sub-principle to mention is *clarity*, a.k.a. *perspicuity*. Clarity means to state exact and unambiguous definitions for all ontological terms in order to effectively communicate the intended semantics. The principle of clarity applies to both the formal and the informal definition of terms:

– For the formal definitions, the same recommendations should be followed as for achieving coherence (cf. Sect. 10.1): That is, the term definitions should be axiomatized to the degree possible, and the use of defined classes and tight constraints are particularly advised. As mentioned before, OntoCAPE has put these recommendations into practice by means of more than a thousand axioms (cf. Sect. 12.4).
– As for the informal definitions, Smith (2006) formulated a number of rules for how to give precise and intelligible term definitions: It is, for example, recommended to clearly distinguish between defined and primitive classes, to reuse term definitions from recognized sources, or to avoid circular definitions. Even though the rules were not yet published during the development of OntoCAPE, the term definitions later proved to largely comply with these rules[104].

[104] The only exception is rule no. 13, which demands to avoid words that invite subjective interpretation, such as ’which may’, ‘indicates’, ‘characterizes’, etc. However, this demand is debatable: While we certainly tried to define terms as precisely as possible, words like ‘characterize’ and ‘indicates’ have been often used to define relations; this is because it is often the function of the relation to indicate or characterize a particular class. Also, the phrase ‘which may’ is often used to describe a possible relation between two classes (i.e., a relation with a cardinality of 0..n). If utilized this way, we do not consider these words to be ambiguous, and thus to be in agreement with the above rule.

10.3.2 Homogeneity

Homogeneity is the second sub-principle to be discussed. Essentially, it means to follow a consistent, uniform modeling style across the ontology, or, in other words, to conceptualize similar things in a similar way. A homogenous style facilitates the understanding of new concepts: Users that have already mastered one part of the ontology will comprehend the other parts more quickly if they recognize familiar structures and can thus draw on their existing knowledge. As a positive side effect, the compatibility between the ontology parts is increased, and new definitions can be added more easily; consequently, the monotonic extensibility (cf. Sect. 10.4) of the ontology is improved (Gómez-Pérez et al. 2004).

A minimum condition for homogeneity is to apply the same modeling patterns (and use the same primitives) for defining sibling terms (cf. Arpírez et al. 1998; Gómez-Pérez et al. 2004). The alikeness of siblings can be enforced by defining constraints on the parent class or relation, thus restraining the possible definitions of sibling terms. This practice is systematically applied throughout OntoCAPE. What's more, the Upper Layer establishes sibling relations and thus enforces alikeness even between semantically distant concepts and across different modules and partial models: For instance, the classes *material*, *model*, and *chemical process system*, which are located in different partial models, are declared to be subclasses of the *system* class. Consequently, they are conceptualized alike, according to the constraints defined on the *system* class – for instance, their respective characteristics are represented via *properties* and *values*.

Over and above, OntoCAPE goes one step further: The Meta Model (cf. Chap. 4) even encourages the application of the same design pattern for modeling semantically dissimilar concepts, if appropriate. For instance, the *multiset* design pattern is used to model such dissimilar concepts as the stoichiometry of a chemical reaction or the tray stack of a distillation column.

10.3.3 Thorough Documentation

Just like any piece of software, OntoCAPE needs to be thoroughly documented to be usable. Typically, software documentation includes the following issues:

(1) *Comments* within the source code (i.e., within the formal specification).

(2) A *reference guide* (intended for developers who want to realize a software application based on OntoCAPE).

(3) A *user manual* (intended for practitioners who want to work with a software tool based on OntoCAPE).

As for (1), the textual term definitions given in the informal specification have been copied in the formal specification. Thus, each term in the formal specification is supplemented by a short description of its intended meaning. Moreover, each OWL file includes a header comment, which (i) shortly summarizes the contents of the respective module, and (ii) lists the classes, relations, and individuals from other modules that are referenced or used by the current module. The latter allows a fast evaluation of the module's interdependencies with other modules. Also, in case the ontology is modified, it can be easily detected if the module is affected by that change.

Issues (2) and (3) are addressed by the informal specification, which presents the ontology in human-readable form. Generally, its function is to help new users familiarize themselves with the ontology; in particular, it is intended to explain the correct usage of the ontology terms to practitioners and to support application developers in refining, extending, or changing the ontology to their particular needs.

The informal specification holds a separate chapter for each partial model. The chapters are further structured into sections describing the individual modules. Every section provides a comprehensive overview on the respective module, followed by a listing of term definitions. An alphabetic term index at the end of each chapter allows quickly locating the definition of a particular term.

A term definition comprises two major parts. It starts off with a lexical description of the term. Subsequently, the formal definition of the term (i.e., the set of axioms that are stated in the formal specification) is paraphrased in natural language. The latter is intended as a sort of "neutral" formal specification[105] (i.e., independent of a particular modeling language). To give an example, the term '*model*' is first established by the following lexical description: A *model* is a system that is used to enable the understanding of or the command over the original system, or to replace the original system. Model system and original system share certain characteristics that are of relevance to the task at hand (Wüsteneck 1963).

Subsequently, its formal definition is paraphrased as follows: Necessary and sufficient condition: A *model* is a *system* that models some other *system*. Further necessary condition: A *model* models only *systems*.

Note that, in the above example, the formal definition captures only a part of the lexical description. This is frequently the case, since, as explained in Sects. 10.1 and 10.5, not all aspects of a term definition can be formally represented.

The term definitions are not sufficient to communicate a comprehensive understanding of a module. Therefore, the informal specification provides further documentation of the following kinds:

- UML-like diagrams provide graphical views on each module. Such diagrams depict the interdependencies between the major ontology terms as well as their hierarchical ordering.
- The diagrams are supplemented by explanatory texts, which specify the meaning of the entire conceptualization and explicate the underlying mod-

[105] The availability of a neutral formal specification helps translating the ontology into another modeling language. Experience shows that developers often choose to represent the ontology in a different formalism, either to benefit from special language features, or because the application requires a particular standard (cf. Chap. 4.1).

eling rationale. The latter is particularly relevant for developers who consider changing some part of the ontology due to application requirements: Understanding the modeling rationale enables them to better assess the consequences of such a change.

– The intended usage of essential concepts has been explicitly stated as an additional help for both practitioners and developers: The former are advised how to utilize the ontology, whereas the latter may learn if the ontology is suitable for a particular application. Usage is typically described by (i) defining one or two illustrative use cases, and (ii) demonstrating how these use cases are realized on the instance level.
– Last but not least, guidelines have been stated for certain modules, advising developers how to further extend and refine that part of the ontology.

10.4 Adaptability

A reusable ontology is not a static model, but evolves over time according to prevailing conditions and requirements; new tasks and application areas may necessitate extending and customizing the ontology with respect to scope, level of detail or granularity, and/or conceptualization (cf. discussion of the interaction problem in Chap. 2.6). Thus, an ontology must anticipate the possibility of later changes and support their realization. This demand is hereafter referred to as the principle of *adaptability*. It can be broken down into two sub-principles – herein referred to as *extensibility* and *customizability* – which will be discussed in the following.

10.4.1 Extensibility

Given the complexity of a domain like chemical engineering as well as the wide range of possible tasks and applications, a domain ontology cannot be expected to be complete. Instead, it must be *extensible*, with respect to scope as well as with respect to the level of detail or granularity; the latter is also referred to as monotonic extensibility (a.k.a. extendibility, cf. Gruber and Olsen 1994).

– *Extensibility in scope* implies that the ontology allows for the addition of entirely new subject areas, which have not been conceptualized so far. The open architecture of OntoCAPE invites this type of extensions: New modules and partial models covering further subject areas can be smoothly added on the Conceptual Layer and below, the only condition being that their conceptualizations comply with the modeling guidelines established by the Upper Layer and the Meta Layer. These guidelines, however, are of generic nature and thus do not rule out any topic areas. The practical feasibility of this approach has been demonstrated, for instance, by the later addition of the modules *process_control* and *process_control_system*, which extend the scope of OntoCAPE to the area of process control.

– *Monotonic extensibility* means that new ontological terms for special uses can be defined based on the existing vocabulary, in a way that does not require a revision of the existing definitions (Gruber and Olsen 1994). The modules located on the Conceptual Layer are explicitly designed for monotonic extension: They provide the "representational machinery" (Gruber 1995) – i.e., the basic concepts of a topic area – required to construct more specialized terms; the specializations are realized within separate modules on the subjacent application-near layers. Various modules on the Application-Oriented Layer have been built by this procedure; their existence demonstrates the practicability of this approach.

According to Aitken (1998), an indication of the principles and guidelines of ontology design facilitates the extension of an ontology tremendously. OntoCAPE explicitly states such principles and guidelines – on the one hand in form of the Meta Model, on the other hand in form of explanations within the informal specification (cf. Sect. 10.3.3). The declaration of these design principles and guidelines ensures consistency and reduces the likelihood of making ad-hoc extensions. This is of particular importance when extensions are carried out by diverse developers and/or at different times.

Extensibility is furthermore enhanced through the modular structure of OntoCAPE: Typically, an extension is realized by creating a new module, which is then embedded in the inclusion hierarchy of OntoCAPE. Within the hierarchy, the new module does usually not include all the other modules of the ontology, but only a selected few. Consequently, the newly defined terms do not have to be compatible with the entire ontology, but only with a subset, thus reducing the likelihood of inconsistencies between new and existing ontological definitions.

10.4.2 Customizability

Over the lifecycle of OntoCAPE, new application contexts are likely to arise, which were not anticipated during ontology development. Since different applications usually imply different views on the world (Noy and Klein 2004; cf. Sect. 3.1.2.3), the applications will have individual demands on the ontology and thus require different conceptualizations. Therefore, the ontology must be *customizable* – that is, it must be able to flexibly adapt to dissimilar, and possibly contradicting, application requirements.

If some new application requires an individual conceptualization, which is different from the one previously specified in the ontology, it will not be necessary to produce an entirely new version of OntoCAPE. Instead, new *variants* of the affected ontology modules will be developed and integrated into the present ontology, as explained in Sect. 3.1.2.3. The introduction of variants is an option whenever different applications need the same scope of knowledge yet represented in a different way – that is, at a different level of detail or granularity, or reflecting a different usage perspective.

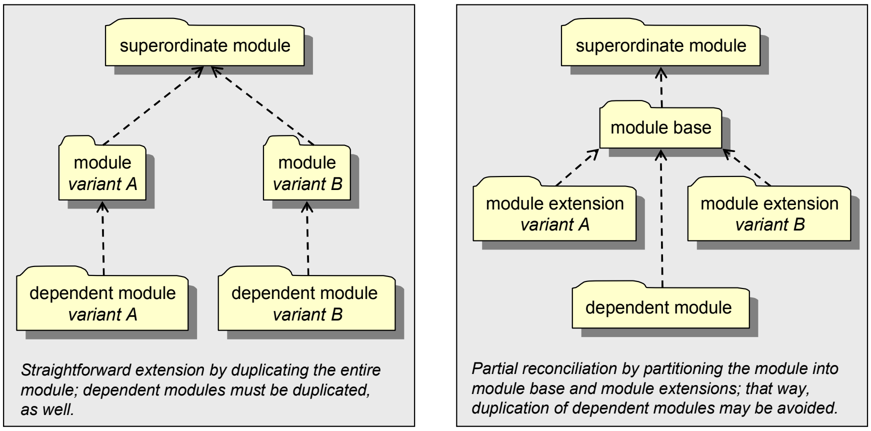
superordinate module
module *variant A*
module *variant B*
dependent module *variant A*
dependent module *variant B*
Straightforward extension by duplicating the entire module; dependent modules must be duplicated, as well.
superordinate module
module base
module extension *variant A*
module extension *variant B*
dependent module
Partial reconciliation by partitioning the module into module base and module extensions; that way, duplication of dependent modules may be avoided.

Fig. 10.3: Introduction and subsequent reconciliation of variants

The use of variants has the advantage that the ontology can quickly be adapted to new tasks, regardless of the impact of changes on existing applications. The obvious drawback of this approach is that, with an increasing number of supported applications, the number of variants gets unmanageably large, in particular because the dependent modules (those that are positioned below the variants in the inclusion lattice, cf. left side of Fig. 10.3) must be split into variants, as well, since they are derived from different conceptualizations. That way, different *configurations* of the ontology evolve. Configuration management systems such as CVS (GNU 2006) can support the handling of the growing number of variants and their mutual dependencies.

At the same time, a reconciliation of the variants should be considered. A simple merging of the variants would contradict the idea of customized conceptualizations for particular applications. As a compromise, a new *module base* can be introduced, which contains the ontological definitions shared by all variants (cf. right side of Fig. 10.3); simultaneously, the variants are reduced to mere *extensions* of the base module. Module extensions hold only those ontological definitions that distinguish the variants from one another. Subordinate modules – if not explicitly dependent on concepts defined in the extensions – can refer to the base module only, thus avoiding unnecessary duplication. Reasoners that offer non-standard inferences – for example determination of the *least common subsumer* of related concepts (Cohen et al. 1992) – can support the reconciliation process (Molitor 2000).

10.5 Minimal Ontological Commitment

As explained in Chap. 2.2, committing to an ontology means accepting the definitions in the ontology as an appropriate conceptualization of the application domain. The principle of minimal ontological commitment states that "an ontology should make as few claims as possible about the world being modeled, allowing the parties committed to the ontology freedom to specialize and instantiate the ontology as needed" (Gruber 1995). In other words: For sake of reusability, the axiomatization of ontological terms should be kept to a minimum in order to allow for different extensions and thus fit a large number of application contexts. However, this conflicts with the previously discussed principles of coherence and intelligibility, which postulate tight axiomatic definitions to constrain the possible interpretation of terms. Borst (1997) summarizes this dilemma as follows: "Overcommitment reduces the reusability, but undercommitment reduces the usability of an ontology".

The problem is partially solved by the modular, layered structure of OntoCAPE: It allows for a "piecemeal ontological commitment" Borst (1997), meaning that a human or software agent may selectively commit to a coherent subset of the ontology that fits its respective application context: For example, an agent could commit to scalar quantities (which are defined in the module *system*), but not to tensor quantities (which are introduced in the module *tensor_quantity*); or, it could commit to a conceptualization of unit operations (given in the module *process*), but not to a conceptualization of plant equipment (given in the module *plant*); or, it could commit to the principles of general systems theory (which are established on the Upper Layer), but not to those of process engineering (established on the Conceptual Layer).

10.6 Efficiency

An ontology (or rather its formal specification) is said to be *efficient* if it allows for efficient reasoning (with respect to computational time and memory requirements), and if it scales adequately for large amounts of instance data.

As a rule, the efficiency depends on the number of formal axioms included in the ontology – the more exist, the longer it will take a reasoner to process the ontology. Thus, efficiency is enhanced by following the principle of conciseness, since a concise ontology usually involves fewer axioms and is thus easier to process than a complex one. On the other hand, efficiency conflicts with the principles of coherence and intelligibility, which call for a high degree of axiomatization.

Yet the number of axioms is only one of several influencing factors. Other factors are the types of axioms used – some axioms are more difficult to process than others – as well as their respective combinations. Particularly the latter can prove to be problematic for reasoner performance: If, for instance, two relations are declared to be both mutually inverse and transitive, their instances will scale badly with most current reasoners (Rector and Welty 2005).

Unfortunately, only very few guidelines are available on how to avoid such unfortunate combinations of axioms. Therefore, efficiency improvement is for the most part a question of trial and error: As explained by Fox and Grüninger (1998), the efficiency of a particular conceptualization can be improved as follows: Firstly, a set of appropriate competency questions is defined and formalized (cf. Sect. 4.1). Next, the reasoner performance is evaluated against a set of test data by measuring the computational time and memory required to answer those queries. If the reasoner performance is found to be unsatisfactory, an alternative conceptualization must be created, and put to the test. Hence, sound procedures and appropriate tool support for testing and systematically improving efficiency would be a highly desirable objective for future research in ontology engineering.

The suggested procedure is very time-consuming, since each concept must be tested individually. And even if this was accomplished for each concept, the resulting ontology could not be claimed to be efficient, in general; instead, it would merely have been optimized towards one particular reasoner. Considering the rapid progress currently being made on the field of reasoner algorithms, it is questionable whether one should spend too much effort on optimizing the efficiency of an ontology – after all, what seems inefficient today may prove to be satisfactory tomorrow if processed by a different reasoner. Still, for the ontology to be practically usable, efficiency must be achieved to some degree at least.

Due to the above considerations, OntoCAPE has not been entirely optimized for efficiency; solely the Meta Model has been systematically improved by the above described trial-and-error procedure (cf. Chap. 4). The Meta Model was deliberately chosen because its design patterns are repeatedly applied throughout the ontology; consequently, the Meta Model is the part of the ontology where time for efficiency improvement is spent most effectively.

In the course of improving the efficiency of the Meta Model, we frequently encountered a conflict between efficiency and knowledge-encoding requirements. A typical example is the conceptualization of the part-whole relationship through the relations hasPart and isPartOf: To model the properties of these relations correctly, hasPart and isPartOf should be declared to be both transitive and mutually inverse. Yet, conversely, the principle of efficiency dictates to give up one of the two relation properties since, as explained above, reasoners scale badly for this particular combination of axioms. As it cannot be known in advance which one of the two relation properties is more important for an application, it has been decided to represent both of them in the formal specification, thus preferring completeness to efficiency. In return, the informal specification points out the conflict and advises the user how the conceptualization can be changed towards efficiency, thus allowing the user to adapt the ontology to the respective application requirements.

10.7 Conclusions

Numerous recommendations for ontology design are given in the literature. This plethora has been condensed to the six major principles, which have guided the design of OntoCAPE: coherence, conciseness, intelligibility, adaptability, minimal ontological commitment, and efficiency. It has been described how OntoCAPE has put these principles into practice and, in the course of this process, significantly gained in quality. Since some of the principles are incompatible, a suitable balance between the conflicting principles had to be found. The finally realized design is a reasonable compromise, which supports the two primary objectives of usability and reusability. However, it is difficult to quantify the degree of quality due to the absence of generally accepted key figures. Thus, we decided to compensate the lack of formal key figures by prototypical software applications. In the end, the degree of (re)usability can only be proven by field-testing OntoCAPE in a (preferably large) number of different software applications.
Correspondingly, OntoCAPE has been field-tested in software projects covering the application areas of modeling and simulation as well as process design and engineering. Through these tests and the iterated application within these projects, the ontology was systematically improved. However, research activities investigating sound procedures for checking the quality of design during development and the availability of universal key figures are highly desirable.

10.8 References

Aitken S (1998) Extending the HPKB-upper-level ontology: experiences and observations. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:11–15.

Arpírez JC, Gómez-Pérez A, Lozano A, Pinto HS (1998) (ONTO)^{2}Agent: an ontology-based WWW broker to select ontologies. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:16–24.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Chandrasekaran B, Josephson JR, Benjamins VR (1999) What are ontologies, and why do we need them? *IEEE Intell. Syst.* **14** (1):20–26.

Cohen WW, Borgida A, Hirsh H (1992) Computing least common subsumers in description logics. In: Swartout W (ed.): *Proceedings of the 10th National Conference on Artificial Intelligence*. MIT Press:754–760.

Fox MS, Grüninger M (1998) Enterprise modeling. *AI Mag.* **19** (3):109–121.

GNU (2006) *CVS – Open Source Version Control*. Online available at http://www.nongnu.org/cvs/. Accessed October 2007.

Gómez-Pérez A, Fernández-López M, Corcho O (2004) *Ontological Engineering*. Springer, Berlin.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/pa-pers/engmath.html. Accessed September 2007.

Molitor R (2000) *Unterstützung der Modellierung verfahrenstechnischer Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken*. PhD thesis, Department of Computer Science, RWTH Aachen University.

Noy NF, Klein M (2004) Ontology evolution: not the same as schema evolution. *Knowl. Inform. Syst.* **6**:428–440.

Racer Systems (2007) *RacerPro Reference Manual*. Online available at http://www.racer-systems.com/products/racerpro/reference-manual-1-9-2-beta.pdf. Accessed June 2008.

Rector A, ed. (2005) *Representing Specified Values in OWL: "value partitions" and "value sets"*. W3C Working Group Note, 17 May 2005. Online available at http://www.w3.org/TR/swbp-specified-values. Accessed November 2007.

Rector A, Welty C, eds. (2005) *Simple part-whole relations in OWL Ontologies*. W3C Editor's Draft, 11 August 2005. Online available at http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. Accessed November 2007.

Rector A, Drummond N, Horridge M, Rogers J, Knublauch H, Stevens R, Wang H, Wroe C (2004) OWL pizzas: practical experience of teaching OWL-DL: common errors and common patterns. In: Motta E, Shadbolt N, Stutt A, Gibbins N (eds.): *Proceedings of the European Conference on Knowledge Acquisition (EKAW)*. Springer:63–81.

Seipel D, Baumeister J (2004) Declarative methods for the evaluation of ontologies. *Künstliche Intelligenz* **18** (4):51–57.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Stanford Center for Biomedical Informatics Research (2008) *The Protégé Ontology Editor and Knowledge Acquisition System*. Online available at http://protege.stanford.edu/. Accessed January 2008.

Tsarkov D, Horrocks I (2007) *FaCT++*. Online available at http://owl.man.ac.uk/factplusplus/. Accessed January 2008.

Wüsteneck KD (1963) Zur philosophischen Verallgemeinerung und Bestimmung des Modellbegriffs. *Deutsche Zeitschrift für Philosophie* **1963**:1504 et sqq.

11 Related Work on Ontologies for Engineering Applications

This chapter gives an overview on the information models and ontologies that are thematically related to OntoCAPE. It is structured into two major parts: Section 11.1 reviews the previous work in the research group of the authors; that is, the information models preceding OntoCAPE are described, and the progress made over time is discussed. In Sect. 11.2, the work of other research groups is analyzed and compared against OntoCAPE.

11.1 History of OntoCAPE

Since the early 1990's, a consecutive series of information models has been developed in the authors' research group. The series includes the information models **VeDa** (Baumeister and Marquardt 1998; Souza and Marquardt 1998a; 1998b; Bogusch and Marquardt 1998; Krobb et al 1998; von Wedel and Marquardt 1999) and **CLiP** (Bayer 2003; Yang et al. 2003; Bayer and Marquardt 2004), and the ontology **OntoCAPE 1.0** (Yang and Marquardt 2004; Yang et al. 2008). These models formalize the domain knowledge and represent the work processes of chemical engineering. As for the modeling of domain knowledge, OntoCAPE 2.0 constitutes - at least for the time being - the final result of this series; the modeling of work processes is now addressed by the WPML (Work Process Modeling Language) an ontology for work processes developed by Theißen et al. (2008a; 2009), which can be combined with OntoCAPE. Fig. 11.1 visualizes the history of model development.

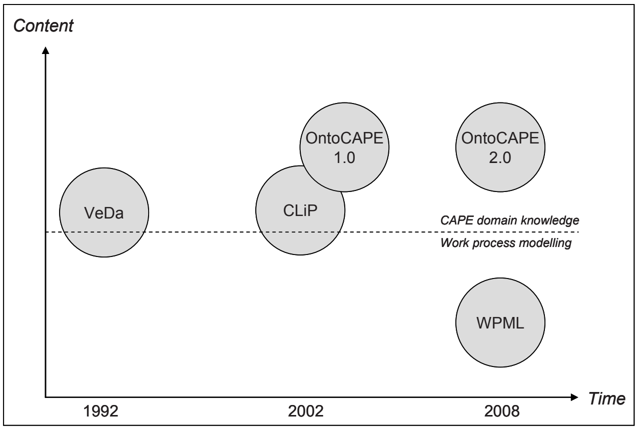

Fig. 11.1: History of model development

In the following, the predecessor models of OntoCAPE 2.0 will be shortly analyzed with regard to scope, structure, representation language, (re)usability, and general quality. A more extensive review can be found elsewhere (Morbach et al. 2008b).

11.1.1 VeDa

The development of VeDa was initiated by Marquardt (1992a; 1992b; 1994a; 1994b) and Marquardt et al. (1993) to support the mathematical modeling of chemical process system. For that reason, VeDa primarily provides concepts for the description of mathematical models, of the geometrical and structural properties of the modeled objects, and – for documentation purposes – of the activities and the decisions taken during model building. Selected elements of these conceptualizations were later absorbed into OntoCAPE for modeling, e.g., materials, physicochemical phenomena, coordinate systems, or mathematical models. General systems theory and network theory were chosen as the basic organizing principles for VeDa; later, these principles were adopted by VeDa's successors CLiP and OntoCAPE (cf. Sects. 11.1.2 and 11.1.3). VeDa also initiated the description of a system from different viewpoints: A structural and a behavioral view on systems have been conceptualized. Whereas the structural view focuses on the description of elements of a (chemical process) system and their couplings, the behavioral view considers the process quantities and model equations relating these quantities. As for the overall structure, VeDa is horizontally subdivided by three abstraction layers; vertically, it is partitioned into five partial models.
Since no suitable modeling language was available in the early nineties, a frame-based language named VDDL was specifically developed for VeDa (cf. Baumeister and Marquardt 1998). Combining features from object-oriented modeling and description logics, VDDL is a highly expressive modeling language, which supports the definition of classes, metaclasses, and instances, as well as different types of attributes and relations. Class definitions can include methods and laws: In this context, methods represent numerical or symbolical functions that act upon the objects; laws restrict the possible instances of classes and attributes by logic expressions. A serious drawback of the language is that it lacks an efficient implementation. In fact, the genuine VDDL-version of VeDa only exists on paper, and its usability is consequently rather limited. Although some executable subsets of the language have been implemented – e.g., in the modeling environment G2 (Gensym 2008) – these implementations are merely pared-down versions, which lack important language features.
Based on VeDa, several software applications have been developed, namely the model repository ROME (von Wedel and Marquardt 2000) and the modeling tools ModKit (Bogusch et al. 2001; Bogusch 2001) and ProMoT (Tränkle et al. 1997;

Tränkle 2000). Yet due to its narrow scope, VeDa's reusability is confined to applications in the area of mathematical modeling.

11.1.2 CLiP

In succession to VeDa, the conceptual information model CLiP has been developed as part of the IMPROVE project (Marquardt and Nagl 2004; Nagl and Marquardt 2008). While VeDa largely focused on mathematical modeling, CLiP is primarily concerned with chemical process design. Consequently, its scope covers materials and their properties, plant equipment and machinery, the major unit operations, the mathematical models used in the various model-based design activities, the documents for archiving and exchanging data between designers and software tools, and the activities performed during process design. Except for the subject of geometric concepts, CLiP comprises all areas covered by VeDa and extends beyond its scope. Regarding the representation of chemical processes, CLiP models not only the structural and behavioral views previously covered by VeDa, but also considers the aspects of function and performance. As for work process modeling, the area of interest has been shifted from work processes targeting the development of mathematical models to work processes targeting chemical process design (Jarke and Marquardt 1995; Lohmann and Marquardt 1998).
Again, systems theory has been chosen as the fundamental modeling principle. In comparison to VeDa, however, CLiP takes a more systematic approach by representing the underlying systems approach explicitly in a meta model. To this end, the meta model defines key concepts like *system*, *property*, *value*, *backdrop*, etc. As before, a system can be described from different viewpoints; unlike before, such viewpoints are now explicitly modeled as instances of the *aspect* class (this concept later evolved to the *aspect system* used in OntoCAPE 2.0, cf. Sect. 5.1.7).
CLiP is structured by means of layers and partial models. In comparison to VeDa, the number of partial models has been significantly increased, resulting in a more fine-grained partition of the domain. As for the layers, a distinction is made between the Meta Meta Class Layer, which holds the meta model defining the *system* concept and its *aspects*; the Meta Class Layer, which introduces different kinds of systems and their specific properties; and the Simple Class Layer, which defines concepts related to different tasks in the design process and therefore corresponds roughly to VeDa's middle layer. Unlike VeDa, CLiP has no layer for application-specific classes.
Two different modeling languages are utilized for the formal specification of CLiP:

– Both Meta Layers and some parts of the Simple Class Layer are implemented in the logic-based modeling language O-Telos, which forms part of the ConceptBase system (Jarke et al. 1995; Bayer and Marquardt 2009).

ConceptBase supports meta modeling and provides basic deductive reasoning services to assist schema development and maintenance.
- The concepts on the Simple Class Layer are represented by means of UML class diagrams. The principal motivation for the use of the UML was that a graphical notation like UML is better suited for the representation and management of large and complex product data models than the frame-based O-Telos language of ConceptBase. On the other hand, the UML is less expressive than O-Telos, and it does not support reasoning.

The two model implementations are not formally integrated; instead, consistency is ensured by an overlap of major concepts on the Simple Class Layer.
CLiP is reusable in different application contexts, such as information management (Eggersmann et al. 2003a), information integration (Bayer and Marquardt 2004), CAD/CAE (Bayer 2003), or mathematical modeling (Hackenberg 2006). Yet since CLiP constitutes a conceptual model, it is not directly usable: For each application, a specific implementation model must be developed on the basis of CLiP. Moreover, CLiP's usability is further limited on account of the following defects:

- Sometimes, the conceptualizations of topic areas are unnecessarily complex, as exemplified by the samples of CLiP presented in Sect. 10.2.
- Classes and relations are not given individual definitions; instead, they are merely specified by means of UML class diagrams accompanied by some explanatory text. Relations are often not even named, such that their functions and semantics remain unclear. As a result, the intended semantics of concepts are only vaguely communicated. Also, in some cases, logical inconsistencies are "hidden" by fuzzy wording.
- There are a number of systematic errors, such as the repeated mix-up of aggregation and composition relations, or the unsystematic use of classes and meta-classes.

11.1.3 OntoCAPE 1.0

OntoCAPE continues the work of CLiP and VeDa, combining results of both models: From CLiP, it largely adopts the terminology and the partial model structure of the Simple Class Layer. Additionally, it includes various concepts from VeDa – particularly for modeling the geometry and the physicochemical behavior of chemical plants, which are not sufficiently covered by CLiP. Some additional subject areas not considered by the previous models have been newly conceptualized in OntoCAPE – for instance, the issue of numerical solution strategies. Unlike CLiP and VeDa, OntoCAPE does not cover documents and work processes; however, OntoCAPE can be combined with other, specialized ontologies representing documents, work processes, and decision-making procedures, as described by Brandt et al. (2008b) and Morbach et al. (2008d).

The first version of OntoCAPE was developed in the COGents project (Braunschweig et al. 2002, 2004; Yang et al. 2008). Within COGents, OntoCAPE 1.0 served as a shared language for communication between cooperating computer agents, which were concerned with the task of retrieving suitable mathematical models for describing chemical process systems from distributed libraries (cf. Sect. 12.1). Accordingly, the scope of OntoCAPE 1.0 had its main focus on mathematical modeling and numerical simulation. Yet right from the beginning, OntoCAPE was designed for later extension to other areas and applications. To this end, the ontology was given an open and extensible structure. Moreover, further essential topic areas of CAPE, such as process design and engineering, were at least basically conceptualized in order to be refined later on. Thus, OntoCAPE 1.0 established the basic framework for a comprehensive domain ontology.

The structure of OntoCAPE 1.0 differs notably from that of OntoCAPE 2.0 described in Chap. 3: Horizontally, the concepts of OntoCAPE 1.0 are split between two layers. The first layer holds the so-called Common Concepts, which are shared between different applications. The second layer contains the Application-Specific Concepts, which are obtained by extending and/or refining the Common Concepts towards a particular application. Each layer is subdivided into a number of modules, which are grouped into partial models. In total there are 23 modules on the Common Concepts layer, plus another six on the Application-Specific Concepts layer.

OntoCAPE 1.0 has been formally specified in DAML+OIL (Connolly et al. 2001), which was the state-of-the-art ontology modeling language at that time. The formal specification has been created by means of the ontology editor OilEd (Bechhofer et al. 2001) and verified by the reasoner FaCT (Horrocks 1998). A supplementary informal specification has been created (Yang et al. 2004b), providing textual term definitions as well as general information about each module in form of UML-like class diagrams and accompanying texts.

After the completion of the COGents project, the further development of OntoCAPE was taken over by the IMPROVE project (Marquardt and Nagl 2004; Nagl and Marquardt 2008). As a first step, the formal specification was translated[106] into OWL, which had by then replaced DAML+OIL as the standard ontology modeling language. Subsequent to the translation, OntoCAPE was fundamentally revised:

[106] This laborious task – as of version 1.0, the ontology contained about 600 classes and 400 relations represented by 20,000 lines of code – was supported by a specially built converter (Amin and Morbach 2008), which handled most of the translation. However, as OWL does not offer equivalents for all language elements of DAML+OIL (particularly not for qualified cardinality constraints), some parts of the ontology had to be remodeled manually.

- The ontology was given a refined structure: The Meta Layer, the Upper Layer, and the Application-Oriented Layer have been added, and the boundaries of the modules and partial models have been redrawn.
- The software applications created within the IMPROVE project required an extension of the ontology. Consequently, 37 new modules have been added to the ontology in order to cover new subject areas and to refine the coverage of existing areas in detail. Concurrently, it was decided to remove five modules from the ontology since their contents were not of general interest but merely relevant for the COGents project (cf. Sect. 12.1 and Sect. 12.4.2).
- The contents of the 24 remaining original modules have been conceptualized anew. The remodeling became necessary since, due to the tight schedule of the COGents project, large parts of OntoCAPE 1.0 had to be formulated in an ad-hoc manner. Within the IMPROVE project, these conceptualizations have been revisited and improved with respect to the design criteria of intelligibility, conciseness, and coherence: A number of redundant concepts were deleted, and inconsistencies have been resolved. In many cases, long-winded conceptualizations could be replaced by more succinct ones, thus improving the intelligibility and conciseness of the ontology. By the addition of numerous axioms, the formal term definitions have been tightened in order to enhance their intelligibility and validate their coherence. For details on these issues, we refer to Sects. 12.4.2 and 12.4.3.
- The informal specification has been rewritten and extended to almost three times its original length. It now includes much more explanatory text and, for the first time, describes the intended usage of concepts as well as the underlying design rationales.

11.2 Work by Other Research Groups

Since the release of the popular OWL modeling language in 2004, the number of publicly available ontologies has grown exponentially: According to Hendler (2007), tens of thousands of ontologies have been in use in 2007. While most of these ontologies are negligible efforts – test cases, academic exercises, lightweight ontologies, or pseudo ontologies – there still remains a considerable number of valuable contributions that are, or have the potential to emerge as, long-lasting shared ontologies. These ontologies cover all sorts of domains, such as mathematics (e.g., the EngMath ontology; Gruber and Olsen 1994), engineering (e.g., the PhysSys ontology; Borst 1997), chemistry (e.g., the ChEBI ontology; EBI 2008), biology (e.g., the Gene Ontology; GO 2007), medicine (e.g., the GALEN ontology; Rector et al. 1995;), geoscience (e.g., the GeoNames ontology: GeoNames

2007), legislation (e.g., the LKIF-Core ontology; Hoekstra et al. 2007), e-commerce (e.g., the SNAP ontology; Morgenstern and Riecken 2005), business enterprises (e.g., the Enterprise Ontology; Uschold et al. 1998), military (e.g., the JFACC ontology; Valente et al. 1999), or surveillance and security (e.g., the BioSTORM ontologies; Crubézy et al. 2005).
A few review articles have been published, which give an overview on the major ontologies in selected fields like medicine (Bodenreider 2001) or biology (Schulze-Kremer 1998; Bard and Rhee 2004), or even across fields (Fridman-Noy and Hafner 1997; Chap. 1.1 of Gómez-Pérez et al. 2004). Moreover, the following web resources may serve as a starting point for searching a particular ontology:

- The Protégé Ontology Library (http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library) is a web portal linking to ontology projects that use the Protégé ontology editor:
- Swoogle (http://swoogle.umbc.edu/) is a specialized web search engine for discovering semantic web documents and ontologies.
- SchemaWeb (http://www.schemaweb.info/default.aspx) is a directory of schemas and ontologies expressed in the RDFS, OWL, and DAML+OIL modeling languages.
- The DAML Ontology Library (http://www.daml.org/ontologies/) contains several hundreds of ontologies represented in the DAML+OIL modeling language.
- The Open Biomedical Ontologies (OBO) repository is a library of publicly accessible biomedical ontologies, which are obtainable in different formats. The OBO repository can be accessed via the website of the OBO Foundry (http://obofoundry.org/), via the Ontology Lookup Service (http://www.ebi.ac.uk/ontology-lookup/), or via a web application called BioPortal (http://www.bioontology.org/ncbo/faces/index.xhtml).

Of the plethora of existing ontologies, we will review only a small portion, namely those ontologies that are of particular relevance in the context of this work. An ontology is considered to be relevant (a) if it bears close resemblance to OntoCAPE with respect to both scope and level of complexity, or (b) if it had a significant influence on the development of OntoCAPE. As for (a), only the more recent efforts are discussed here; a review of the earlier contributions in the area of process engineering can be found elsewhere (Bayer and Marquardt 2003). Moreover, this chapter considers only "true" ontologies (i.e., ontologies that have explicitly been built for sharing and reuse, cf. Sect. 2.5); pseudo ontologies are not included in this review.

11.2.1 EngMath

EngMath (Gruber and Olsen 1994) is an ontology for engineering mathematics. It formally defines fundamental mathematical concepts required for the modeling of engineering systems, such as scalar and tensor quantities, mathematical functions, physical dimensions, and units. The ontology is intended (i) as an unambiguous communication language that is usable by both humans and software agents, and (ii) as a conceptual foundation for other, more comprehensive engineering ontologies. As for (ii), EngMath forms an integral part of the PhysSys ontology (Borst 1997; cf. Sect. 11.2.3).

EngMath had an important influence on the modeling of certain upper-level concepts of OntoCAPE: It inspired the conceptualization of *physical quantities*, *physical dimensions*, *units*, and (to a lesser degree) *tensor quantities*. An important difference between EngMath and OntoCAPE is that the latter distinguishes between a *physical quantity* and its *quantitative value(s)* (cf. Sect. 5.1.9), whereas the former does not. In fact, EngMath does not have an equivalent for the OntoCAPE concept of a *physical quantity*; the homonymous concept in EngMath rather corresponds to the OntoCAPE concept of a *quantitative value*.

11.2.2 YMIR

YMIR (Alberts 1994) is a formal ontology that is concerned with the modeling of technical systems. Technical systems are described through a systems approach based on bond-graph theory (e.g., Karnopp et al. 1990), which models a system as a network of interconnected subsystems that interact via flows of energy. According to this approach, complex system models can be synthesized by combining and refining generic subsystems, the so-called Generic System Models (GSM). The specification of a particular GSM comprises three different aspects, which reflect its geometric form, its structure, and its behavior. Moreover, a GSM can be associated to certain function(s) and the corresponding “context” in which the system is to perform the function, such that the “purpose” of the system can be defined.

A weakness of YMIR is its bias towards mathematics: The ontological description of a technical system is firmly associated with one particular mathematical model; there is no possibility to characterize the technical system qualitatively without mathematics. This emphasis of mathematics limits the applicability of the ontology, since many applications only require a qualitative description of technical systems. Furthermore, as argued by Borst (1997), the relationship between a technical system and its possible mathematical descriptions is n-to-n, in general; YMIR, however, presumes a one-to-one relation and thus looses the flexibility of introducing alternative models for a technical system.

Another critical point about YMIR is its lack of extensibility: As pointed out by Gruber and Olsen (1994), the ontology lacks a mechanism for introducing new physical quantities and dimensions in addition to those that are predefined; also, the modeling approach is not extensible towards vectors and tensors.

11.2.3 PhysSys

PhysSys (Borst 1997; Borst et al. 1995; Borst et al. 1997) is an ontology for the modeling of technical systems. The ontology, which is formally specified in the Knowledge Interchange Format, KIF (Genesereth and Fikes 1992), is divided into subontologies organized in an inclusion hierarchy. The topmost and most generic subontology is the Mereology Ontology, which establishes a mereological theory. The Topology Ontology imports this theory and extends it towards a theory of mereotopology. Based on these, the Systems Theory Ontology defines notions of general systems theory. The subjacent subontologies specify technical systems from complementary viewpoints: The Component Ontology describes the structural aspect of a technical system, whereas the Physical Process Ontology characterizes its physicochemical behavior qualitatively. Additionally, the behavior can be quantitatively modeled by using concepts from the EngMath ontology (Gruber and Olsen 1994; cf. Sect. 11.2.1), which has been included as an integral part of PhysSys.
The modular architecture of PhysSys has inspired the structural design of OntoCAPE: Particularly the consecutive definition of mereology, topology, and systems theory in separate subontologies has been adopted. Also, selected axiomatic definitions from these theories have been transferred to OntoCAPE. On the whole, however, the conceptualization of mereology, topology, and systems theory is handled quite differently in OntoCAPE:

– PhysSys is aimed at a complete representation of mereotopology, whereas OntoCAPE implements only a reduced theory: Certain mereotopological properties, such as the asymmetry of the part-whole relation, have not been modeled in OntoCAPE due to the limited expressive power of OWL (cf. Sect. 2.3). On the other hand, the reduced theory has the advantage of enabling more efficient reasoning than a complex theory.
– While both ontologies cover the areas of mereotopology and systems theory, they place emphasis on different topics: For instance, mereotopology, as conceptualized in PhysSys, is particularly suited for modeling the overlapping parts of two objects – a topic that is not covered by OntoCAPE. Conversely, the mereotopological theory implemented in OntoCAPE sets priorities on the representation of graph structures and their hierarchical refinement. Systems Theory in PhysSys particularly emphasizes the modeling of system boundaries, which enables the definition of open

and closed systems. OntoCAPE rather focuses on the representation of system properties and their values, as well as on modeling a system from different viewpoints.
- Finally, PhysSys is particularly suited for modeling electronic and mechanical systems, which have been the target of by the bond graph approach originally (e.g., Karnopp et al. 1990; see Cellier and Kofman 2006 for an application of bond graphs to thermodynamical systems). OntoCAPE, on the other hand, rather specializes in the representation of chemical, thermodynamical, and fluid mechanical systems.

11.2.4 MDF

Batres, Naka, and co-workers developed an ensemble of interrelated ontologies intended for applications in the process engineering domain, which is referred to as *multi-dimensional formalism*, or MDF in short (Batres and Naka 2000; Batres et al. 2002). The MDF ontologies are formally specified in the Knowledge Interchange Format, KIF (Genesereth and Fikes 1992). The multidimensional information model MDOOM (Lu et al. 1997; Batres et al. 1999) served as the conceptual framework for the design of the ontologies.
MDF consists of the following interconnected ontologies:

- The Plant Structure Ontology, which is comparable to the partial model **CPS_realization** in OntoCAPE, describes the physical structure and the spatial layout of chemical plants. The structural description is based on a mereotopological theory, which is specified by a number of axiomatic definitions.
- The Material Ontology describes the thermodynamic behavior of materials. Like the partial model **material** in OntoCAPE, the Material Ontology covers only those aspects of material behavior that are independent of a material's concrete occurrence in time and space (cf. Sect. 7.1.1). The context-dependent properties of material are handled by the Behavior Ontology described next.
- The Behavior Ontology characterizes the behavior of a particular amount of material in a given spatiotemporal setting. It thus has the same function as the OntoCAPE partial model **CPS_behavior** (cf. Sect. 8.6). Just like in OntoCAPE, the material behavior may be qualitatively described by the indication of the relevant physicochemical phenomena. A further similarity is the so-called 'metamodel' concept, which is the equivalent to OntoCAPE's *process unit* (cf. Sect. 8.8).
- Finally, the Management and Operation Ontology provides the vocabulary to specify typical tasks in plant operation and control. These topics are not covered in an equally detailed fashion in OntoCAPE, but a comparable

functionality can be achieved by combining OntoCAPE with the process ontology (Eggersmann et al. 2008) as described by Morbach et al. (2008d). Moreover, OntoCAPE is currently extended to cover operational processes (cf. Theißen et al. 2008a, Hai et al. 2009)

MDF was developed contemporaneously with CLiP, the predecessor model of OntoCAPE. Due to the exchange of experience between the two research groups during the development process, both models share a number of common features, such as the definition of the structural and material-related views on the plant, which were later adopted by OntoCAPE[107]. Furthermore, OntoCAPE adopted some basic concepts from the MDF Behavior Ontology for the realization of the partial model **CPS_behavior**. However, when it comes to the details of the implementations, MDF and OntoCAPE have been conceptualized quite differently. In addition, there are some noteworthy structural differences, the most notable of which is the absence of abstraction layers: MDF does not attempt to separate the generic knowledge from the domain knowledge and/or application-specific knowledge – mereotopological axioms, for instance, are intertwined with a conceptualization of plant items. Moreover, there is no upper ontology that would serve as a common basis for the four MDF ontologies; consequently, the consistency between the different ontologies is not formally enforced. Finally, OntoCAPE has a higher degree of modularization: For example, the **material** partial model – unlike its equivalent in MDF (the Material Ontology) – is further subdivided into six ontology modules on the Conceptual Layer plus five additional ones on the Application-Oriented Layer. Judging from these differences, it can be concluded that MDF is less reusable than OntoCAPE.

11.2.5 Plant Ontology and Functional Ontology

Mizoguchi et al. (2000) and Mizoguchi (2001) have developed a Plant Ontology, which describes equipment, materials, and operating activities in chemical plants. Combining this ontology with a generic Functional Ontology (Kitamura and Mizoguchi 1999; Mizoguchi 2001; Kitamura and Mizoguchi 2003) enables the functional description of chemical plants. Both ontologies were built in a prototypical ontology editor named Hozo (Kozaki et al. 2000) and can be exported to XML, Lisp, or text formats.

– The Plant Ontology is subdivided into a Domain Ontology and a Task Ontology. The Domain Ontology describes the application domain, in particular plant equipment, materials, and material properties. The Task Ontology (i) represents the plant operations that are performed by the operating per-

[107] A detailed comparison between CLiP and MDF can be found elsewhere (Bayer and Marquardt 2003).

sonnel (monitor, diagnose, operate, etc.), and (ii) defines how the concepts of the domain ontology are to be used in order to perform a desired task[108]. The key idea for the organization of both subontologies is the distinction of "essential categories" and "view-dependent concepts", according to which all ontological concepts are classified. Roughly speaking, the former category denotes entities with unchanging characteristics and/or appearances (e.g., chemical substances), while the latter comprises concepts the appearances of which depend on the context. An example of the latter would be the role of a chemical substance in a plant – depending on context, a substance could be fuel, distillate, or end-product.

– The Functional Ontology enables a comprehensive representation of conceptual design knowledge. Engineering artifacts are described from a structural, functional, and behavioral perspective. The ontology allows both functional and behavioral decomposition and defines relations between the decomposed parts. Additionally, it is possible to indicate (i) the physicochemical phenomena based on which a certain function can be achieved as well as (ii) the manufacturing steps that are required to realize the desired function. The Functional Ontology is organized as a framework of several interconnected ontologies, which are distributed across several layers of abstraction; each layer contains a number of subontologies.

In principle, any technical artifact can be described through the Functional Ontology; however, it is especially appropriate for modeling fluid-processing plants in the process and energy engineering domain. According to the authors, the usability of the Functional Ontology has been demonstrated by building, amongst others, functional models of an oil refinery, a chemical plant, and a power plant. To this end, the Functional Ontology has presumably been combined with the Plant Ontology (however, this issue is not clearly stated in the available publications).

A point of criticism is that – to our knowledge – the ontologies have not been made publicly available. Yet as far as it can be judged from the available sources, the respective conceptualizations are rather complicated, particularly those in the Functional Ontology. Another problematic issue is that both ontologies have been designed for particular applications, possibly without any intention of reusing them in different application contexts: The Plant Ontology serves as the knowledge base for an intelligent plant operating system, whereas the Functional Ontology is used within a novel type of design support system. Since other applications have not been reported, it remains unclear whether the ontologies are reusable or must be classified as pseudo ontologies.

[108] According to the classification framework established in Chap. 2.6, the Task Ontology would rather be categorized as an application (task) ontology.

11.2.6 ISO 15926

ISO 15926 is an evolving international standard that defines information models for the integration and the exchange of lifecycle data about chemical plants. The standard comprises seven distinct parts; three parts have already been released (ISO 2003; 2004; 2007), while the others are still under development (ISO 2005; 2006). Eventually, the standard will provide information models for the representation of product data, documents, and activities. These models describe the physical objects that exist in a process plant (materials, equipment and machinery, control systems, etc.) as well as the design requirements for and the functional specification of these objects. They cover the lifecycle stages of development, construction, operation, and maintenance.

The ISO 15926 is organized in a layered architecture, starting with a generic Data Model, which is iteratively extended and refined to a Reference Data Library (second layer), Templates (third layer), and Object Information Models (fourth layer). Further lower layers allow for user-specific extensions.

Whether ISO 15926 truly constitutes an ontology is subject to debate. While ISO 15926 was originally intended as an information model for a shared database or data warehouse, its upper layers are now additionally promoted as an ontology. Indeed, the semantics of the data model have been formally specified in the EXPRESS modeling language (ISO 1994) and are currently re-formulated in OWL (cf. Batres et al. 2007; Teijgeler 2007). Yet Smith (2006) lists a number of systematic defects of ISO 15926, which cause it to be both unintelligible and incoherent and thus make Smith conclude that "we do not have here anything which could properly be described as an ontology".

A detailed comparison of ISO 15926 and OntoCAPE has been published elsewhere (Morbach et al. 2008d). Below the major findings of this evaluation are summarized:

– Due to its intended usage as a model for lifecycle data, ISO 15926 gives a fine-grained and highly detailed description of the domain, resulting in a very complex model that incorporates a large number of specialized classes. A drawback of this approach is that the ISO 15926 is only accessible to experts who are willing to spend a considerable amount of time to get acquainted with it. OntoCAPE, by contrast, has been designed to be as simple and intuitive as possible, such that it can be easily understood and applied by less experienced practitioners. In consequence of this principle, it is sometimes necessary to trade accurateness and precision against usability, which leads to a coarse- to mid-grained conceptualization of the application domain. However, this level of detail has proven to be sufficient for many applications, as will be demonstrated in Chap. 12.
– A further notable difference is the conceptualization of temporal persistence: ISO 15926 advocates the perdurantistic (or 4D) worldview, while

OntoCAPE takes an endurantistic (or 3D) perspective[109]. There are arguments for both sides – a summary of the current philosophical debate on this issue is, for example, given by Hawley (2004) and Noonan (2006). In the end, the decisive factor must be the applicability of the chosen approach in the target applications: ISO 15926 is aimed at storage and exchange of lifecycle data and thus places a strong emphasis on the representation of temporal changes, for which the 4D perspective is advantageous. By contrast, temporal changes are less relevant for the applications targeted by OntoCAPE, such that we consider the 3D paradigm more practicable for our purposes: It has the advantage of being more intuitive, which supports our goal of an easily usable ontology. If required, temporal changes can be modeled by means of the backdrop concept (cf. Sect. 5.1.10).

– Judging from the available sources published so far (Batres et al. 2007; Teijgeler 2007), the ISO models will not be realized in OWL DL, but in the more expressive sublanguage OWL Full (cf. Smith et al. 2004). OWL Full supports such advanced language features as metamodeling (i.e., instantiation across multiple levels) or augmenting the meaning of the predefined language primitives, yet at the cost of loosing scalability and compatibility with DL reasoners. This, in combination with the above discussed general complexity of the ISO, reduces the efficiency (and thus the usability) of the ontology.

11.2.7 Conclusions

Summarizing, it can be stated that there are currently two major ontologies for the domain of CAPE, OntoCAPE and ISO 15926. All other ontologies known to the authors are either intended for related but different domains (EngMath, YMIR, PhysSys), or they are not truly reusable (MDF, Plant Ontology).
OntoCAPE and ISO 15926 pursue different design objectives: OntoCAPE is intended as a general-purpose ontology; in order to be reusable, it strives to be both concise and generic. ISO 15926, by contrast, primarily aims at applications in the area of data management; therefore, it puts emphasis on being detailed and complete, even at the cost of being less reusable. In view of that, the two ontologies should not be considered as competing, but as complementary options for a conceptualization of the CAPE domain.

[109] Endurantism is a philosophic theory, which assumes that an object exists as a whole at each moment of its history. Perdurantism by contrast, perceives an object as a four-dimensional entity consisting of a series of temporal parts.

At present, ISO 15926 still suffers from a number of defects and has not yet been published completely. Thus, it may be claimed OntoCAPE is currently the only (re)usable ontology that is available for the CAPE domain.

11.3 References

Alberts LK (1994) YMIR: a sharable ontology for the formal representation of engineering design knowledge. In: Gero JS, Tyugu E (eds.): *Formal Design Methods for CAD*. Elsevier, New York:3–32.

Amin MA, Morbach J (2008) *DAML+OIL to OWL converter*. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=523. Accessed January 2008.

Bard JBL, Rhee SY (2004) Ontologies in biology: design, applications and future challenges. *Nat. Rev. Genet.* **5**:213–222.

Batres R, Naka Y (2000) Process plant ontologies based on a multi-dimensional framework. In: Malone MF, Trainham JA, Carnahan B (eds.): *Fifth International Conference on Foundations of Computer-Aided Process Design*. AIChE:433–437.

Batres R, Naka Y, Lu ML (1999) A multidimensional design framework and its implementation in an engineering design environment. *Concur. Eng.* **7** (1):43–54.

Batres R, Aoyama A, Naka Y (2002) A life-cycle approach for model reuse and exchange. *Comput. Chem. Eng.* **26** (4/5):487–498.

Batres R, West M, Leal D, Price D, Masaki K, Shimada Y, Fuchino T, Naka Y (2007) An upper ontology based on ISO 15926. *Comput. Chem. Eng.* **31** (5/6):519–534.

Baumeister M, Marquardt W (1998) *The Chemical Engineering Data Model VeDa, part 1: VDDL – The Language Definition*. Technical Report (LPT-1998-01), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bayer B (2003) *Conceptual Information Modeling for Computer Aided Support of Chemical Process Design*. Fortschritt-Berichte VDI: Reihe 3, Nr. 787. VDI-Verlag, Düsseldorf.

Bayer B, Marquardt W (2003) A Comparison of Data Models in Chemical Engineering. *Concur. Eng.* **11** (2):129-138.

Bayer B, Marquardt W (2004) Towards integrated information models for data and documents. *Comput. Chem. Eng.* **28** (8):1249–1266.

Bayer B, Marquardt W (2009) A Conceptual Information Model for the Chemical Process Design Lifecycle. In : Jarke M, Jeusfeld M, Mylopoulos J (eds.): *Meta Modeling and Method Engineering*, MIT:357-381.

Bechhofer S, Horrocks I, Goble C, Stevens R (2001) OilEd: a reasonable ontology editor for the semantic web. In: Baader F, Brewka G, Eiter T (eds.): *KI 2001: Advances in Artificial Intelligence*. Springer, Berlin:396–408.

Bodenreider O (2001) *Medical Ontology Research*. Technical Report, Lister Hill National Center for Biomedical Communications, U.S. National Library of Medicine. Online available at http://mor. nlm.nih.gov:8000/pubs/pdf/ 2001-MOR-BoSC.pdf. Accessed February 2008.

Bogusch R (2001) *A Software Environment for Computer-Aided Modeling of Chemical Processes*. Fortschritt-Berichte VDI: Reihe 3, Nr. 705, VDI-Verlag, Düsseldorf.

Bogusch R, Marquardt W (1998) *The Chemical Engineering Data Model VeDa. Part 4: Behavioral Modeling Objects*. Technical Report (LPT-1998-04), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bogusch R, Lohmann B, Marquardt W (2001) Computer-aided process modeling with ModKit. *Comput. Chem. Eng.* **25** (7/8):963–995.

Borst P, Akkermans JM, Top JL (1997) Engineering ontologies. *Int. J. Hum Comput Stud.* **46**:365–406.

Borst P, Akkermans JM, Pos A, Top JL (1995) The PhySys ontology for physical systems. In: Bredeweg (ed.): *Proceedings of the 9th International Workshop on Qualitative Reasoning*. University of Amsterdam:11–21.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Brandt SC, Morbach J, Miatidis M, Theißen M, Jarke M, Marquardt W (2008b) An ontology-based approach to knowledge management in design processes. *Comput. Chem. Eng.* **32**:320–342.

Braunschweig B, Fraga ES, Guessoum Z, Paen D, Piñol D, Yang A (2002) CO-Gents: cognitive middleware agents to support e-CAPE. In: Stanford-Smith B, Chiozza E, Edin M (eds.): *Challenges and Achievements in E–business and E–work*. IOS Press:1182–1189.

Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I, Yang A (2004) CAPE web services: the COGents way. In: Barbarosa-Póvoa A, Matos H (eds.): *European Symposium on Computer Aided Process Engineering -14*. Elsevier, Amsterdam:1021–1026.

Cellier FE, Kofman E (2006) *Continuous System Simulation*. Springer, Berlin.

Connolly D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2001) *DAML+OIL reference description*. W3C Note, 18 December 2001. Online available at http://www.w3.org/TR/daml+oilreference. Accessed January 2008.

Crubézy M, O'Connor M, Buckeridge DL, Pincus Z, Musen MA (2005) Ontology-centered syndromic surveillance for bioterrorism. *IEEE Intell. Syst.* **20** (5):26–35.

EBI – European Bioinformatics Institute (2008) *Chemical Entities of Biological Interest (ChEBI)*. Online available at http://www.ebi.ac.uk/chebi/. Accessed June 2008.

Eggersmann M, Bayer B, Jarke M, Marquardt W, Schneider R (2003a) Prozess- und Produktmodelle für die Verfahrenstechnik. In: Westfechtel B, Nagl M (eds.): *Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*. Wiley-VCH, Weinheim:75–90.

Eggersmann M, Hai R, Kausch B, Luczak H, Marquardt W, Schlick C, Schneider N, Schneider R, Theißen M (2008) Work process models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:126–152.

Fridman-Noy N, Hafner CD (1997) The state of the art in ontology design – a survey and comparative review. *AI Mag.* **18** (3):53–74.

Genesereth MR, Fikes RE et al. (1992) *Knowledge Interchange Format, Version 3.0 Reference Manual*. Technical Report (Logic-92-1), Stanford University Logic Group. Online available at http://citeseer.ist.psu.edu/genesereth92knowledge.html. Accessed October 2007.

Gensym (2008) *Business Rule Management, Business Rules, BPM Software*. Online available at http://www.gensym.com/. Accessed March 2008.

GeoNames (2007) *GeoNames Ontology*. Online available at http://www.geonames.org/ontology/. Accessed October 2007.

GO Consortium (2007) *An Introduction to the Gene Ontology*. Online available at http://www.geneontology.org/GO.doc.shtml. Accessed October 2007.

Gómez-Pérez A, Fernández-López M, Corcho O (2004) *Ontological Engineering*. Springer, Berlin.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

Hackenberg J (2006) *Computer Support for Theory-Based Modeling of Process Systems*. Fortschritt-Berichte VDI, Reihe 3, Nr. 860, VDI-Verlag, Düsseldorf.

Hai R, Theißen M, Marquardt W (2009) An integrated ontology for operational processes. In: Jezowski J, Thullie J (eds.): *Proceedings of the 19th European Symposium on Computer-Aided Process Engineering*. Elsevier:1087–1091.

Hawley K (2004) Temporal Parts. In: Zalta, E.N. (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2004 Edition)*. Online available at http://plato.stanford.edu/archives/win2004/entries/temporal-parts/. Accessed October 2007.

Hendler J (2007) Where are all the intelligent agents? *IEEE Intell. Syst.* **22** (3):2-3.

Hoekstra R, Breuker J, Di Bello M, Boer A (2007) The LKIF core ontology of basic legal concepts. In: Casanovas P, Biasiotti MA, Francesconi E, Sagri MT (eds.): *Proceedings of the 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques*. CEUR Workshop Proceedings:43-63.

Horrocks I (1998) Using an expressive description logic: FaCT or fiction? In: Cohn AG, Schubert L, Shapiro SC (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*. Morgan Kaufmann, San Francisco:636–647.

ISO (1994) Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. International Standard ISO 10303-11:1994, International Organization for Standardization, Geneva, Switzerland.

ISO (2003) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 2: Data model. International Standard ISO 15926-2:2003, International Organization for Standardization, Geneva, Switzerland.

ISO (2004) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 1: Overview and fundamental principles. International Standard ISO 15926-1:2004, International Organization for Standardization, Geneva, Switzerland.

ISO (2005) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 7: Implementation methods for data exchange and integration. International Standard under development ISO/CD TS 15926-7, International Organization for Standardization, Geneva, Switzerland.

ISO (2006) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 3: Ontology for geometry and topology. International Standard under development ISO/NP TS 15926-3, International Organization for Standardization, Geneva, Switzerland.

ISO (2007) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 4: Initial reference data. International Standard ISO/TS 15926-4:2007, International Organization for Standardization, Geneva, Switzerland.

Jarke M, Marquardt W (1995) Design and Evaluation of Computer-Aided Process Modeling Tools. In: Davis J, Stephanopoulos G, Venkatasubramanian V (eds): *Intelligent Systems in Process Engineering,* AICHE Symp. Ser., **312** (92):97-109.

Jarke M, Gallersdörfer R, Jeusfeld MA, Staudt M, Eherer S (1995) ConceptBase – a deductive object base for meta data management. *J. Intell. Inf. Syst.* **4** (2):167–192.

Karnopp DC, Margolis DL, Rosenberg RC (1990) *System Dynamics: A Unified Approach*. John Wiley, New York.

Kitamura Y, Mizoguchi R (1999) Meta-functions of artifacts. In: Price C (ed.): *Proceedings of the 13th International Workshop on Qualitative Reasoning*. University of Aberystwyth:136–145.

Kitamura Y, Mizoguchi R (2003) Ontology-based description of functional design knowledge and its use in a functional way server. *Expert Syst. Appl.* **24** (2):153–166.

Kozaki K, Kitamura Y, Ikeda M, Mizoguchi R (2000) Development of an environment for building ontologies which is based on a fundamental consideration of "relationship" and "role". In: Compton P, Hoffmann A (eds.): *Proceedings of the 6th Pacific Knowledge Acquisition Workshop*. University of New South Wales:205–221.

Krobb C, Lohmann B, Marquardt W (1998) *The Chemical Engineering Data Model VeDa. Part 6: The Process of Model Development.* Technical Report (LPT-1998-06), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Lohmann B, Marquardt W (1998) Entwicklungsprozesse für den konzeptionellen Entwurf. In: Nagl M, Westfechtel B (eds.): *Integration von Entwicklungssystemen in Ingenieuranwendungen*, Springer, Berlin:253-264.

Lu ML, Batres R, Li HS, Naka Y (1997) A G2 based MDOOM testbed for concurrent process engineering. *Comput. Chem. Eng.* **21**:11–16.

Marquardt W (1992a) An object-oriented representation of structured process models. *Comput. Chem. Eng.* **16**:329–336.

Marquardt W (1992b) Rechnergestützte Erstellung verfahrenstechnischer Prozessmodelle. *Chem. Ing. Tech.* **64**:25-40.

Marquardt W (1994a) Trends in Computer-Aided Process Modeling. *5th International Symposium on Process Systems Engineering, PSE'94,* Proceedings PSE'94:1-24.

Marquardt W (1994b) Computer-aided generation of chemical engineering process models. *Int. Chem. Eng.* **34**:28–46.

Marquardt W, Nagl M (2004) Workflow and information centered support of design processes – the IMPROVE perspective. *Comput. Chem. Eng.* **29** (1):65–82.

Marquardt W, Gerstlauer A, Gilles ED (1993) Modeling and Representation of Complex Objects: A Chemical Engineering Perspective, *6th Int. Conf. on Industrial and Engineering Applications to Artificial Intelligence and Expert Systems*, Proceedings:219-228.

Mizoguchi R (2001) Ontological engineering: foundation of the next generation knowledge processing. In: Zhong N, Yao Y, Liu J, Ohsuga S (eds.): *Web Intelligence: Research and Development*. Springer, Berlin:44–57.

Mizoguchi R, Kozaki K, Sano T, Kitamura Y (2000) Construction and deployment of a plant ontology. In: Dieng R, Corby O (eds.): *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. Springer, Berlin:113–128.

Morbach J, Bayer B, Yang A, Marquardt W (2008b) Product data models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:93–110.

Morbach J, Theißen M, Marquardt W (2008d) Integrated application domain models for chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:169–182.

Morgenstern L, Riecken D (2005) SNAP: An action-based ontology for e-commerce reasoning. In: *Proceedings of the 1st Workshop "FOMI 2005" – Formal Ontologies Meet Industry.*

Nagl M, Marquardt W, eds. (2008) *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support*. Springer, Berlin.

Noonan H (2006) Identity. In: Zalta EN (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2006 Edition)*. Online available at http://plato.stanford.edu/archives/win2006/ entries/identity/. Accessed January 2008.

Rector A, Solomon WD, Nowlan WA, Rush TW (1995) A terminology server for medical language and medical information systems. *Methods Inf. Med.* **34** (1/2):147–157.

Schulze-Kremer S (1998) Ontologies for molecular biology. In: *Proceedings of the Third Pacific Symposium on Biocomputing*. AAAI Press:693–704.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Smith MK, Welty C, McGuinness DL eds. (2004) *OWL Web Ontology Languages Guide*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-guide/. Accessed October 2007.

Souza D, Marquardt W (1998a) *The Chemical Engineering Data Model VeDa. Part 2: Structural Modeling Objects*. Technical Report (LPT-1998-02), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Souza D, Marquardt W (1998b) *The Chemical Engineering Data Model VeDa. Part 3: Geometrical Modeling Objects*. Technical Report (LPT-1998-03), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Teijgeler H (2007) *InfowebML, OWL-based Information Exchange and Integration based on ISO 15926*. Online available at http://www.infowebml.ws/. Accessed November 2007.

Theißen M, Hai R, Marquardt W (2008a) Computer-assisted work process modeling in chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:656–666.

Theißen M, Hai R, Marquardt W (2009) A framework for work process modeling in the chemical industries. *8th World Congress of Chemical Engineering (WCCE8)*. Montréal, August 2009.

Tränkle F (2000) *Rechnerunterstützte Modellierung verfahrenstechnischer Prozesse für die Simulationsumgebung DIVA*. Fortschritt-Berichte VDI 309, Reihe 20, Nr. 309. VDI-Verlag, Düsseldorf.

Tränkle F, Gerstlauer A, Zeitz M, Gilles ED (1997) Application of the modeling and simulation environment PROMOT/DIVA to the modeling of distillation processes. *Comput. Chem. Eng.* **21**:841–846.

Uschold M, King M, Moralee S, Zorgios Y (1998) The enterprise ontology. *Knowl. Eng. Rev.* **13**:31–89.

Valente A, Russ T, MacGregor R, Swartout W (1999) Building and (re)using an ontology of air campaign planning. *IEEE Intell. Syst.* **14** (1):27–36.

von Wedel L, Marquardt W (1999) *The Chemical Engineering Data Model VeDa. Part 5: Material Modeling Objects*. Technical Report (LPT-1998-05), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

von Wedel L, Marquardt W (2000) ROME: a repository to support the integration of models over the lifecycle of model-based engineering. In: Pierucci S. (ed.): *Proceedings of the European Symposium on Computer Aided Process Engineering – ESCAPE 10*. Elsevier:535–540.

Yang A, Schlüter M, Bayer B, Krüger J, Haberstroh E, Marquardt W (2003) A concise conceptual model for material data and its applications in process engineering. *Comput. Chem. Eng.* **27** (4):595–609.

Yang A, Marquardt W (2004) An ontology-based approach to conceptual process modeling. In: Barbarosa-Póvoa A, Matos H (eds.): *Proceedings of the European Symposium on Computer Aided Process Engineering – ESCAPE 14*. Elsevier:1159–1164.

Yang A, Marquardt W, Stalker I, Fraga I, Serra M, Pinol D, Paen D, Roux P, Braunschweig B (2004b) Principles and Informal Specification of OntoCAPE. COGents. Technical Report, COGents Project, Information Society Technologies Program IST-2001-34431.

Yang A, Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I (2008) A multiagent system to facilitate component-based process modeling and design. *Comput. Chem. Eng.* **32** (10):2290–2305.

12 Evolutionary Improvement and Validation through Applications

In this chapter, some prototypical software applications are presented, which have been created by the authors' colleagues within the research projects COGents and IMPROVE. What all prototypes have in common is that their respective implementations are based on OntoCAPE. While these prototypes are of interest in themselves, they are presented here primarily for other reasons: that is, (a) to demonstrate the practical usability and reusability of OntoCAPE, and (b) to highlight the gradual improvement of the different versions of OntoCAPE, which have been triggered by implementations in different application contexts. Therefore, the descriptions of the different prototypes are intentionally kept short; detailed accounts of these have been published elsewhere, as will be indicated by appropriate references. Rather, the focus will be on the validation of the ontology through practical applications as well as on the feedback for ontology engineering gained from these applications.

Sect. 12.1 describes two software prototypes from the field of process modeling. Both prototypes have been developed in the context of the COGents project and are based on OntoCAPE 1.0. Sect. 12.2 discusses the use of OntoCAPE 2.0 by a tool for knowledge management developed in the IMPROVE project. Finally, Sect. 12.3 presents an ongoing research project, in which parts of OntoCAPE 2.0 are tested in a large-scale industrial application.

For the sake of completeness, it should be mentioned that OntoCAPE is, or was, furthermore involved in some other software research projects:

- Within the IMPROVE project, rule-driven integrator tools have been developed (Becker et al. 2008a). One particular application of such tools is the integration of the process simulator Aspen Plus (AspenTech 2008) and the CAE system Comos PT (Innotec 2008). The integration rules for this particular case have been derived from the domain knowledge represented in OntoCAPE[110] (Becker et al. 2008b).
- Sheremetov et al. (2007) have developed a knowledge-based framework that allows interchanging data and integrating different engineering applications, simulators and tools, thus facilitating the collaboration of piping, stress and civil engineers in pipe networks design. The framework includes a domain model for piping, which is based on OntoCAPE.

[110] In this particular case, OntoCAPE is not used directly, but indirectly as a conceptual domain model (cf. Morbach et al 2008a). In the areas of software engineering and database design, a conceptual domain model describes the major entities of the domain of interest on a conceptual level, independently of some particular application or implementation. Its function is to familiarize with the vocabulary of the domain and to establish a common understanding of its key concepts.

- De Giacomo et al. (2007) employed OntoCAPE as a benchmark ontology for the testing and evaluation of reasoning services and tools.
- A presentation by Morbach and Marquardt (2006) explored the usage of OntoCAPE in the area of e-procurement.
- Theißen et al. (2008a; 2008b) are developing tools for the modeling and analysis of design and operational work processes in chemical engineering (cf. Sect. 13.4). Generic models for the representation of work processes are combined with OntoCAPE in order to represent domain-specific work processes.

While the above listed projects also demonstrate the reusability of OntoCAPE, they have influenced the development of OntoCAPE far less than the projects presented in the following.

12.1 Process Modeling

In this section, two software prototypes are presented, which both support the composition of process models. The prototypes are based on OntoCAPE 1.0 and have been developed in the context of the COGents project. Sect. 12.1.1 describes a software that enables the composition of process models from reusable components, whereas Sect. 12.1.2 presents an environment for conceptual process modeling. Sect. 12.1.3 summarizes the lessons learned from using OntoCAPE in these two modeling applications.

12.1.1 Component-Based Process Modeling

The first software prototype to be discussed is a multi-agent framework, which supports the retrieval of suitable *Process Modeling Components* (PMCs) from model libraries distributed on the interest. A PMC is a model constituent which is (re)usable for a particular process simulation; it may be one of the following: an entire process model, a part thereof (such as a unit model or a property data file), or a numerical solver. The COGents system supports the user in retrieving appropriate PMCs for a given task and in embedding those into the user's simulation environment.

Within the multi-agent framework, OntoCAPE serves as a communication language between the interacting software agents, and between the software agents and the human users: Concepts from OntoCAPE are used to formulate a *Modeling Task Specification* (MTS), which is then matched against available PMCs also described through OntoCAPE. A typical usage scenario is illustrated in Fig. 12.1.

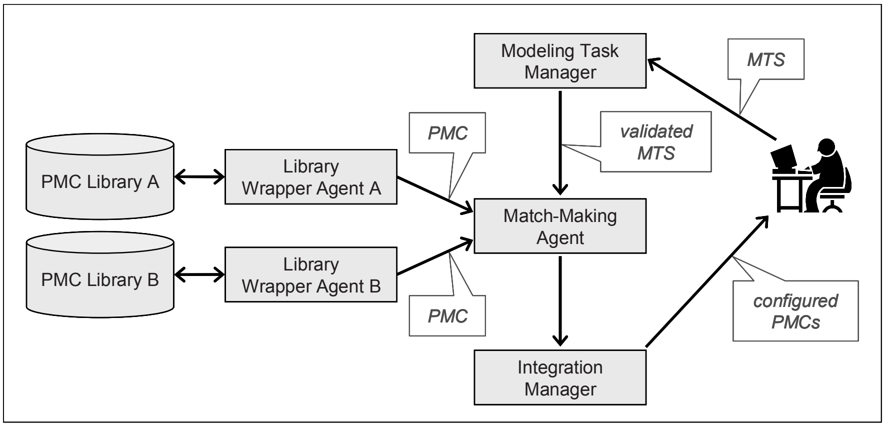

Fig. 12.1: Simplified COGents architecture

At the outset of the scenario, a user, supported by the *Modeling Task Manager*, composes a MTS by selecting and instantiating appropriate concepts from OntoCAPE. The MTS describes the properties of a desired PMC in terms of the following: (i) the object to be modeled, which is typically a *phase system* or the behavioral aspect of a *chemical process system* (cf. Sects. 7.3 and 8.6.1, respectively); (ii) the desired mathematical and numerical properties of the PMC thereby using concepts from partial model **mathematical_model** (cf. Sect. 9.1); and (iii) the software features of the PMC, which are required for its integration in the user's software environment[111].

The MTS is then verified by the Modeling Task Manager to assure its compliance with the general modeling rules specified in the ontology, thus attempting to discover any contradictory specifications already in advance: For instance, an axiom in partial model **CPS_behavior** (cf. Sect. 8.6) states that a phase of matter cannot simultaneously (a) be ideally mixed and (b) incorporate intra-phase mass diffusion. The Modeling Task Manager, which has a built-in reasoner, will check whether that axiom is violated by the MTS. Thus, if an MTS requests a phase model with these conflicting properties, a warning will be issued that the specification is inconsistent.

Subsequently, the validated MTS is sent to the *Match-Making Agent*, which is responsible for retrieving information about existing PMCs. Such information can be acquired by communicating with the *Library Wrapper Agents*, which represent the contents of the libraries accessible to the COGents system, again by using OntoCAPE. Matching is then performed by the Match-Making Agent, which tries to find one or more PMC that satisfy the requirements stated in the MTS.

[111] Note that OntoCAPE 1.0 still included some software-related modules providing the concepts required for (iii). As explained in Sect. 11.1.3, these modules were later outsourced to a separate software ontology.

Once a desirable PMC is found, it must be integrated into the user's simulation environment. This task is handled by the *Integration Manager*. That particular agent assembles the individual PMCs (which may stem from different libraries) into an executable simulation model and configures it according to the given MTS.

The COGents multi-agent framework has been built upon existing platforms and software tools developed by both the CAPE and the ontological engineering community. A detailed description of the implementation can be found elsewhere (Braunschweig et al. 2004; Yang et al. 2008). The functionality of the prototype has been tested by means of three case studies, which typify representative modeling tasks in the areas of process design, process synthesis, and process simulation. While the multi-agent software is not fully operational and requires further research, particularly on the issue of matchmaking, OntoCAPE has proven itself a reliable communication language, which enables the characterization of complex PMCs and MTSs.

12.1.2 Conceptual Process Modeling

The second application to be described is a prototypical modeling environment, which enables the construction of a mathematical model in two successive steps, named *conceptual modeling* and *model generation*:

- In the first step, the user constructs a *conceptual model* – that is, a physical or phenomenological (not mathematical) characterization of the object to be modeled. The conceptual model defines the essential features of the mathematical model to be developed, such as the boundaries of control volumes or the modeling scale. Thus, this step is comparable to the formulation of an MTS in the COGents system, although a conceptual model is typically more detailed than an MTS.
- In the second step, the conceptual model is automatically transformed into a mathematical model. This task is accomplished by assembling the mathematical model from elementary building blocks, which are selected and configured according to the specifications of the conceptual model.

The modeling environment itself is domain-independent. For conceptual modeling, it requires a domain ontology, which functions as a modeling language for the respective application domain. In the case of process modeling, OntoCAPE provides the concepts for describing structural and phenomenological details of chemical processes.

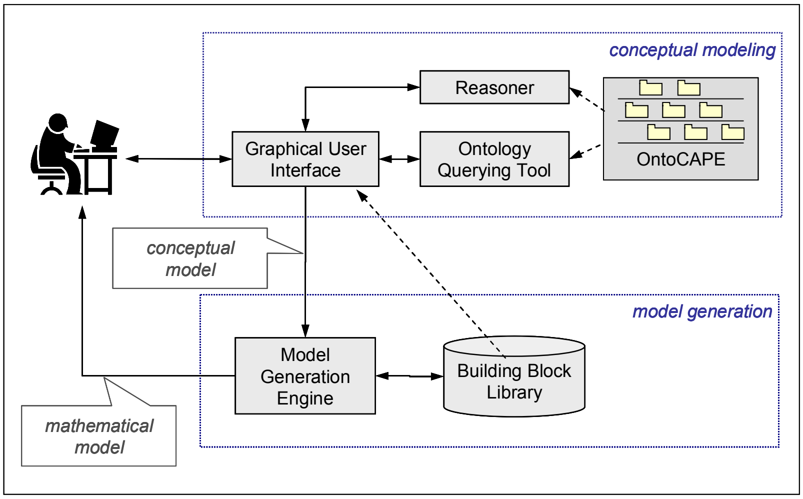

Fig. 12.2: Conceptual modeling approach

In a typical application scenario (Fig. 12.2), the *Ontology Querying Tool* retrieves a set of concepts from OntoCAPE that are relevant to the current modeling context. These concepts are then presented to the user through the *Graphical User Interface* (GUI). Working with the GUI, the user composes the conceptual model by selecting, instantiating, and connecting relevant concepts. The result can be validated by means of a reasoner, which checks if the conceptual model violates any axioms defined in the ontology.

The verified conceptual model is passed on to the *Model Generation Engine* (MGE). Its function is to automatically translate the conceptual model into a mathematical model, which can then be solved within an existent simulation environment. For this task, the MGE relies on a set of elementary model building blocks stored in the *Building Block Library*. Initially, the MGE analyzes the conceptual model in order to identify suitable building blocks, which are then retrieved from the Building Block Library. Each building block consists of a short Modelica[112] statement, typically representing a single model equation. Subsequently, the variable names and parameter values within the selected Modelica statements are customized according to the specifications of the conceptual model. Finally, the individual Modelica statements are combined into an overall model.

While the conceptual model is created, the graphical user interface is aware of the contents of the Building Block Library. It is therefore able to restrict the concepts available for conceptual modeling to those for which a corresponding building block exists. Thus, it can be ensured that the conceptual model is later transformable into a mathematical model.

[112] Modelica (e.g., Tiller 2001) is a free modeling language designed for the mathematical description of physical systems.

Like the COGents framework, the modeling environment has been built upon existing software tools, such as Jena (HP Labs 2007) and ModKit+ (Hackenberg 2006). A detailed account of the implementation and its merits can be found elsewhere (Yang and Marquardt 2004; Yang et al. 2004a).
There are comparable efforts aiming at an improved computer support for process modeling – see for example the publications of Stephanopoulos et al. (1990), Preisig (1995), Marquardt (1996), Perkins et al. (1996), Jensen (1998), Bieszczad (2000), Linninger (2000), Cameron et al. (2001), and Bogusch et al. (2001). The novelty of the approach to process modeling described herein is the following: It can be realized by reusing existing domain ontologies and generic tools developed by the ontology engineering community, hence eliminating the need for developing (i) tool-specific modeling languages and (ii) domain-specific modeling tools. As for (i), the modeling language (i.e., OntoCAPE) is declaredly reusable in various application contexts. As for (ii), the modeling environment can be easily reconfigured to a different application domain, simply by replacing OntoCAPE with another domain ontology. Such a domain ontology must merely be consistent with the environment's core ontology[113], which establishes some high-level modeling concepts such as *system*, *property*, *phenomenon*, *law*, etc[114]. The domain concepts are then declared as refinements of these high-level concepts.

12.1.3 Lessons Learned

For the realization of the above software prototypes, only a few application-specific modules had to be added to OntoCAPE 1.0. Since this required only moderate effort, it can be stated that the ontology has proven to be *usable* for applications in process modeling.
However, it is worth mentioning that, in both applications, OntoCAPE primarily served as a shared vocabulary (cf. Sect. 2.2): in the first case as a communication language between human users and computer agents, in the second case as a domain-specific modeling language. The other possible usage of OntoCAPE – that is, the formal representation of domain knowledge (cf. Sect. 2.2) – could only be tested to a small extent. This is due to the fact that, although the above software applications involve some knowledge-intensive tasks (e.g., checking an MTS or a conceptual model for consistency and completeness), their demands can be met by

[113] Note that the core ontology was called 'meta ontology' in the original publication (Yang et al. 2004a), owing to the genericness of its concepts. However, its function is rather that of a core ontology, as it has been defined in Chap. 2.6, and is therefore referred to as such.

[114] Most of the terms of the core ontology have been included in OntoCAPE 2.0. However, in the core ontology, the terms typically have a more narrow meaning than in OntoCAPE 2.0: For instance, 'system' has the connotation of an 'object to be modeled' rather than that of a general system.

a comparably small part of the entire ontology. From most of the modules of the ontology, however, the applications do not retrieve any formalized knowledge[115]. Consequently, the ontology's usability as a knowledge component library could only be validated in part.

In both application cases, the expressiveness of the ontology was adequate for describing the items addressed in the usage scenarios: That is, the ontological vocabulary has proven suitable for the representation of the mathematical and numerical aspects of process models as well as for representing the modeled objects (typically, behavior of a process plant or material). On the other hand, the usability of those parts of OntoCAPE that are not covered in the application scenarios (i.e., the partial models **CPS_function**, **CPS_realization**, and **CPS_performance**) cold not be validated conclusively.

The *reusability* of the ontology was not greatly challenged by the applications, as both software prototypes are situated in the context of process modeling. Thus, to arrive at any definite conclusion about its reusability, the ontology must be tested in a completely different application context (such as knowledge management, which is discussed in the subsequent section). Yet even the reuse of the ontology between these rather similar process modeling applications revealed the need for a more adaptable model architecture, in general, and for an upper ontology, in particular. To that end, on the basis of the core ontology of the prototypical modeling environment, the first version of such an upper ontology was created. This upper ontology evolved and eventually became the Upper Layer of OntoCAPE 2.0. Accordingly, these software prototypes do not really validate the ontology, but rather present a transition stage, which allowed gaining experiences for the further development of OntoCAPE.

12.2 Knowledge Management in Engineering Design

12.2.1 Introduction

After the completion of the COGents project, the development of OntoCAPE was taken over by the IMPROVE project. Within IMPROVE, the ontology was extensively restructured, thereby addressing the abovementioned need for a more flexible architecture and an upper ontology. In parallel, OntoCAPE was reused in a software prototype called *Process Data Warehouse* (PDW), which was developed within the IMPROVE project by Jarke and co-workers (e.g., Brandt et al. 2006).

[115] Accordingly, large parts of OntoCAPE 1.0 have the character of a lightweight ontology (cf. Sect. 12.4)

The PDW, which supports knowledge management in engineering design processes, will be described in the following.

Engineering design processes involve highly creative and knowledge-intensive tasks that require extensive information exchange and communication among distributed teams. Knowledge about engineering design processes constitutes one of the most valuable assets of a modern enterprise. Normally, this knowledge is only known *implicitly* to the participating designers, relying heavily on the personal experience and background of each designer. To fully exploit this intellectual capital, it must be made *explicit* and shared among designers and across the enterprise.

Knowledge management (*KM*) is a scientific discipline that stems from management theory and concentrates on the systematic creation, leverage, sharing, and reuse of knowledge resources in a company (Awad and Ghaziri 2003). Knowledge management approaches are generally divided into personalization approaches that focus on human resources and communication, and codification approaches that emphasize the collection and organization of knowledge (McMahon et al. 2004). Here, only the latter approach is considered. In particular, the PDW addresses the capture and reuse of *experience knowledge*: This term summarizes all the potentially reusable knowledge that emerges over the course of an engineering design project.

In this context, two different types of experience knowledge need to be distinguished: On the one hand, there are the *products* created during the design process (e.g., simulation models, design calculations, cost estimates, etc.); these may be organized into documents, which act as logical (or virtual) and sometimes as real units to enable work distribution or version control. On the other hand, there are the (*work) processes* or activities themselves, in which the products are created, used, or manipulated. The prominent concern of any successful KM approach must be the integration of *product knowledge* and (work) *process knowledge* in a coherent framework.

12.2.2 Approach

There is a plethora of software tools addressing engineering knowledge management inside manufacturing enterprises. *Document management systems* such as Windream (Windream 2008) or Documentum (EMC^2 2008) are widely used in industrial praxis for the storage, maintenance, and distribution of documents. A step further, *Product Data Management (PDM)* systems provide extended facilities for the handling of detailed product information, ranging from design to production stage. They are being succeeded by *Product Lifecycle Management (PLM*) systems, such as Windchill (PTC 2008), TeamCenter (Siemens PLM Software 2008) or CATIA (Dassault Systemes 2008). The aim of these systems is to integrate information on the manufacturing processes (usually CAM systems) with design data (CAD systems) on the one hand, and information about Enterprise Resource Planning (ERP) processes on the other hand.

The PDM/PLM systems available today adequately support information exchange between developers, especially in the later phases of the engineering lifecycle which are characterized by more deterministic[116] and well-known processes. However, they lack essential capabilities for the management and reuse of design knowledge (Bilgic and Rock 1997; Gao et al. 2003; Maropoulos 2003).

A significant shortcoming of existing PDM and PLM systems criticized by many authors is their lack of adequate information models for product representation (e.g., Szykman et al. 2001). These models would be needed to effectively capture, exchange, retrieve, and reuse design knowledge. In particular, formal and well-structured information models for the conceptual design stage are missing (e.g., Bilgic and Rock 1997; Szykman et al. 2001; Gao et al. 2003; Mesihovic et al. 2004).

Regarding process support, current PDM systems have largely focused on the support of micro-level processes on the administrative level, such as versioning or engineering change management (Mesihovic et al. 2004). Some attention is paid to project management, however without reaching the capabilities of full-fledged project management systems (Bilgic and Rock 1997; Mesihovic et al. 2004). They lack the functionality to capture complex work processes and decisions. They are particularly inappropriate for conceptual design processes (e.g. Douglas 1988; Biegler et al. 1997), which are highly creative and dynamic processes and thus hardly predictable (Westerberg et al. 1997; Marqardt and Nagl 2004). Any software solution has to cope with the continually changing requirements and the many degrees of freedom within these processes. Because of their hard-wired usage processes and the restricted ability for interoperation, the software tools available today are unable to offer appropriate support.

Moving beyond the established approaches, a novel type of knowledge management system has been developed, which has been named the Process Data Warehouse, or PDW in short. Basically, the PDW supports the mining of experience knowledge and its reuse on demand. Its functionality can be summarized as follows.

Typically, discrete items of product knowledge are stored in heterogeneous sources such as electronic documents and data bases, which are distributed across the enterprise. The PDW (i) provides a comprehensive representation of the contents of these sources, thereby correlating the scattered knowledge items and providing a single point of access to design knowledge. As such a comprehensive representation cannot be complete (for reasons of scaling, maintainability, practicability, etc.), the PDW (ii) supplies mechanisms for easily locating the original knowledge sources, where more detailed information can be retrieved. To this aim, meta information about the sources (e.g., type, structure, version history, storage location) is combined with information about their contents. Moreover, the

[116] A deterministic work process can be completely planned and/or scheduled in advance.

PDW has been integrated with existing tools and data stores to promote easy access to the original sources. Last but not least, the PDW (iii) enables the capture and archival of process knowledge (i.e., the actual work processes during the design lifecycle) in order to provide information about the circumstances in which the product knowledge has been created. In particular, recording of the decision-making procedures allows recalling the design rationale applied at that time. Thus, process and product knowledge are captured in an integrated manner. This allows the systematic retrieval of experience knowledge that is suitable for a particular situation or working context.

The concept of the PDW has been derived from the concept of Data Warehousing (Jarke et al. 2003), where large amounts of structured data (e.g., from sales or accounting) are stored, aggregated, and then presented. For those conventional tools and warehouses, fixed schemas are used for data storage. Yet to support design and other creative work processes, a dynamic extension of the existing data structures and the integration of additional, suitable domain models must be supported (Jarke et al. 2000). To this end, the PDW uses formal ontologies for the representation and storage of experience knowledge. Ontologies have two major advantages over conventional data schemas: Firstly, they are highly flexible, enabling modifications and extensions of the data structures even during project execution and thus facilitating the handling of the dynamic design processes. Secondly, they enable the computer to interpret and reason with the information stored in the ontology. In consequence, advanced support for knowledge management and knowledge retrieval can be provided.

12.2.3 The Core Ontology

Knowledge representation within the PDW is realized via a set of loosely connected ontologies, which are held together by a central *Core Ontology* (Brandt et al. 2006; 2008b; cf. Sect. 2.6). The Core Ontology introduces top-level concepts that describe products and processes, as well as their interrelations and dependencies, independently from any particular domain or application. These fundamental concepts are then refined and concretized within the *Peripheral Ontologies*. Different Peripheral Ontologies can be added to the Core Ontology to flexibly adapt the PDW to the requirements of a specific application domain.

Fig. 12.3 displays a simplified view of the Core Ontology. Four prominent areas of conceptualization are arranged around the *object* as the central concept.

- The *Product Area* (top) contains concepts for the description of the type and version history of electronic documents and other information resources, as well as their mutual dependencies and their structural decomposition. The *product* class denotes all kinds of information elements, such as data items or decision representation objects, which are created or modified during a work process. Different versions of a *product* may exist, which can be bundled into a *version set*. Contextually related *products* can be aggregated into a *document version*. A (logical) *document* bundles different *document versions* and is thus a specialization of a *version set*.

– The *Storage Area* (right) describes at which location (*storage place*) inside a *store* a particular *version set* is stored. Examples of *stores* are data bases, document management systems, and external tools.
– The *Descriptive Area* (left) contains basic concepts for describing the content or the role of *product objects* on a semantic level. This includes *content descriptions* and *categorizations*. Unlike *categorizations*, which simply classify *product objects* according to certain categories, a *content description* is a placeholder for a term (or even a composite expression) from a controlled vocabulary that characterizes the contents of the associated *product object*. Thus, the descriptive area provides content-specific meta information for the annotation of documents and data stores.

The *Process Area* (bottom) represents the *process objects* that manipulate (i.e., create, use, or modify) *product objects*. In particular, it comprises the *activities*, which are performed by *users* or (software) *tools*.

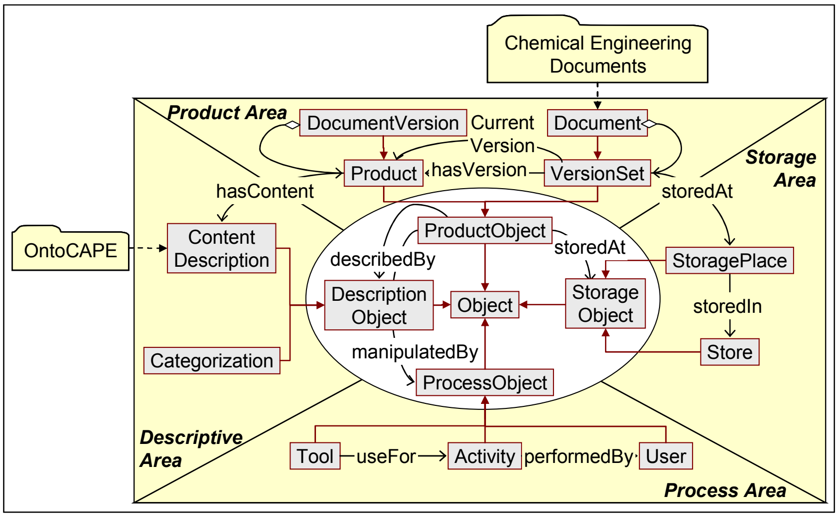

Fig. 12.3: Simplified view of the Core Ontology and some Peripheral Ontologies

The Core Ontology only contains high-level concepts required to establish, organize, and integrate the four fundamental Areas. In addition, each Area offers some *extension points* where Peripheral Ontologies for specific application domains or other specializations can be added. These ontologies refine the domain independent concepts introduced in the Core Ontology. Two exemplary extensions are shown in Fig. 12.3 (refinement is indicated by dashed arrows):

– OntoCAPE is the most substantial of these extensions. In the context of this application, the concepts of OntoCAPE are used as refinements of the *content description*, thus providing a shared vocabulary for the description of *product objects* and *process objects*. If *product* or *process objects* are to be described that do not fall in the domain of CAPE, OntoCAPE can be replaced by a different domain ontology (for example, Raddatz et al. 2006, describe an application of the PDW in the a sub-domain of rubber production).
– The ontology *Chemical Engineering Documents* refines the generic *document* class in the Product Area. It provides a taxonomy of the document types that are typically used in chemical process design, such as Process and Instrumentation Diagrams, equipment lists, or model files. A detailed description of this ontology has been compiled by Morbach et al. (2008c). In an analogous manner, alternative taxonomies can be developed by the users of the PDW to represent the document types that exist in the users' organization.

12.2.4 Implementation of the PDW

The PDW has been implemented on top of an existing ontology framework. For reasons of expressiveness, reasoning support, and interoperability, a framework supporting the OWL ontology language would have been first choice for the realization of the PDW. Unfortunately, the OWL-based repositories available at that time could not efficiently handle large amounts of instance data and thus were not usable for real-world applications. For that reason, it was decided to relinquish some of the expressive power of OWL in exchange for improved scalability. The KAON system (Oberle et al. 2004), based on which the PDW has been realized, therefore constitutes a reasonable compromise: It enables efficient storage and retrieval of instance data, at the cost of loosing some of the description-logic capabilities of OWL (cf. Brandt et al. 2008a). Since KAON complies with the same set of base standards as OWL (i.e., RDF and RDFS; cf. Brickley and Guha 2004), the Peripheral Ontologies, most of which were originally implemented in OWL, could be relatively easily transferred in the KAON system.
Extending the KAON system, the PDW offers specialized functionality for knowledge management. In the following, the major issues of the PDW architecture are summarized; details about the implementation can be found elsewhere (e.g., Brandt et al. 2008a, 2008b; Miatidis et al. 2008):

– The PDW is integrated with an external document management system, which is responsible for supporting document-based storages, offering version management and change notification support. The ontology instances corresponding to the document, its new version, and its storage place are automatically created or updated within the Product Area and Storage Area of the Core Ontology.

– For particular formats, such as the model files of the simulator Aspen Plus (AspenTech 2008), specialized converters have been built, which automatically annotate the documents' contents through concepts of the Descriptive Area (e.g., Amin and Morbach 2007; cf. Sect. 3.6).
– External tools that use repositories or databases for storage purposes instead of simple files or documents can also be integrated with the PDW. A generic mechanism for integrating such external data sources has been implemented using XML (Bray et al. 2006b) as an exchange format, and XSLT (Clark 1999) to transform the intermediate files into a form that matches the conceptual representation of the PDW.
– Furthermore, the PDW is integrated with the PRIME (Process-Integrated Modeling Environment) system, a prototypical design environment for chemical engineering (Pohl et al. 1999; Jarke et al. 1999; Miatidis and Jarke 2005). Within the PRIME system, the actions performed by the designer are (semi-)automatically traced, as well as the products worked upon by these actions. The recorded experience knowledge is represented through the concepts of the Core Ontology and stored in the PDW.
– Several specialized user interfaces support the retrieval of the recorded experience knowledge. A particular knowledge item can be found in two different ways: either indirectly by browsing a graphic representation of the ontology instances, or directly via a query interface enabling a semantic search. The latter enables a highly systematic retrieval of information: First, a query to the PDW is composed from arbitrary concepts (classes, instances, relations, attributes) of the PDW ontologies. Next, the inference engine of the PDW processes the query and tries to match it against the available knowledge resources, which are again represented by terms of the PDW ontologies. Since the semantics of the ontology terms have been formally defined, the search engine can interpret the meaning of both query and annotations in order to find a semantic match, thus achieving better recall and precision than a conventional search based on string comparisons.

Comparable ontology-based repositories for design knowledge are currently being developed in other areas than process engineering – e.g., Kopena and Regli (2003) or Szykman et al. (2000) in electromechanical engineering. A review of the recent developments in this area is given by Szykman et al. (2001). However, these repositories are limited to the storage of product data and documents and do not record the associated work processes and decision-making procedures.

Some recent research projects try to extend the capabilities of PDM/PLM systems towards knowledge management (e.g., Kim et al. 2001; Gao et al. 2003). Yet knowledge must be entered manually and explicitly, since the suggested ap-

proaches lack the recording capabilities of the PDW. Moreover, they assume deterministic processes, which is not the case in conceptual process design.
An ontological architecture for knowledge management, which resembles the Core Ontology described herein, has been proposed by Abecker et al. (1998). Yet the application of these concepts to engineering domains does not fall into the scope of this particular research group.
In the area of chemical engineering, several prototypical design environments have been developed based on ontological models and tools (Bañares-Alcántara and Lababidi 1995; Batres et al. 1999; Kitamura and Mizoguchi 2003; Venkatasubramanian et al. 2006; Bañares-Alcántara et al. 2003). Unlike the PDW, these environments focus on other aspects than the reuse of experience knowledge.

12.2.5 Lessons Learned

The applicability of the PDW (and consequently that of OntoCAPE) has been demonstrated by means of several use cases (cf. Brandt et al. 2008a), including a major application scenario about the conceptual design of a polyamide-6 production facility (cf. Eggersmann et al. 2002). By testing the PDW and its ontologies against these use cases, OntoCAPE has been progressively refined:
Initially, an older version of OntoCAPE (named OntoCAPE 1.1) was used within the PDW. This ontology was created by firstly translating OntoCAPE 1.0 from DAML+OIL into OWL and secondly implementing those changes with respect to the model architecture that were considered necessary due to earlier testing of OntoCAPE 1.0 in the prototypical modeling environment (cf. Sect. 12.1.3). As a result, the architecture of OntoCAPE 1.1 already largely resembled that of OntoCAPE 2.0, yet the individual modules were still conceptualized as those of OntoCAPE 1.0.
Generally speaking, OntoCAPE 1.1 proved to be reusable: The new architecture enabled an easy integration with the other PDW ontologies, and the expressivity of vocabulary proved sufficient for the annotation of the knowledge resources. Some partial models had to be extended – particularly **CPS_function** and **CPS_realization**; yet this was to be expected since these partial models were not really required by the earlier process modeling applications of OntoCAPE 1.0 (cf. Sect. 12.1.3) and were therefore only fragmentarily developed.
At the same time, however, testing revealed that the usability of OntoCAPE 1.1 was insufficient for the PDW. Basically, it suffered from two major types of defects:

– It turned out that OntoCAPE 1.1 had serious deficiencies with respect to the design criteria of conciseness, coherence, intelligibility, and performance. These issues are extensively discussed in Sect. 3.1.2 and Chap. 10 as well as in Sects. 12.4.2 and 12.4.3.

– Most parts of OntoCAPE 1.1 still had the character of a lightweight ontology and thus an insufficient level of axiomatization. While this is rather irrelevant for the annotation of knowledge resources, per se, it becomes important when knowledge is to be retrieved by a semantic search: For inferring the semantic equivalence between a query and an annotated knowledge resource, the inference engine requires the semantics of the query and the annotation terms to be formally defined. This can only be achieved if a preferably large number of axioms clarify the meaning of the ontology classes and relations.

Due to these defects, OntoCAPE was repeatedly revised and thus gradually improved. In the course of this process, the conceptualization of the individual models was optimized with respect to the design criteria of coherence, conciseness, intelligibility, and performance, as described in Chap. 10. Concurrently, the level of axiomatization was increased, as documented in Sect. 12.4. These efforts finally cumulated in the design of OntoCAPE 2.0, which fulfills all requirements stated by the use cases of the PDW.

12.3 Integration of Design Information

12.3.1 Introduction

A shortcoming of all the OntoCAPE-based software prototypes presented so far is that they have only been tested in academic use cases of limited scope[117]. Such use cases are sufficient to demonstrate the basic functioning of the software as well as the ontology's fundamental capability for (re)use; yet they cannot guarantee the ontology's suitability for real-world applications.
To address this issue, OntoCAPE is currently being field-tested in a large-scale industrial project, which is run in cooperation with partners from the chemical and software industries (Morbach and Marquardt 2008). The project objective is the development of an ontology-based software prototype for the integration and reconciliation of design information. Extensive testing will ensure that both the integration tool and the ontology comply with the challenging requirements of industrial practice.
The objective of the integration tool is to support information handling in engineering design projects. In the course of a typical design project, information is created by disparate software tools and stored in heterogeneous sources, such as technical documents, CAE systems, or simulation files. Eventually, the scattered

[117] Note that the PDW has been tested in industrial practice (Raddatz et al. 2006), however not in combination with OntoCAPE.

information items need to be merged and consolidated. Unfortunately, the structural, syntactic, and semantic heterogeneities of the different sources hinder the provision of efficient computer support.

Commercial software systems for the integration and management of process design data still rely on conventional database technology. Thus, the approaches primarily address the integration of structural (schematic) heterogeneities. Most of these systems are based on a central data warehouse that can import and redistribute data created by application tools. These systems support navigation and retrieval of data and provide some basic data management functions such as version management and change management. Typical examples are SmartPlant Foundation (Intergraph 2008) from Intergraph and VNET (AVEVA 2008) from AVEVA. However, these off-the-shelf solutions can only process the proprietary data formats of the respective vendors and are thus limited to the data created by the vendors' application tools, which normally represent only a minor subset of all data created in the course of a development project. Extending these tools towards the handling of non-proprietary formats is difficult, as one need to modify the source code and map onto the internal data models of the tools, both of which are poorly documented in most cases.

Thus, information integration is yet largely performed manually, thereby creating a significant overhead for the designers (cf., e.g., Gallaher et al. 2004).

To overcome these deficiencies, XML has recently gained acceptance as a way of providing a common syntax for exchanging heterogeneous information. XML is increasingly applied for data exchange, ultimately becoming a standard for data interchange between software tools (Klein 2002). In the area of chemical engineering, various XML-based solutions for data exchange are currently under development or already in use, such as CAEX (Fedai and Drath 2004), XMpLant (Noumonon 2008), or PlantXML (Anhäuser et al. 2004). The latter has been established by the engineering department of Evonik Degussa to improve the interoperability of their application tools. Data exchange is realized via XML files that comply with an in-house standard. PlantXML defines specific XML schemata for the different phases and functions of a design project: XML-EQP for the design of machines and apparatuses, XML-EMR for the design of instruments and control systems, XML-RLT for pipe engineering, and XML-SiAr for the design of fittings and safety valves. Custom-made converters handle the import and export of the XML files to and from the application tools by mediating between the in-house standard and the internal data models of the application tools.

Although solutions like PlantXML do not directly support information integration, they can greatly facilitate that task: By translating the information from the proprietary formats of the disparate application tools and data stores into a uniform XML format, they resolve the structural and syntactic heterogeneities of the different information sources. For that reason, it was decided to build the integration tool on top of an XML-based solution for data exchange. The abovementioned PlantXML has been chosen as the first application case, but the tool is designed in such a way that can handle arbitrary XML formats.

The use of ontologies and semantic technologies for the integration of distributed data is a long-standing research issue; an overview on the more recent contributions in this field can be found elsewhere (Wache et al. 2001; Paton et al 2000; Visser et al. 2000; Crubézy et al. 2005). However, for the time being there are very few approaches which emphasize on the domain of engineering. None of these approaches aims at the consolidation of large amounts of distributed engineering design data. Thus, to the authors' knowledge, the software prototype presented herein is unique in that respect.

12.3.2 Approach

From our perspective, the solution to the problem of semantic heterogeneity is to formally specify the meaning of terminology of each system; from this, the computer can autonomously infer a translation between the different system terminologies. Unfortunately, the semantics assumed by a particular source are rarely documented, and there is no explicit representation of a data source's semantics (at least not in the same way as a data schema provides a representation of the data structures). Hence, the important link missing at this point is the connection between the structured information stored in the XML document and the domain knowledge, which relates meaning to the stored information within the particular context. In other words, a semantic annotation of the XML documents is necessary. To that end, OntoCAPE is perfectly suited as a controlled vocabulary for annotation, similarly as it has been used in the PDW (cf. Sect. 12.2.3).

As stated before, the software prototype will make use of the information available in the PlantXML format. To this end, appropriate converters will extract selected data from the XML files and assemble them in a *comprehensive information base*, which has the function of integrating and consolidating the distributed information. For interaction and visualization, a GUI has been developed which is particularly adapted to the design engineers' needs. Finally, the consolidated, possibly integrated data can be transferred back to the corresponding software tools and thus the consolidated data can be directly reintegrated in the standardized workflow. The entire information consolidation process is shown in Fig. 12.4.

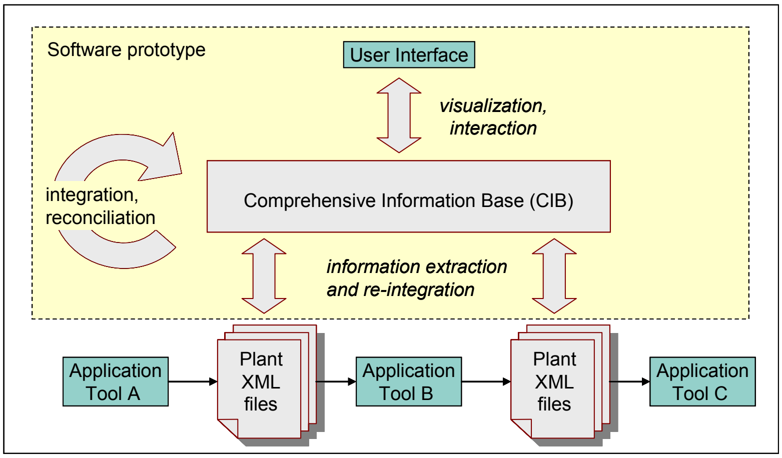
Software prototype
User Interface
visualization, interaction
integration, reconciliation
Comprehensive Information Base (CIB)
information extraction and re-integration
Application Tool A
Plant XML files
Application Tool B
Plant XML files
Application Tool C

Fig. 12.4: Information consolidation process

This software prototype basically represents a mediation layer, which is placed between the user and the data sources (Wiesner et al. 2008b). The tool follows the so-called hybrid approach (e.g. Paton et al. 2000; Wache et al. 2001), as shown in Fig. 12.5:

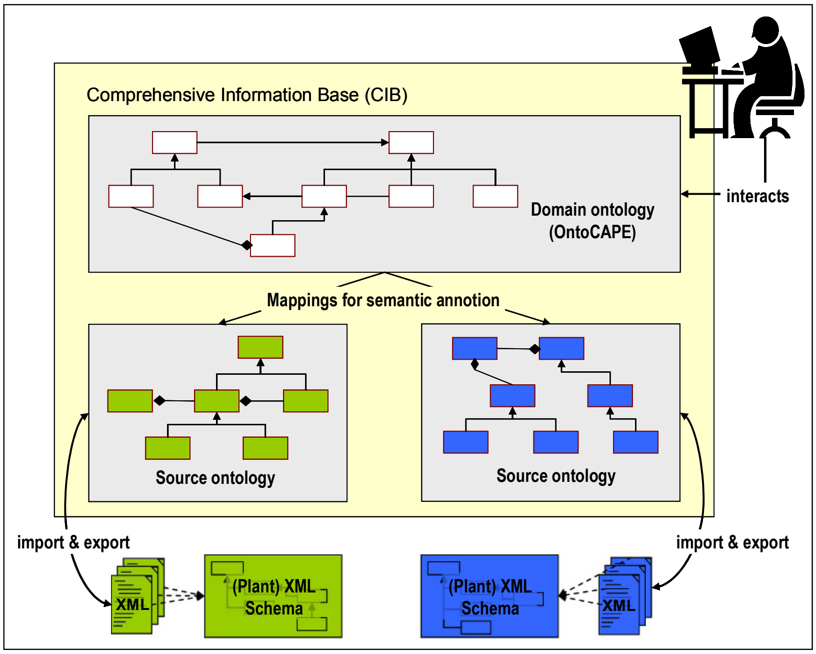

Fig. 12.5: Schematic representation of the CIB

Each information source (i.e., each XML document) is described by a local source ontology, which basically replicates the document (its original structure as well as its data) in the ontology language of the integration tool. The interactions between a source ontology and its corresponding XML document are handled by a bidirectional converter, which mediates between the XML representation and the ontology language of the integration tool. This rather complex procedure is called "semantic lifting" and it follows a two step approach: step (1) is to lift the XML schema to the level of an ontology: i.e., a skeleton source ontology is created, which incorporates only the (structural) information from the schema expressed in an ontology language. Step (2) establishes further relations between concepts and attributes in order to clarify context information, finally leading to a complete source ontology (cf. Fig. 12.5).

Via mappings, the entities of the source ontologies are linked to the entities of a global domain ontology (OntoCAPE in our case), thus making their semantics explicit. Lastly, by means of a graphical user interface (GUI), information queries can be made via the domain ontology, which functions as a global query schema[118]: The user formulates a global query in the terminology of the domain ontology. Next, the inference engine transfers the query to the source ontologies via the existing mappings. Then, the inference engine retrieves and combines matching information from the different source ontologies. Finally, the results are retranslated into the terminology of the domain ontology and presented to the user.

The new approach assumes the tool to be layered on top of an existing XML-based data exchange architecture. According to PlantXML, it is realized in the sense of "publish and subscribe" (Britt et al. 2004) for serialization.

The information processing in the integration prototype is performed in two consecutive steps:

(1) **Extraction and preprocessing**. In the first step, the relevant information is extracted from the XML files and transferred into the integration tool. Within the tool, the information stemming from different XML documents is kept separately and is individually prepared for integration. In this context, the term 'XML document' denotes a set, the members of which are XML files that (i) originate from the same source and (ii) conform to the same XML Schema (e.g. XML-RLT). Such XML files are considered as different versions of the same XML document. Preparation includes the detection and elimination of outdated and redundant information, as well as the resolution of versioning conflicts. Such preprocessing is necessary since there may be several versions of an XML document. Typically, no stringent version management exists for these, due to the complex workflow of a design project (split of work, concurrent engineering, distributed engineering …). In other words each version may

[118] In database engineering, this kind of approach is called "local-as-view" (e.g. Halevy 2001).

include both current information as well as outdated and/or redundant information. The resulting versioning conflicts need to be resolved before integration.

(2) **Integration and reconciliation**. In this step, the hitherto separated information from different documents is integrated. Integration includes (i) the linking of logically correlated information items and (ii) the detection of design errors. An example for (i) and (ii) would be the following: Consider two data sets (originating from separate XML documents), where the first data set specifies the geometry of a vessel, and the second data set specifies the geometry of a pipe. As for (i), the tool would detect that the pipe is connected to a nozzle of the vessel, and thus establish a link between those two. As for (ii), the tool would then discover that internal diameters of pipe and nozzle are not identical; it would thus issue a warning that the design is inconsistent.

Information processing within the integration tool is controlled by rules (cf. Sect. 2.3.6), which are executed by a built-in inference engine. Two sets of rules need to be distinguished, which are stored in two separate ontologies:

- The *Information Versioning Ontology*[119] contains the rules that are required for information extraction and preprocessing (step 1). These rules are domain-independent; they interact only with the source ontologies.
- The *Information Integration Ontology*[120] defines the rules for information integration and reconciliation (step 2). These rules are domain-specific. Their antecedents and consequents are formulated by means of vocabulary terms provided by the domain ontology.

In step 1 (extraction and preprocessing), only the source ontologies and the Information Versioning Ontology are involved. Here, the major issue is to resolve the versioning conflicts. For this task, the inference engine does not have to understand the contents of the source ontologies in detail; it only needs to know the structures of the XML documents as well as the creation dates of their respective versions.
However, in order to perform step 2 (integration and reconciliation), the inference engine needs to understand the contents of the source ontologies. To this end, the semantics of the information contained in the source ontologies must be formally specified. This is achieved by annotating the elements of the source ontologies with concepts from the global domain ontology. That way, the semantics of the information are made explicit, and the inference engine can now apply the rules of

[119] According to the classification framework introduced Sect. 2.6, the Information Versioning Ontology can be classified as a task ontology.

[120] Thus, the Information Integration Ontology can be classified as a domain-task ontology (cf. Sect. 2.6).

the Information Integration Ontology on the source ontologies. A schematic overview of the integration tool is given in Fig. 12.6.
Once information integration has been completed, one has obtained a *consolidated comprehensive information base* (CCIB) reflecting the current status of the design project. The CCIB can then be accessed via the GUI by any member of the project team who requires validated and up-to-date design information. Moreover, the project manager can extract key figures and performance indicators from the CCIB; exemplary key figures would be the accumulated equipment cost or the percentage of completed sub-tasks.

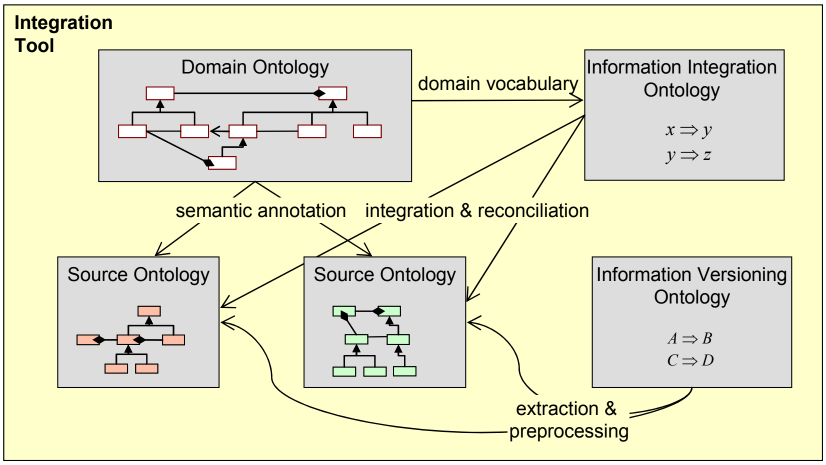

Fig. 12.6: Schematic representation of the integration tool

12.3.3 Implementation

The formulation of rules requires a special ontology language: The deductive ontology language F-Logic (Kifer et al. 1995) has been chosen for this purpose. Deduction is particularly useful for merging and consolidating distributed information (Maier et al. 2003), and thus it was decided to employ a deductive language (and a compatible inference engine) for the software prototype. It allows not only the definition of rules for integration and mapping purposes, but also the formulation of queries. Also, the inference mechanism of deduction is especially applicable to partially automating the information integration process. Finally, the deductive ontology language is more intuitive and less error-prone than conventional database languages, especially in complex contexts with many relations between the data objects (Maier et al. 2003).

Moreover, all converters and the GUI are based on the Java programming language, using Eclipse (Eclipse 2009) as development environment.
An implementation basis had to be chosen which fulfils the requirements for both technologies equally well. Accordingly, the development system OntoStudio (Ontoprise 2009), which has been developed by the project partner ontoprise, serves as the implementation basis. Unlike most of the ontology-based systems available today, OntoStudio comes with a built-in inference engine that is scalable for large data sets; it thus provides an appropriate implementation basis for the integration tool. Of course, the downside of using OntoStudio is that the relevant parts of OntoCAPE had to be translated from OWL into F-Logic. Details about the implementation have been reported by Wiesner et al. (2008b).

12.3.4 Lessons Learned

To date (as of 2009), the core of the implementation has been completed: The ontologies of the integration tool have been established and tested against small to medium-size quantities of real-world data. Moreover, a new interface has been implemented in OntoStudio, which allows the import/export of XML files (for details on the implementation see Wiesner et al. 2008b). So far, the integration tool has fulfilled all requirements for industrial use. In the future, the software will be further tested with larger amounts of data and more complex integration tasks to prove the suitability of the tool as well as the (re)usability of OntoCAPE for applications in the chemical industry.

12.4 Some Measures of Improvement

This section presents some measures to compare OntoCAPE 1.0 und 2.0 on a quantitative basis, thus accounting for the improvement of the ontology in the course of the software projects described in this chapter. We discuss how the data can be interpreted with respect to a correct choice of the design principles introduced in Chap. 10. Section 12.4.1 presents the statistical data, which are discussed in the subsequent Sect. 12.4.2 and 12.4.3 with respect to the design principles of coherence and conciseness. This attempt to quantitatively assess – at least to some extent – the degree of quality of an ontology with respect to usability and reusability is inevitably incomplete and may be criticized for not meeting its objectives. Still, we argue that the following statistical data and subsequent interpretation provides a modest first step towards the highly desirable quantitative assessment of ontology improvement based on few key figures.

12.4.1 Statistical Data

In Fig. 12.7 some statistical data about OntoCAPE 1.0 and 2.0 are summarized, which have been provided by the Protégé ontology editor. Given are

– the total number of classes as well as the subset of defined classes;
– the number of relations;
– the number of local constraints (cf. Chap. 2.3.4) as well as the subset of local range restrictions.

With respect to the number of local constraints, it should be noted that OntoCAPE 1.0 originally included 746. However, later testing[121] revealed that 46 of these constraints were redundant, such that the genuine number of constraints is 700 only.

	OntoCAPE 1.0	OntoCAPE 2.0 [1)]
# o. classes	576	472
# o. defined classes	36	140
# o. relations	390	210
# o. local constraints	700	1041
# o. local range restrictions	109	368

[1)] Upper three layers including the Meta Model

Fig. 12.7: Statistical data of OntoCAPE

Do also note that only the application-independent part of OntoCAPE 2.0 is considered here (i.e., the Meta Layer, the Upper Layer, and the Conceptual Layer). The application-dependent extensions are not considered since the different applications require disparate extensions and thus produce dissimilar statistical data.
Due to the large differences in scope and conceptualization between OntoCAPE 1.0 and 2.0, the absolute numbers listed in Fig. 12.7 are not directly comparable. More conclusive are the relative numbers presented in Fig. 12.8, which set the above numbers in relation to the total number of classes. Given are the *class-to-relation ratio*, the *local-constraint-to-class ratio*, the *local-range-restriction-to-class ratio*, and the *defined-to-primitive-classes ratio*. These relative numbers will be revisited and interpreted in the following sections.

[121] During the development of OntoCAPE 1.0, the functionality for redundancy testing (cf. Sect. 10.3) was not yet provided by the existing ontology editors.

	OntoCAPE 1.0	OntoCAPE 2.0 [1)]
Class-to-relation ratio	1.48	2.25
Local-constraint-to-class ratio	1.22	2.21
Local-range-restriction-to-class ratio	0.19	0.78
Defined-to-primitive-classes ratio	0.07	0.42

[1)] Upper three layers including the Meta Model

Fig. 12.8: Key figures of OntoCAPE

12.4.2 Improvement in Coherence

There were a number of incoherencies – i.e., logical inconsistencies and modeling errors – in the formal specification of OntoCAPE 1.0, which have later been rectified in OntoCAPE 2.0. Among those were the following:

- Sanity tests performed with the Protégé ontology editor revealed 13 cases of relation property mismatches[122]. Consistency checking with the reasoner RacerPro furthermore discovered four inconsistent class definitions. These errors have been corrected in OntoCAPE 2.0
- The modelers of OntoCAPE 1.0 were obviously not aware of the differences between a local and a global range restriction: The former is merely a constraint that produces an error if violated. The latter, by contrast, implies that any target of the respective relation is automatically inferred to be a subclass of the declared range class (cf., e.g., Rector et al. 2004). In OntoCAPE 1.0, global range restrictions have often been utilized where local range restrictions would have been more appropriate. This has been corrected in OntoCAPE 2.0, as can be seen on the basis of the *local-range-restrictions-to-class ratio* introduced above: Its value has increased from 0.19 to 0.79.
- Similar considerations apply to the usage of global domain restrictions. Their erroneous use may even lead to unintended semantics: For example, in OntoCAPE 1.0, the class *problem statement* has been declared as a global domain for the relation has_chemical_reaction – this wrongly implies that anything which has a chemical reaction is inferred to be a *problem statement*. Such obvious modeling errors have been rectified in OntoCAPE 2.0.

Besides the above deficiencies, the coherence of OntoCAPE 1.0 is also impaired by the incomplete axiomatization of its term definitions, which prevents proper

[122] A relation property mismatch occurs if the properties of some subrelation do not match those of the superrelation.

testing for consistency (cf. Sect. 10.1). This, amongst others, includes the following issues:

- Hardly any siblings are declared to be mutually disjoint.
- Similarly, the instances of a common class are not stated to be mutually distinct.
- Very few relation properties (such as functionality or transitivity) are declared.
- The ontology contains only a small number of defined classes.
- Last but not least, the term definitions are not properly "closed off" (cf. Sect. 10.1), thus not accounting for the open world assumption made by DL reasoners.

Considering these findings, OntoCAPE 1.0 must be classified as a lightweight ontology. By contrast, OntoCAPE 2.0 is characterized by a much higher degree of axiomatization qualifying it as a heavyweight ontology. The progress made between the two versions is reflected by the statistical data presented in the previous section:

- The value of the *local-constraint-to-class-ratio* has increased from 1.22 to 2.21, which represents an improvement of 80%.
- The *local-range-restrictions-to-class ratio* has considerably risen from 0.19 to 0.78 (i.e., by a factor of 4). This increase reflects the progress made on the issue of closure axioms, as local range restrictions are one possible means for closing off a term definition (cf. Sect. 10.1).
- Most notable is the raise of the *defined-to-primitive-classes ratio* from 0.07 to 0.41, which constitutes an improvement by a factor of 6.

As for the declaration of relation properties and disjointness axioms, their numbers have been significantly increased, as well. Unfortunately, these numbers are not measured by the Protégé ontology editor, such that the increase cannot be quantified.

12.4.3 Improvement in Conciseness

Advancing in conciseness – that is, simplifying the conceptualization without loosing expressiveness – has been a design priority for OntoCAPE 2.0. As already discussed in Sect. 10.2, the progress made on this issue cannot be quantified easily. However, the increase of the *class-to-relation ratio* may serve as an indicator for the improvement between the two versions:
OntoCAPE 1.0 is characterized by a *class-to-relation ratio* of only 1.48. This rather low value signifies an unnecessary large number of relations, which principally arises from two different causes:

- Firstly, instead of systematically reusing existing relations, they are redefined multiple times under different names in OntoCAPE 1.0. This is exemplified by the conceptualization presented in Sect. 10.2, Fig. 10.1, where the relations hasPort and hasConnection are introduced instead of reusing the existing relation hasPart. This not only disagrees with the principle of conciseness, but also makes it more difficult for the user to understand and apply the ontology.
- Secondly, OntoCAPE 1.0 frequently introduces specialized subrelations in order to represent semantics that should be represented through specialized classes, instead. A typical pattern of this type is presented in Fig. 12.9: In OntoCAPE 1.0, specialized system properties are usually represented through a specialization of the hasProperty relation referring to the general *property* class. By contrast, OntoCAPE 2.0 uses a specialization of the *property* class, which is referred to via the general hasProperty relation.

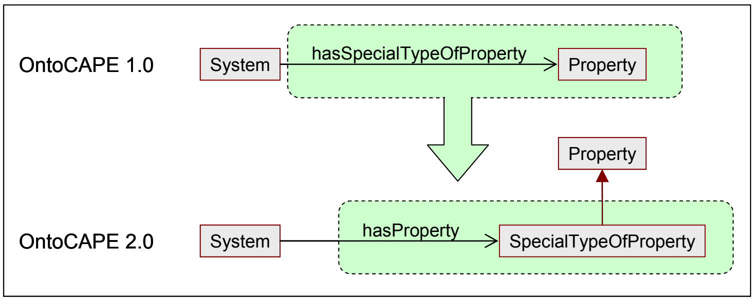

Fig. 12.9: Representation of specialized properties in OntoCAPE 1.0 and 2.0

The pattern utilized in OntoCAPE 2.0 allows for a higher degree of axiomatizations as a class can be further defined through axioms while a relation cannot – for example, it could be defined what differentiates this particular property from another specialized property. Moreover, in our judgment, the 2.0 pattern is more intuitive and better captures the intended semantics.

By improving on these two issues, OntoCAPE 2.0 has significantly cut down on the number of relations. This is reflected by a *class-to-relation ratio* of 2.25, which constitutes an advancement of 50%.

12.5 References

Abecker A, Bernardi A, Hinkelmann K, Kühn O, Sintek M (1998) Toward a technology for organizational memories. *IEEE Intell. Syst.* **13** (3):40–48.

Amin MA, Morbach J (2007) *XML to OWL Converter*. Technical Report (LPT-2007-02), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Anhäuser F, Richert H, Temmen H (2004) Degussa PlantXML – integrierter Planungsprozess mit flexiblen Bausteinen. *Autom. Tech. Prax.* **46** (10):61 – 71.

AspenTech (2008) *Aspen Plus*. Online available at http://www.aspentech.com/products/aspen-plus.cfm. Accessed June 2008.

AVEVA (2008). *AVEVA NET – website*. Online available at http://www.aveva.com/products_services_aveva_net.php. Accessed June 2008.

Awad EM, Ghaziri HM (2003) *Knowledge Management*. Prentice Hall, New Jersey.

Bañares-Alcántara R, Lababidi HMS (1995) Design support systems for process engineering. – II. KBDS: An experimental prototype. *Comput. Chem. Eng.* **19**:279–301.

Bañares-Alcántara R, Kokossis A, Aldea A, Jiménez L, Linke P (2003) A knowledge management platform to extract and process information from the Web, In: Chen B, Westerberg AW (eds.): *Process Systems Engineering 2003*. Elsevier:1262–1267.

Batres R, Naka Y, Lu ML (1999) A multidimensional design framework and its implementation in an engineering design environment. *Concur. Eng.* **7** (1):43–54.

Becker S, Nagl M, Westfechtel B (2008a) Incremental and interactive integrator tools for design product consistency. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:224–267.

Becker S, Marquardt W, Morbach J, Nagl M (2008b) Model dependencies, fine-grained relations, and integrator tools. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:612 – 620.

Bieszczad J (2000) *A Framework for the Language and Logic of Computer-Aided Phenomena-Based Process Modeling*. PhD Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology.

Biegler LT, Grossmann IE, Westerberg AW (1997) *Systematic Methods of Chemical Process Design*. Prentice-Hall.

Bilgic T, Rock D (1997) Product data management systems: State-of-the-art and the future. In: *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Sacramento, CA.

Bogusch R, Lohmann B, Marquardt W (2001) Computer-aided process modeling with ModKit. *Comput. Chem. Eng.* **25** (7/8):963–995.

Brandt SC, Schlüter M, Jarke M (2006) A process data warehouse for tracing and reuse of engineering design processes. *Int. J. Intell. Inf. Technol.* **2** (4):18–26.

Brandt SC, Fritzen O, Jarke M, List T (2008a) Goal-oriented information flow management in development processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:369–400.

Brandt SC, Morbach J, Miatidis M, Theißen M, Jarke M, Marquardt W (2008b) An ontology-based approach to knowledge management in design processes. *Comput. Chem. Eng.* **32**:320–342.

Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I, Yang A (2004) CAPE web services: the COGents way. In: Barbarosa-Póvoa, A., Matos, H. (Eds.): *European Symposium on Computer Aided Process Engineering -14*. Elsevier, Amsterdam:1021–1026.

Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F, eds. (2006b). *Extensible Markup Language (XML) 1.0 (Forth Edition)*. W3C Recommendation, 16 August 2006. Online available at http://www.w3.org/TR/xml/. Accessed September 2008.

Brickley D, Guha RV, eds. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/rdf-schema/. Accessed September 2008.

Britt H, Chen C, Mahalec V, McBrien A (2004) Modeling and simulation in 2004: an industrial perspective. In: *Proceedings of the 6th International Conference on Foundations of Computer Aided Process Design*, CACHE Publications, 55–68.

Cameron I, Hangos K, Stephanopolous G, Perkins J (2001) *Process Modeling and Model Analysis*. Academic Press.

Clark J, ed. (1999) *XSL Transformations (XSLT), Version 1.0*. W3C Recommendation, 16 November 1999. Online available at http://www.w3.org/TR/xslt. Accessed May 2008.

Crubézy M, O'Connor M, Buckeridge DL, Pincus Z, Musen MA (2005) Ontology-centered syndromic surveillance for bioterrorism. *IEEE Intell. Syst.* **20** (5):26–35.

Dassault Systemes (2008) *Industry PLM Solutions*. Online available at http://www.3ds.com/solutions/. Accessed December 2008.

De Giacomo G, Franconi E, Cuenca Grau B, Haarslev V, Kaplunova A, Kaya A, Lembo D, Lutz C, Milicic M, Möller R, Sattler U, Sertkaya B, Suntisrivaraporn B, Turhan AY, Wandelt S, Wessel M (2007) *Analysis of Test-Results on Individual Test Ontologies*. TONES project deliverable (TONES-D23). Online available at http://www.tones-project.org. Accessed June 2008.

Douglas JM (1988) *Conceptual Design of Chemical Processes*. McGraw-Hill, New York.

Eclipse (2009) *Eclipse download*. Online available at http://www.eclipse.org/ Accessed April 2009.

Eggersmann M, Hackenberg J, Marquardt W, Cameron I (2002) Applications of modeling: A case study from process design. In: Braunschweig B, Gani R (eds.): *Software Architecture and Tools for Computer Aided Process Engineering*. Elsevier Science:335–372.

EMC^2 (2008) *Documentum Family*. Online available at http://software.emc.com/products/product_family/documentum_family.htm. Accessed May 2008.

Fedai M, Drath R (2004) CAEX – ein neutrales Datenaustauschformat für Anlagendaten – Teil 1. *Autom. Tech. Prax.* **46** (2):52–56.

Gallaher MP, O'Connor AC, Dettbarn Jr JL, Gilday LT (2004) *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. Technical Report (NIST GCR 04-867), National Institute of Standards and Technology, Gaithersburg, Maryland.

Gao JX, Aziz H, Maropoulos PG, Cheung WM (2003) Application of product data management technologies for enterprise integration. *Int. J. Computer Integr. Manuf.* **16** (7-8):491–500.

Hackenberg J (2006) *Computer Support for Theory-Based Modeling of Process Systems*. Fortschritt-Berichte VDI, Reihe 3, Nr. 860, VDI-Verlag, Düsseldorf.

Halevy AY (2001) Answering queries using views: a survey. *VLDB J.* **10** (4):270–294.

HP Labs (2007) *HP Labs Semantic Web Research*. available at www.hpl.hp.com/semweb/. Accessed January 2008.

Innotec (2008) *Innotec – Product Overview*. Online available at http://www.innotec.com/produktuebersicht.html?&L=1. Accessed March 2008.

Intergraph (2008). *Intergraph: SmartPlant foundation*. Online available at http://www.intergraph.com/products/ppm/smartplant/default.aspx. Accessed March 2008.

Jarke M, List T, Köller J (2000) The challenge of process data warehousing. In: *Proceedings of the 26th International Conference on Very Large Databases*. Morgan Kaufmann:473–483.

Jarke M, List T, Weidenhaupt K (1999) A process-integrated conceptual design environment for chemical engineering. In: *Proceedings of the 18th International Conference on Conceptual Modeling*. Springer, Berlin:520–537.

Jarke M, Lenzerini M, Vassiliou Y, Vassiliadis P (2003) *Fundamentals of Data Warehouses*. Springer, Berlin.

Jensen AK (1998) *Generation of problem specific simulation models within an integrated computer aided system.* PhD Thesis, Department of Chemical Engineering, Technical University of Denmark.

Kifer M, Lausen G, Wu J (1995) Logical foundations of object-oriented and frame-based languages. *JACM* **42** (4):741–843.

Kim Y, Kang S, Lee S, Yoo S (2001) A distributed, open, intelligent product data management system. *Int. J. Computer Integr. Manuf.* **14**:224–235.

Kitamura Y, Mizoguchi R (2003) Ontology-based description of functional design knowledge and its use in a functional way server. *Expert Syst. Appl.* **24** (2):153–166.

Klein M (2002) Interpreting XML documents via an RDF schema ontology. In: *Proceedings of the 13th International Workshop on Database and Expert Systems Applications*:889–893.

Kopena J, Regli WC (2003) Functional modeling of engineering designs for the semantic web. *IEEE Data Eng. Bull.* **26** (4):55–61.

Linninger A (2000) Towards computer-aided model generation. In: *Proceedings of the JSPS International Workshop on Safety-Assured Operation and Concurrent Engineering*. Yokohama, Japan:C35–C49.

Maier A, Schnurr HP, Sure Y (2003) Ontology-based information integration in the automotive industry. In: *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Florida:897-912.

Maropoulos PG (2003) Digital enterprise technology – defining perspectives and research priorities. *Int. J. Computer Integr. Manuf.* **16** (7/8):467–478.

Marquardt W (1996) Trends in computer-aided process modeling. *Comput. Chem. Eng.* **20** (6/7):591–609.

Marquardt W, Nagl M (2004) Workflow and information centered support of design processes – the IMPROVE perspective. *Comput. Chem. Eng.* **29** (1):65–82.

McMahon C, Lowe A, Culley SJ (2004) Knowledge management in engineering design, personalisation and codification. *J. Eng. Des.* **15** (4):307–325.

Mesihovic S, Malmqvist J, Pikosz P (2004) Product data management system-based support for engineering project management. *J. Eng. Des.* **15** (4):389–403.

Miatidis M, Jarke M (2005) Integrating workflow extensions into a process-integrated environment for chemical engineering. In: *Proceedings of the 7th International Conference on Enterprise Information Systems*:255–260.

Miatidis M, Jarke M, Weidenhaupt K (2008) Using developers' experience in co-operative design processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:185–223.

Morbach J, Marquardt W (2006) Wissenssprache im Anlagenbau – die Erstellung von Branchenleistungsverzeichnissen mit Hilfe von Ontologien. In: Schenk M (ed.): *Industriearbeitskreis „Kooperation im Anlagenbau": Arbeitsbericht 03/04*. Fraunhofer IRB Verlag, Stuttgart:1–17.

Morbach J, Marquardt W (2008) Ontology-based integration and management of distributed design data. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:647–655.

Morbach J, Theißen M, Hai R, Marquardt W (2008a) An introduction to application domain modeling. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:83–92.

Morbach J, Hai R, Bayer B, Marquardt W (2008c) Document models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:111–125.

Noumonon Consulting Limited (2008) *Open Access to Intelligent Process Plant Models*. Online available at http://www.noumenon.co.uk/. Accessed April 2008.

Oberle D, Volz R, Motik B, Staab S (2004) An extensible ontology software environment. In: Staab S, Studer R (eds.): *Handbook on Ontologies*. Springer, Berlin:311–333.

Ontoprise GmbH (2009) *OntoStudio*. Online available at http://www.ontoprise.de/en/home/products/ontostudio/. Accessed April 2009.

Paton NW, Goble CA, Bechhofer S (2000) Knowledge-based information integration systems. *Inform. Software Technol.* **42** (5):299–312.

Perkins JD, Sargent RWH, Vazquez-Roman R, Cho JH (1996) Computer generation of process models. *Comput. Chem. Eng.* **20** (6):635–639.

Pohl K, Weidenhaupt K, Dömges R, Haumer P, Jarke M, Klamma R (1999) PRIME: Towards Process-Integrated Environments. *ACM Transactions on Software Engineering and Methodology* **8** (4):343–410.

Preisig HA (1995) MODELLER – An object-oriented computer-aided modeling tool. In: Biegler LT, Doherty MF (eds.): *Foundations of Computer-Aided Process Design*. AIChE:328–331.

PTC (2008) *Windchill*. Online available at http://www.ptc.com/products/windchill/. Accessed May 2008.

Raddatz M, Schlüter M, Brandt SC (2006) Identification and reuse of experience knowledge in continuous production processes. Presented at the *9th IFAC Symposium on Automated Systems Based on Human Skill and Knowledge*. Online available at http://www-i5.informatik.rwth-aachen.de/i5new/publications/pubs2006.html. Accessed May 2008.

Rector A, Drummond N, Horridge M, Rogers J, Knublauch H, Stevens R, Wang H, Wroe C (2004) OWL pizzas: practical experience of teaching OWL-DL: common errors & common patterns. In: Motta E, Shadbolt N, Stutt A, Gibbins N (eds.): *Proceedings of the European Conference on Knowledge Acquisition (EKAW)*. Springer:63–81.

Sheremetov L, Batyrshin I, Chi M, Vergara E, Rosales A (2007) Knowledge-based collaborative engineering of pipe networks in the upstream and downstream petroleum industry. In: Shen W, Yang Y, Yong J, Hawryszkiewycz I, Lin Z, Barthes JPA, Maher ML, Hao Q, Tran MH (eds.): *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007)*. IEEE:458–463.

Siemens PLM Software (2008) *Teamcenter*. Online available at http://www.ugs.com/ products/ teamcenter/. Accessed May 2008.

Stephanopoulos G, Henning G, Leone H (1990) MODEL.LA. A modeling language for process engineering. I. The formal framework. *Comput.. Chem. Eng.* **14** (8):813–846.

Szykman S, Sriram RD, Bochenek C, Racz JW, Senfaute J (2000) Design repositories: engineering design's new knowledge base. *IEEE Intell. Syst.* **15** (3):48-55.

Szykman S, Sriram RD, Regli WC (2001) The role of knowledge in next-generation product development systems. *J. Comput. Inf. Sci. Eng.* **1** (1):3–11.

Theißen M, Hai R, Marquardt W (2008a) Computer-assisted work process modeling in chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:656–666.

Theißen M, Hai R, Marquardt W (2008b) Design process modeling in chemical engineering. *J. Comput. Inf. Sci. Eng.* **8** (1), 011007 (9 pages).

Tiller MM (2001) *Introduction to Physical Modeling with Modelica*. Springer, Berlin.

Venkatasubramanian V, Zhao C, Joglekar G, Jain A, Hailemariam L, Suresh P, Akkisetty P, Morris K, Reklaitis GV (2006) Ontological informatics infrastructure for pharmaceutical product development and manufacturing. *Computers and Chemical Engineering* **30** (10/12):1482–1496.

Visser U, Stuckenschmidt H, Wache H, Vögele T (2000) Enabling technologies for interoperability. In: Visser U, Pundt H (eds.): *Workshop: Information Sharing: Methods and Applications at the 14th International Symposium of Computer Science for Environmental Protection*:35–46.

Wache H, Vögele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hübner S (2001) Ontology-based integration of information – a survey of existing approaches. In: Goméz-Pérez A, Gruninger M, Stuckenschmidt H, Uschold M (eds.): *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*. CEUR Workshop Proceedings:108–117.

Westerberg AW, Subrahmanian E, Reich Y, Konda S, the n-dim group (1997) Designing the process design process. *Comput. Chem. Eng.* **21**:1–9.

Wiesner A, Morbach J, Marquardt W (2008b) Semantic data integration for process engineering desgin data. In: *Proceedings of the 10th International Conference on Enterprise Information Systems – ICEIS 2008*:190-195.

Windream GmbH (2008) *Managing Documents by Windream*. Online available at http://www.windream.com/. Accessed May 2008.

Yang A, Marquardt W (2004) An ontology-based approach to conceptual process modeling. In: Barbarosa-Póvoa A, Matos H (eds.): *Proceedings of the European Symposium on Computer Aided Process Engineering – ESCAPE 14*. Elsevier:1159–1164.

Yang A, Morbach J, Marquardt W (2004a) From conceptualization to model generation: the roles of ontologies in process modeling. In: Floudas CA, Agrawal R (eds.): *Sixth International Conference on Foundations of Computer-Aided Process Design*. Omnipress:591–594.

Yang A, Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I (2008) A multi-agent system to facilitate component-based process modeling and design. *Comput. Chem. Eng.* **32** (10):2290–2305.

13 Conclusions

13.1 OntoCAPE in a Nutshell

OntoCAPE constitutes an ontology framework designed for multiple applications in the domain of computer-aided process engineering. It comprises a top-level ontology, a domain ontology, several application ontologies as well as a generic meta-ontology that provides best-practice design patterns for various modeling problems. The individual sub-ontologies of OntoCAPE can be easily extended, customized, or integrated with other ontologies.

OntoCAPE has the objectives of being both usable and reusable. These two objectives are in a natural conflict: Usability implies specialization to match the requirements of a particular task, whereas reusability requires generality in order to facilitate an application in different contexts. Consequently, it is difficult to simultaneously achieve a high degree of usability and reusability at the same time. A reasonable compromise can only be reached partially, and it requires considerable time and effort since the ontology needs to be iteratively redesigned and tested in different applications. Contrary to numerous pseudo ontologies, which are content to support only one single application, OntoCAPE nevertheless takes up the challenge to realize a reasonable trade-off between usability and reusability.

OntoCAPE is published in form of two complementary parts, a formal specification and an informal specification. The formal specification is given by means of the modeling language OWL DL. Its current release consists of 62 OWL files comprising about 500 classes, 200 relations, and 40,000 individuals in total. The ontology terms are formally defined through a large number of axioms. Therefore, OntoCAPE can be characterized as a heavy-weight ontology. The informal specification of OntoCAPE comprises about 500 pages of natural language documentation. It serves the double purpose of (i) a user manual and (ii) a reference guide, in that it (i) explains the ontology and its handling to common users and (ii) supports applications developers in refining, extending, or changing the ontology to their particular needs.

OntoCAPE is hierarchically structured by layers, which subdivide the ontology into different levels of abstraction and thus separate general knowledge from knowledge about particular domains and even applications. The topmost Meta Layer, is the most abstract one. It holds the Meta Model, which guides ontology development and enforces design consistency when changing or extending the ontology. Next, the Upper Layer of OntoCAPE defines key concepts such as system, physical quantity, or backdrop, and introduces the principles of general systems theory according to which the ontology is organized. On the subjacent Conceptual Layer, a conceptual model of the CAPE domain is established, which covers such different areas as unit operations, equipment and machinery, materials and their thermophysical properties, chemical process behavior, and mathematical modeling. The two bottommost layers refine the conceptual model by adding classes and relations required for the practical application of the ontology: The Application-Oriented Layer generically extends the ontology towards certain application areas, whereas the Application-Specific Layer provides specialized classes and relations for concrete applications.

Each layer is subdivided into a number of modules. The boundaries of a module are chosen such that the module can be designed, adapted, and reused independently from the other parts of the ontology to the extent possible. Different variants of an ontology module may evolve, allowing for the coexistence of alternative knowledge representations of the same or overlapping chunks of knowledge. Modules that address closely related topics are grouped into a common partial model. Unlike modules, partial models may be nested and may stretch across several layers. Their boundaries reflect the "natural" categorization of the domain, thus providing a stable frame of orientation for the organization of the modules. Overall, the structuring of the ontology into layers, modules, and partial models follows two principal objectives, namely to facilitate the ontology's extensibility and long-term maintenance, and to enable its customization and reuse in different application contexts.

With regard to related work, OntoCAPE has incorporated certain aspects from engineering ontologies of related domains. In the chemical engineering domain, OntoCAPE is unique in the sense that it is currently the only (re)usable ontology available to support CAPE software development.

13.2 Design Rationale

Numerous recommendations for ontology design are given in the literature. They have been condensed to six major principles, which have guided the design of OntoCAPE: coherence, conciseness, intelligibility, adaptability, minimal ontological commitment, and efficiency. It has been demonstrated how OntoCAPE has put these principles into practice and, in the course of this process, significantly gained in quality. Since some of the principles are incompatible, a suitable balance between the conflicting principles had to be found. The finally realized design is a reasonable compromise between the two major objectives of usability and reusability.

However, it is difficult to quantify the degree of quality due to the absence of generally accepted key measures assessing an agreed set of quality indicators. Thus, we decided to compensate the lack of formal measures by putting OntoCAPE to the test in a number of prototypical software applications. Even if formal measures for quality indicators were available, the degree of (re)usability can be proven ultimately only in an inductive experience-based manner by testing OntoCAPE in a (preferably large) number of different software applications.

Correspondingly, OntoCAPE has been field-tested in a number of software projects covering the chemical process modeling and simulation as well as chemical process design. Through these tests, and through the iterated application of the ontology within these projects, the quality of OntoCAPE has been systematically improved. Nevertheless, we consider it highly desirable to investigate suitable formal measures and sound procedures for checking the quality of an ontology design during ontology development prior to its use in a software project.

13.3 Ontology Evolution by a Continuous Improvement Process

Compliance with the above design principles is hard to achieve in a top-down manner. One reason is the generality, and to some degree also the vagueness, of the design principles. There are only few clear-cut and well-understood rules and guidelines to implement these design principles in a given ontology design project on the concrete level. In other words, we are still lacking validated procedures for the translation of the general design principles into concrete design and modeling decisions both on the architectural as well as on the elementary level of concepts and axioms. A second reason is the inherent complexity of the ontology design task and the lack of measurable indicators to assess quality and the degree of requirements fulfillment.

Consequently, an ontology has to be continuously improved in a systematic process. Such a strategy is comparable to the well-known and successfully applied Kaizen principle, or continuous improvement process, in management (Imai 1997). According to the Kaizen principle, reflection of the current business process constitutes the foundation for (i) the identification of suboptimal process chunks and (ii) the improvement of the business process by a series of incremental steps in an evolutionary manner, thus avoiding quantum leaps with an unpredictable outcome.

Like in business process engineering, we believe that a continuous improvement process is inevitable to achieve a good usability-reusability trade-off and thus an ontology of high quality. Reflection, as of Kaizen, is implemented in the context of ontology engineering by extensive field-testing of the ontology in a (preferably large) number of different software applications. These field tests reveal the improvement potential which is then gradually and continuously realized.

Our research and development work followed such a continuous improvement approach. The field-testing of earlier versions of OntoCAPE revealed errors, inconsistencies and opportunities for improvement. The remediation of the discovered flaws eventually led to the creation of OntoCAPE 2.0. As of this version, the ontology passed all our tests. On the one hand, our ontology proved to be applicable in different contexts, which is an indicator of reusability. On the other hand, the effort required for adapting OntoCAPE to a concrete application turned out to be moderate, which proves the usability of the ontology.

So far, OntoCAPE has not yet been tested in a real-world industrial application, but only in software prototypes and against academic usage scenarios. To close this gap, OntoCAPE is currently being put to the test in a software tool for the integration and consolidation of engineering information, which is used and produced by an interdisciplinary design team working in different organizational units during different tasks of the design lifecycle of a chemical plant. This software tool is developed in cooperation with partners from the chemical and the software industries. Extensive testing by industrial practitioners will establish whether the ontology complies with the requirements of industrial practice. The project is still in progress, but judging from our preliminary results, the ontology seems to fulfill all the requirements for industrial use. Such field-testing in an industrial context is just another and obviously essential phase of a continuous improvement process.

Yet another phase of ontology evolution requires the application and testing of the ontology by people with diverse disciplinary backgrounds and in other types of software projects, preferably in fields of application not considered so far. With respect to the latter, the areas of e-procurement and e-learning are particularly promising since they require well-structured knowledge representations, which are consensual not only across disciplinary, but in particular across institutional boundaries.

The history of OntoCAPE exemplarily shows that ontologies are dynamic information systems, which evolve and change according to the prevailing experience, context and requirements. Ontology evolution is unavoidable. Therefore, it is expected to continue in the future, in particular due to the following drivers: On the one hand, the expected advancement of ontology languages – in combination with improved algorithms resulting in reasoners with high performance for large-scale ontologies – will enable the use of more rules of more complex nature as well as the use of advanced DL constructs in the formal specification, thus allowing to further increase the level of axiomatization. On the other hand, the ontology is expected to change in scope and conceptualization in order to adopt new insights into ontology design, to extend the coverage of domain knowledge and to facilitate an increasing number and type of applications.

13.4 From Product to Process

Design processes in the chemical process industries, like those in many other businesses, are of a cooperative nature and involve different departments in one or more enterprises, typically at geographically distributed sites. They use and produce a vast amount of information organized in a multitude of documents with many interdependencies and overlaps stored in very different electronic formats. These documents are often called the *products* of the work *process*.

Various software tools are used to support a project team. Some of them are of a domain-specific nature (i.e., chemical process simulators, CAD or CAE systems, etc.), others are of generic type and thus independent of the requirements of chemical engineering (i.e., word processors, project management systems, etc.). An efficient support of chemical process design processes requires adequate IT support across the lifecycle of the project. Existing software tools for dedicated tasks in the project lifecycle have to be enhanced and linked to evolve into a coherent design support system, which not only offers user-interface, data, and control integration, but also integrates the distributed, collaborative, and concurrent design process carried out by interdisciplinary teams in different organizations.

To date, no satisfactory solution is available in industry. Therefore, an enormous potential exists to increase productivity of design teams, or, to put it more precisely, to reduce cost and to improve quality at the same time. Leveraging such potential constitutes a tremendous economical opportunity in particular for enterprises in high wage countries such as Europe, Japan, the US, and Canada.

Industrial work processes are complex and consequently difficult to plan, document, improve, and reuse. A fundamental understanding of these work processes is considered to be a prerequisite for their reengineering and for the development of effective support systems based on information technology. To date, the focus of semantic technologies has been mainly on the representation, integration, and retrieval of information about the *results* of work processes – sometimes called *products* – such as the specification of a chemical plant by means of documents like flow sheets and equipment lists in case of design processes. In fact, many tasks in a design process require an integrated view on work processes and their products. Examples include the monitoring of the progress in a concrete design project, the detection of inconsistencies in the design data, or even the uncovering of incomplete design tasks. Such an integration of product and process representation is crucial if we aim at software tools that reach beyond traditional data, control, and user interface integration, but provide additional functionality to support the design process more effectively.

Considerable research activity has been devoted to the modeling of work processes and to the design of supporting software systems, in particular in software engineering (e.g. Jacobson et al. 2003; ISO 12207 2008), business process (re-)engineering (e.g. Davenport 1993; Hammer and Champy 1993) and to a lesser extent in the different engineering sciences (e.g., Hubert and Houten 1999; Ullman 2002). Ontologies have also been used for the formal representation of and for the reasoning on work processes even in an engineering context (cf. Kitamura et al. 2006; Batres et al. 2000; Fuchino et al. 2005; Eggersmann et al. 2003b). A comprehensive review of this literature is obviously beyond the scope of this concluding chapter. However, we want to point to the research on ontologies for work process modeling in our group in this area, because the available and still evolving results will be integrated with OntoCAPE in the future.

As a first attempt towards the capturing and representation of work processes in chemical process design a graphical notation, C3 (Killich et al. 1999; Eggersmann et al. 2008), has been developed in the IMPROVE project (Nagl and Marquardt 2008). C3 allows a coarse-grained representation of work processes including the various *actions* (i.e., the steps of a complex work process), the *actors* (i.e., humans or software) and their *roles* in the work process, the *information* used and produced during the actions, and the *resources* required (i.e., software tools, lab equipment, etc.). The deliberately simple syntax and semantics of the notation support the participative modeling of collaborative work processes, i.e., C3 models can be created by people involved in a particular work process with little or even without any assistance of modeling experts. An iterative procedure for the creation of C3 models and the application of these models to improve industrial work processes has been established based on best-practices. An overview on the modeling procedure and its application to a number of industrial case studies has been presented by Theißen et al. (2008b; 2008c).

While the C3 notation is very useful for participative work process modeling, it lacks detail and the degree of formality which is required for an integrated representation of processes and products, in particular in the context of information system design and construction. Therefore, the development of formal work process models using semantic modeling and ontological technologies has been identified as a logical next step. Some first results are reported by Eggersmann et al. (2008) and Theißen and Marquardt (2008), who present an ontology for work processes extending and refining C3 and an ontology for design decisions, respectively.

In an ongoing research project (Theißen et al. 2008a; Hai et al. 2009), we aim at an extension of OntoCAPE to also provide capabilities for work process modeling. To this end, two steps are required. First, a work process ontology is developed, which does not only address the particular needs of design processes, but rather covers work processes in more general terms. This work process ontology can be integrated into the Upper Layer of OntoCAPE in the future. The second step is the refinement of the work process ontology on the Conceptual and Applications Layers of OntoCAPE to account for the characteristics of different types of work processes. The extension of OntoCAPE towards the representation of work processes of different types, including not only process design but also product design and operational processes, is a further benchmark test for the reusability of the core concepts and architecture of the ontology.

13.5 Semantic Technologies in Engineering – Dream or Reality?

Our research on the use of semantic technologies in chemical engineering has clearly revealed the enormous potential of ontology-based information modeling and of the reasoning capabilities of current semantic software technologies. A properly chosen architecture of the ontology empowers the chemical engineer to extend and modify the ontology by means of high-level modeling tools without the assistance of ontology experts. The direct access to the domain knowledge facilitates maintenance and extension of the information model and the knowledge-bases. The reasoning capabilities of semantic tools facilitate the checking of the logical consistency of the information model and the implemented knowledge. We have not only validated the concepts on academic "toy" problems, but have also implemented prototypical software systems to demonstrate advanced design support functionality in an academic environment. We are furthermore validating the methodology – OntoCAPE and the underlying semantic technologies – in ongoing industrial projects.

There are, however, also some drawbacks. In particular, the development of an extensible and widely usable ontology is by no means straightforward. Though we believe that OntoCAPE constitutes a very good foundation for a generic and widely usable ontology for chemical engineering applications, we still see a lot of room for improvement and for the extension of the ontology. In fact, we believe that an ontology is never ready for use. It cannot be complete since it is impossible to cover all the concepts in a given domain in a comprehensive manner. Even it would be complete in this sense, it would not be readily usable, because there will always be the need for adaptations and refinements to match the requirements of an envisioned application. In our opinion, this can be compared to libraries of mathematical models provided by all state-of-the-art simulation tools: These libraries provide simulation models for many standard devices. The available models can be further specialized and parameterized by a user at very little effort. Often, new models have to be created, either by deriving them from similar models available in the library or by developing them from scratch. Such a library extension may serve the purpose of extending the coverage of a certain domain in the library or of tailoring the simulation models towards the requirements of a certain model-based application. Such a library extension can be conveniently achieved only if the library is built on a sound theoretical basis.

Since OntoCAPE is primarily based on the concepts of systems engineering, we believe that its application is not restricted to the domain of chemical engineering, but it is applicable to other engineering domains, as well. Particularly the generic parts with their emphasis on reusability are conceptualized in a way to support various engineering domains.

Furthermore, the semantic technologies, in particular the ontology editors and reasoning tools, are still under development. Performance is an issue for the latter, while usability by application-domain rather than knowledge-engineering experts is the key issue for the former. The most important bottleneck at the moment is the lack of applications, which demonstrate the capabilities of semantic technologies and their advantages compared to the more established software technologies that are nowadays used for the design and construction of industrial-strength information systems. In addition, there is a lack of detailed application models, which can be used for a variety of tasks during the lifecycle of a design project or even of the plant itself.

Due to the immense workload required to come up with useful and comprehensive ontologies for chemical engineering applications, it is highly desirable that academics and industry join forces in order to develop, maintain, and gradually extend a process engineering ontology for process engineering. The open-source ontology OntoCAPE is definitely an excellent starting point for such an undertaking.

13.6 References

Batres R, West M, Leal D, Price D, Masaki K, Shimada Y, Fuchino T, Naka Y (2007) An upper ontology based on ISO 15926. *Comput. Chem. Eng.* **31** (5/6):519–534.

Davenport TH (1993) *Process Innovation.* Harvard Business School, Boston.

Eggersmann M, Gonnet S, Henning GP, Krobb C, Leone HP, Marquardt W (2003b): Modeling and understanding different types of process design activities. *Latin Am. Appl. Res.* **33**:167-175

Eggersmann M, Hai R, Kausch B, Luczak H, Marquardt W, Schlick C, Schneider N, Schneider R, Theißen M (2008) Work process models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:126–152.

Fuchino T, Takamura T, Batres R (2005) Development of engineering ontology on the basis of IDEF0 activity model. In: Khosla R, Howlett RJ, Jain LC (eds.): *Knowledge-Based Intelligent Information and Engineering Systems, 9th International Conference (KES 2005).* Springer, Berlin:162–168.

Hai R, Theißen M, Marquardt W (2009) An integrated ontology for operational processes. In: Jezowski J, Thullie J (eds.): *Proceedings of the 19th European Symposium on Computer-Aided Process Engineering*. Elsevier:1087–1091.

Hammer M, Champy J A (1993) *Reengineering the Corporation: a Manifesto for Business Revolution.* HarperCollins, NewYork.

Hubert H, van Houten F, eds. (1999) *Integration of Process Knowledge into Design Support Systems.* Springer.

Imai M (1997) *Gemba Kaizen: A Commonsense, Low-Cost Approach to Management*. McGraw-Hill, New York.

ISO/IEC 12207 (2008) *Systems and Software Engineering – Software life cycle processes.*

Jacobson I, Booch G, Rumbaugh J (2003) *The Unified Software Development Process: UML.* Addison-Wesley.

Killich S, Luczak H, Schlick C, Weißenbach M, Wiedenmaier S, Ziegler J (1999) Task modelling for cooperative work. *Behaviour and Information Technology* **18** (5):325–338.

Kitamura Y, Koji Y, Mizoguchi R (2006) An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology* **1** (3-4):237–262.

Nagl M, Marquardt W, eds. (2008) *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support*. Springer, Berlin.

Theißen M, Marquardt W (2008) Decision models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:153–168.

Theißen M, Hai R, Marquardt W (2008a) Computer-assisted work process modeling in chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:656–666.

Theißen M, Hai R, Marquardt W (2008b) Design process modeling in chemical engineering. *J. Comput. Inf. Sci. Eng.* **8** (1), 011007 (9 pages).

Theißen M, Hai R, Morbach J, Schneider R, Marquardt W (2008c) Scenario-based analysis of industrial work processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:433–450.

Ullman D (2002) Toward the ideal mechanical engineering design support system. *Research in Engineering Design* **13** (2), 55– 64.

14 Bibliography

Abecker A, Bernardi A, Hinkelmann K, Kühn O, Sintek M (1998) Toward a technology for organizational memories. *IEEE Intell. Syst.* **13** (3):40–48.

Ackoff RL (1989) From data to wisdom. *J. Appl. Syst. Anal.* **16**:3–9.

Aitken S (1998) Extending the HPKB-upper-level ontology: experiences and observations. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:11–15.

Alberts LK (1994) YMIR: a sharable ontology for the formal representation of engineering design knowledge. In: Gero JS, Tyugu E (eds.): *Formal Design Methods for CAD*. Elsevier, New York:3–32.

Amin MA, Morbach J (2007) *XML to OWL Converter*. Technical Report (LPT-2007-02), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Amin MA, Morbach J (2008) *DAML+OIL to OWL converter*. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id= 523. Accessed January 2008.

Andresen T (1999) The macroeconomy as a network of money-flow transfer functions. *Model. Ident. Contr.* **19** (4):207–223.

Anhäuser F, Richert H, Temmen H (2004) Degussa PlantXML – integrierter Planungsprozess mit flexiblen Bausteinen. *Autom. Tech. Prax.* **46** (10):61 – 71.

Arpírez JC, Gómez-Pérez A, Lozano A, Pinto HS (1998) $(\text{ONTO})^2$Agent: an ontology-based WWW broker to select ontologies. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:16–24.

AspenTech (2008) *Aspen Plus*. Online available at http://www.aspentech.com/products/aspen-plus.cfm. Accessed June 2008.

Atkinson C, Kühne T (2002) The role of metamodeling in MDA. In: *Proceedings of the Workshop in Software Model Engineering (in conjunction with UML'02, Dresden, Germany).* Online available at http://www.meta model.com/wisme-2002/papers/atkinson.pdf. Accessed January 2008.

AVEVA (2008). *AVEVA NET – website*. Online available at http://www.aveva.com/products_services_aveva_net.php. Accessed June 2008.

Awad EM, Ghaziri HM (2003) *Knowledge Management*. Prentice Hall, New Jersey.

Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (2003) *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, Cambridge.

Bañares-Alcántara R, Lababidi HMS (1995) Design support systems for process engineering. – II. KBDS: An experimental prototype. *Comput. Chem. Eng.* **19**:279–301.

Bañares-Alcántara R, Kokossis A, Aldea A, Jiménez L, Linke P (2003) A knowledge management platform to extract and process information from the Web. In: Chen B, Westerberg AW (eds.): *Process Systems Engineering 2003*. Elsevier:1262–1267.

Bard JBL, Rhee SY (2004) Ontologies in biology: design, applications and future challenges. *Nat. Rev. Genet.* **5**:213–222.

Barkmeyer EJ, Feeney AB, Denno P, Flater DW, Libes DE, Steves MP, Wallace EK (2003) *Concepts for Automating Systems Integration*. Technical Report (NISTIR 6928), National Institute of Standards and Technology (NIST), Gaithersburg, MD.

Batres R, Naka Y (2000) Process plant ontologies based on a multi-dimensional framework. In: Malone MF, Trainham JA, Carnahan B (eds.): *Fifth International Conference on Foundations of Computer-Aided Process Design*. AIChE:433–437.

Batres R, Naka Y, Lu ML (1999) A multidimensional design framework and its implementation in an engineering design environment. *Concur. Eng.* **7** (1):43–54.

Batres R, Aoyama A, Naka Y (2002) A life-cycle approach for model reuse and exchange. *Comput. Chem. Eng.* **26** (4/5):487–498.

Batres R, West M, Leal D, Price D, Masaki K, Shimada Y, Fuchino T, Naka Y (2007) An upper ontology based on ISO 15926. *Comput. Chem. Eng.* **31** (5/6):519–534.

Baumeister M (2000) *Ein Objektmodell zur Repräasentation und Wiederverwendung verfahrenstechnischer Prozeßmodelle*. PhD thesis (LPT-diss-2000-12), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L= 0&Nummer=LPT-diss-2000-02.

Baumeister M, Marquardt W (1998) *The chemical engineering data model VeDa, part 1: VDDL – The Language Definition*. Technical Report (LPT-1998-01), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Baxter JE, Juster NP, de Pennington A (1994) A functional data model for assemblies used to verify product design specifications. *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* **208** (B4):235-244.

Bayer B (2003) *Conceptual Information Modeling for Computer Aided Support of Chemical Process Design*. Fortschritt-Berichte VDI: Reihe 3, Nr. 787. VDI-Verlag, Düsseldorf.

Bayer B, Marquardt W (2003) A Comparison of Data Models in Chemical Engineering. *Concur. Eng.* **11** (2):129-138.

Bayer B, Marquardt W (2004) Towards integrated information models for data and documents. *Comput. Chem. Eng.* **28** (8):1249–1266.

Bayer B, Marquardt W (2009) A Conceptual Information Model for the Chemical Process Design Lifecycle. In : Jarke M, Jeusfeld M, Mylopoulos J (eds.): *Meta Modeling and Method Engineering*, MIT:357-381.

Bayer B, Krobb C, Marquardt W (2001) *A Data Model for Design Data in Chemical Engineering – Information Models*. Technical Report LPT–2001–15. Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bayer B, Schneider R, Marquardt W (2000) Integration of data models for process design – First steps and experiences. *Comput. Chem. Eng.* **24**, 599-605.

Bechhofer S, Horrocks I, Goble C, Stevens R (2001) OilEd: a reason-able ontology editor for the semantic web. In: Baader F, Brewka G, Eiter T (eds.): *KI 2001: Advances in Artificial Intelligence*. Springer, Berlin:396–408.

Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness D, Patel-Schneider L, Stein LA (2004) *OWL Web Ontology Language Reference*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-ref/. Accessed September 2007.

Becker S, Nagl M, Westfechtel B (2008a) Incremental and interactive integrator tools for design product consistency. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:224–267.

Becker S, Marquardt W, Morbach J, Nagl M (2008b) Model dependencies, fine-grained relations, and integrator tools. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:612–620.

Bernaras A, Laresgoiti I, Corera J (1996) Building and reusing ontologies for electrical network applications. In: Wahlster W (ed.): *ECAI 1996 – Proceedings of the 12th European Conference on Artificial Intelligence*. John Wiley, New York:298–302.

Bertalanffy L (1968) *General System Theory: Foundations, Development, Applications*. Braziller, New York.

Bieszczad, J (2000) *A Framework for the Language and Logic of Computer-Aided Phenomena-Based Process Modeling*. PhD Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology.

Biegler LT, Grossmann IE, Westerberg AW (1997) *Systematic Methods of Chemical Process Design*. Prentice-Hall.

Bilgic T, Rock D (1997) Product data management systems: State-of-the-art and the future. In: *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Sacramento, CA.

BIPM (2006) *The International System of Units (SI), 8th edition*. SI brochure, published by the International Committee for Weights and Measures (Bureau International des Poids et Measures, BIPM). Online available at http://www.bipm.fr/en/si/si_brochure/. Accessed September 2007.

BIPM (2007) *BIPM: Bureau International des Poids et Mesures*. Website, available at www.bipm.org. Accessed October 2007.

Bird RB, Stewart WE, Lightfoot EN (2001) *Transport Phenomena.* John Wiley, New York.

Biron PV, Permanente K, Malhotra A (2004) *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. Online available at http://www.w3.org/TR/xmlschema-2/. Accessed January 2007.

Black PE, ed. (2004) Data structure. In: *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology. Online available at http://www.itl.nist.gov/div897/sqg/dads/. Accessed January 2009.

Blitz D (1992) Emergent Evolution, Qualitative Novelty and the Kinds of Reality. Springer.

Bodenreider O (2001) *Medical Ontology Research*. Technical Report, Lister Hill National Center for Biomedical Communications, U.S. National Library of Medicine. Online available at http://mor. nlm.nih.gov:8000/pubs/pdf/2001-MOR-BoSC.pdf. Accessed February 2008.

Bogusch R (2001) *A Software Environment for Computer-Aided Modeling of Chemical Processes*. Fortschritt-Berichte VDI: Reihe 3, Nr. 705, VDI-Verlag, Düsseldorf.

Bogusch R, Marquardt W (1998) *The Chemical Engineering Data Model VeDa. Part 4: Behavioral Modeling Objects*. Technical Report (LPT-1998-04), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bogusch R, Lohmann B, Marquardt W (2001) Computer-aided process modeling with ModKit. *Comput. Chem. Eng.* **25** (7/8):963–995.

Borst P, Akkermans JM, Top JL (1997) Engineering ontologies. *Int. J. Hum Comput Stud.* **46**:365–406.

Borst P, Akkermans JM, Pos A, Top JL (1995) The PhySys ontology for physical systems. In: Bredeweg (ed.): *Proceedings of the 9th International Workshop on Qualitative Reasoning*. University of Amsterdam:11–21.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Brandt SC, Schlüter M, Jarke M (2006) A process data warehouse for tracing and reuse of engineering design processes. *Int. J. Intell. Inf. Technol.* **2** (4):18–26.

Brandt SC, Fritzen O, Jarke M, List T (2008a) Goal-oriented information flow management in development processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:369–400.

Brandt SC, Morbach J, Miatidis M, Theißen M, Jarke M, Marquardt W (2008b) An ontology-based approach to knowledge management in design processes. *Comput. Chem. Eng.* **32**:320–342.

Braunschweig B, Gani R, eds. (2002) *Software Architecture and Tools for Computer Aided Process Engineering*. Elsevier Science B.V.

Braunschweig B, Fraga ES, Guessoum Z, Paen D, Piñol D, Yang A (2002) COGents: cognitive middleware agents to support e-CAPE. In: Stanford-Smith B, Chiozza E, Edin M (eds.): *Challenges and Achievements in E–business and E–work*. IOS Press:1182–1189.

Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I, Yang A (2004) CAPE web services: the COGents way. In: Barbarosa-Póvoa A, Matos H (eds.): *European Symposium on Computer Aided Process Engineering -14*. Elsevier, Amsterdam:1021–1026.

Bray T, Hollander D, Layman A, Tobin R, eds. (2006a) *Namespaces in XML 1.0 (Second Edition).* W3C Recommendation, 16 August 2006. Online available at http://www.w3.org/TR/xml-names. Accessed September 2008.

Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F, eds. (2006b). *Extensible Markup Language (XML) 1.0 (Forth Edition).* W3C Recommendation, 16 August 2006. Online available at http://www.w3.org/TR/xml/. Accessed September 2008.

Brickley D, Guha RV, eds. (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/rdf-schema/. Accessed September 2008.

Britt H, Chen C, Mahalec V, McBrien A (2004) Modeling and simulation in 2004: an industrial perspective. In: *Proceedings of the 6th International Conference on Foundations of Computer Aided Process Design*, CACHE Publications, 55–68.

Bunge M (1979) *Treatise on Basic Philosophy, Volume 4. Ontology II: A World of Systems*. Reidel, Dordrecht.

Bylander T, Chandrasekaran B (1988) Generic tasks in knowledge-based reasoning: the right level of abstraction for knowledge acquisition. In: Gaines B, Boose J (eds.): *Knowledge Acquisition for Knowledge-Based Systems*. Academic Press, London:65–77.

Cameron I, Hangos K, Stephanopolous G, Perkins J (2001) *Process Modeling and Model Analysis*. Academic Press.

Carlisle D, Ion P, Miner R, Poppelier N, eds. (2003) *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*. W3C Recommendation, online available at http://www.w3.org/TR/MathML2/. Accessed September 2007.

CAS (2007) CAS Registry Overview. Website, online available at http://www.cas.org/EO/regsys.html. Accessed February 2007.

Casati R, Varzi A (1999) *Parts and Places: The Structures of Spatial Representation*. MIT Press.

Cellier FE, Kofman E (2006) *Continuous System Simulation*. Springer, Berlin.

Chandrasekaran B (1994) Functional representation and causal processes. In: Yovits MC (ed.): *Advances in Computers*. Academic Press, New York.

Chandrasekaran B, Johnson TR (1993) Generic tasks and task structures: history, critique and new directions. In: David JM, Krivine JP, Simmons R (eds.): *Second Generation Expert Systems*. Springer, New York:232–272.

Chandrasekaran B, Josephson JR (2000) Function in device representation. *J. Eng. Comput.* **16**(3/4):162-177.

Chandrasekaran B, Josephson JR, Benjamins VR (1999) What are ontologies, and why do we need them? *IEEE Intell. Syst.* **14** (1):20–26.

Chen PP (1976) The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems* **1** (1):9–36.

Chertov AG (1997) Units of physical measure. In: Grigoriev IS, Meilikhov EZ (eds.): *Handbook of Physical Quantities*, CRC Press.

Clark J, ed. (1999) *XSL Transformations (XSLT), Version 1.0*. W3C Recommendation, 16 November 1999. Online available at http://www.w3.org/TR/xslt. Accessed May 2008.

Clark P, Thompson J, Porter B (2000) Knowledge patterns. In: Cohn A, Giunchiglia F, Selman B (eds.): *KR-2000: Proceedings of the Conference on Knowledge Representation and Reasoning*. Morgan Kaufmann:591–600.

Cohen WW, Borgida A, Hirsh H (1992) Computing least common subsumers in description logics. In: Swartout W (ed.): *Proceedings of the 10th National Conference on Artificial Intelligence*. MIT Press:754–760.

Connolly D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2001) *DAML+OIL reference description*. W3C Note, 18 December 2001. Online available at http://www.w3.org/TR/daml+oilreference. Accessed January 2008.

Crubézy M, O'Connor M, Buckeridge DL, Pincus Z, Musen MA (2005) Ontology-centered syndromic surveillance for bioterrorism. *IEEE Intell. Syst.* **20** (5):26–35.

Daintith J (2005) *Oxford Dictionary of Physics*. Oxford University Press.

Davenport TH (1993) *Process Innovation*. Harvard Business School, Boston.

Davis R, Shrobe H, Szolovits P (1993) What is a knowledge representation? *AI Mag.* **14** (1):17–33.

Dassault Systemes (2008) *Industry PLM Solutions*. Online available at http://www.3ds.com/solutions/. Accessed December 2008.

De Giacomo G, Franconi E, Cuenca Grau B, Haarslev V, Kaplunova A, Kaya A, Lembo D, Lutz C, Milicic M, Möller R, Sattler U, Sertkaya B, Suntisrivaraporn B, Turhan AY, Wandelt S, Wessel M (2007) *Analysis of Test-Results on Individual Test Ontologies*. TONES project deliverable (TONES-D23). Online available at http://www.tones-project.org. Accessed June 2008.

Dietz A (1995) Yet another representation of molecular structure? *J. Chem. Inf. Comput. Sci.* **35**(5):787-802.

Doerr M, Hunter J, Lagoze C (2003) Towards a core ontology for information integration. *J. Digit. Inf.* **4** (1), Article No. 169.

Douglas JM (1988) *Conceptual Design of Chemical Processes*. McGraw-Hill, New York.

Drewitz W, Szczepanski R, Pinõl D, Banks P, Halloran M, van Baten J, Pons M, eds. (2006) Thermodynamic and physical properties, version 1.1. CAPE-OPEN Interface Standards Specification. Online available at http://www.colan.org/index.html. Accessed March 2007.

EBI – European Bioinformatics Institute (2008) *Chemical Entities of Biological Interest (ChEBI)*. Online available at http://www.ebi.ac.uk/chebi/. Accessed June 2008.

EClass (2009) *eCl@ss*. Online available at: http://www.eclass.de.

Eclipse (2009) *Eclipse download*. Online available at http://www.eclipse.org/ Accessed April 2009.

Eggersmann M, Hackenberg J, Marquardt W, Cameron I (2002) Applications of modeling: A case study from process design. In: Braunschweig B, Gani R (eds.): *Software Architecture and Tools for Computer Aided Process Engineering*. Elsevier Science:335–372.

Eggersmann M, Bayer B, Jarke M, Marquardt W, Schneider R (2003a) Prozess- und Produktmodelle für die Verfahrenstechnik. In: Westfechtel B, Nagl M (eds.): *Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*. Wiley-VCH, Weinheim:75–90.

Eggersmann M, Gonnet S, Henning GP, Krobb C, Leone HP, Marquardt W (2003b): Modeling and understanding different types of process design activities. *Latin Am. Appl. Res.* **33**:167-175

Eggersmann M, Hai R, Kausch B, Luczak H, Marquardt W, Schlick C, Schneider N, Schneider R, Theißen M (2008) Work process models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:126–152.

EMC^2 (2008) *Documentum Family*. Online available at http://software.emc.com/products/product_family/documentum_family.htm. Accessed May 2008.

Encyclopedia Britannica (2009) *Unit process*. Online available at: http://www.britannica.com/EBchecked/topic/615307/unit-process.

Fedai M, Drath R (2004) CAEX – ein neutrales Datenaustauschformat für Anlagendaten – Teil 1. *Autom. Tech. Prax.* **46** (2):52–56.

Fensel D, Schönegge A, Groenboom R, Wielinga BJ (1996) Specification and verification of knowledge-based systems. In: Gaines BR, Musen MA (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Ferstl OK, Sinz EJ (2001) *Grundlagen der Wirtschaftsinformatik,* Bd. 1. Oldenbourg, München.

Föllinger O (1982) *Einführung in die Zustandsbeschreibung dynamischer Systeme*. Oldenbourg, München.

Föllinger O (1992) *Regelungstechnik – Einführung in die Methoden und ihre Anwendung*. Hüthig, Heidelberg.

Fowler M (1997) *UML Distilled – Applying the Standard Object Modeling Language*. Addison-Wesley.

Fox MS, Grüninger M (1998) Enterprise modeling. *AI Mag.* **19** (3):109–121.

Fridman-Noy N, Hafner CD (1997) The state of the art in ontology design – a survey and comparative review. *AI Mag.* **18** (3):53–74.

Froment GF, Bischoff KB (1990) *Chemical Reactor Analysis and Design.* John Wiley, New York.

Früh KF (ed.) (2000) *Handbuch der Prozessautomatisierung*. Oldenbourg, München.

Fuchino T, Takamura T, Batres R (2005) Development of engineering ontology on the basis of IDEF0 activity model. In: Khosla R, Howlett RJ, Jain LC (eds.): *Knowledge-Based Intelligent Information and Engineering Systems, 9th International Conference (KES 2005).* Springer, Berlin:162–168.

Gallaher MP, O'Connor AC, Dettbarn Jr JL, Gilday LT (2004) *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. Technical Report (NIST GCR 04-867), National Institute of Standards and Technology, Gaithersburg, Maryland.

Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gao JX, Aziz H, Maropoulos PG, Cheung WM (2003) Application of product data management technologies for enterprise integration. *Int. J. Computer Integr. Manuf.* **16** (7-8):491–500.

Gear CW, Petzold L (1984) ODE methods for the solution of differential/algebraic systems. *Trans. Society Computer Simulation* **1**:27–31.

Genesereth MR, Fikes RE et al. (1992) *Knowledge Interchange Format, Version 3.0 Reference Manual.* Technical Report (Logic-92-1), Stanford University Logic Group. Online available at http://citeseer.ist.psu.edu/genesereth92knowledge.html. Accessed October 2007.

Gensym (2008) *Business Rule Management, Business Rules, BPM Software*. Online available at http://www.gensym.com/. Accessed March 2008.

GeoNames (2007) *GeoNames Ontology*. Online available at http://www.geonames.org/ontology/. Accessed October 2007.

Gigch JP (1991) *System Design Modeling and Metamodeling*. Springer, New York.

Gilles ED (1998) Network theory for chemical processes. *Chem. Eng. Technol.* **21** (8):121–132.

GNU (2006) *CVS – Open Source Version Control*. Online available at http://www.nongnu.org/cvs/. Accessed October 2007.

GNU Project (2007) *The GNU General Public Licence*. Online available at http://www.gnu.org/copyleft/gpl.html. Accessed December 2007.

GO Consortium (2007) *An Introduction to the Gene Ontology*. Online available at http://www.geneontology.org/GO.doc.shtml. Accessed October 2007.

Gold V, Loening KL, McNaught AD, Sehmi P (1987) *Compendium of Chemical Terminology*. Blackwell, Oxford.

Gómez-Pérez A, Fernández-López M, Corcho O (2004) *Ontological Engineering*. Springer, Berlin.

Graßmuck J, Houben KW, Zollinger RM (1994) *DIN-Normen in der Verfahrenstechnik*. Teubner, Stuttgart.

Green DW, Perry RH (deceased) (1997) *Perry's Chemical Engineers' Handbook.* 7th Edition. McGraw-Hill, New York.

Gruber TR (1993) A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.* **5** (2):199–220.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

Grüninger M, Fox MS (1995) Methodology for the design and evaluation of ontologies. In: Skuce D (ed.): *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*.

Guarino N (1997a) Understanding, building, and using ontologies: A commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga. *Int. J. Hum Comput Stud.* **46** (2/3):293–310.

Guarino N (1997b) Semantic matching: formal ontological distinctions for information organization, extraction, and integration. In: Pazienza MT (ed.): *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer, Berlin:139–170.

Guarino N (1998) Formal ontology and information systems. In: Guarino N (ed.): *Formal Ontology in Information Systems*. IOS Press, Amsterdam:3–15.

Guarino N, Boldrin L (1993) Ontological requirements for knowledge sharing. In: Skuce D (ed.): *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*.

Guarino N, Giaretta P (1995) Ontologies and knowledge bases: towards a terminological clarification. In: Mars N (ed.): *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam:25–32.

Guthrie K (1969) Data and techniques for preliminary capital cost estimation, *Chem. Eng. (New York)* **24** (3):114-142.

Gutsche B (1986) Phase equilibria in oleochemical industry – application of continuous thermodynamics. *Fluid Phase Equilib.* **30**:65-70.

Haarslev V, Möller R, van der Straeten R, Wessel M (2004) Extended query facilities for racer and an application to software-engineering problems. In: *Proceedings of the 2004 International Workshop on Description Logics (DL-2004)*:148–157.

Haase R (1990) *Thermodynamics of Irreversible Processes*, Dover Publications, New York.

Hackenberg J (2006) *Computer Support for Theory-Based Modeling of Process Systems*. Fortschritt-Berichte VDI, Reihe 3, Nr. 860, VDI-Verlag, Düsseldorf.

Hai R, Theißen M, Marquardt W (2009) An integrated ontology for operational processes. In: Jezowski J, Thullie J (eds.): *Proceedings of the 19th European Symposium on Computer-Aided Process Engineering*. Elsevier:1087–1091.

Hairer E, Wanner G (1996) *Solving Ordinary Differential Equations II -- Stiff and Differential-Algebraic Problems*, Springer, Berlin.

Halevy AY (2001) Answering queries using views: a survey. *VLDB J.* **10** (4):270–294.

Hammer M, Champy J A (1993) *Reengineering the Corporation: a Manifesto for Business Revolution.* HarperCollins, NewYork.

Hariu OH, Sage RC (1969) Crude split figured by computer. *Hydrocarbon Process., Int. Ed.* **48** (4):143-148.

Hawley K (2004) Temporal Parts. In: Zalta EN (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2004 Edition)*. Online available at http://plato.stanford.edu/archives/win2004/entries/temporal-parts/. Accessed October 2007.

Heflin J, Hendler J (2000) Dynamic ontologies on the web. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).* AAAI Press, Menlo Park, CA:443–449.

Hendler J (2007) Where are all the intelligent agents? *IEEE Intell. Syst.* **22** (3):2-3.

Hodges W (1983) Elementary predicate logic. In: Gabbay DM, Guenthner F (eds.): *Handbook of Philosophical Logic – Vol. I: Elements of Classical Logic*. Reidel, Dordrecht:1–131.

Hoekstra R, Breuker J, Di Bello M, Boer A (2007) The LKIF core ontology of basic legal concepts. In: Casanovas P, Biasiotti MA, Francesconi E, Sagri MT (eds.): *Proceedings of the 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques*. CEUR Workshop Proceedings:43-63.

Hofweber T (2005) Logic and ontology. In: Zalta EN (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2005 Edition)*. Online available at http://plato.stanford.edu/achives/win2005/entries/logic-ontology/. Accessed January 2007.

Horrocks I (1998) Using an expressive description logic: FaCT or fiction? In: Cohn AG, Schubert L, Shapiro SC (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*. Morgan Kaufmann, San Francisco:636–647.

Horrocks I, Patel-Schneider P (2004) Reducing OWL entailment to description logic satisfiability. *J. Web Sem.* **1** (5):345–357.

Horrocks I, Patel-Schneider P, Boley H, Tabet S, Grosof B, Dean M (2004) *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C Member Submission 21 May 2004. Online available at http://www.w3.org/Submission/SWRL/. Accessed December 2007.

Hubert H, van Houten F, eds. (1999) *Integration of Process Knowledge into Design Support Systems.* Springer.

HP Labs (2007) *HP Labs Semantic Web Research.* Available at www.hpl.hp.com/semweb/. Accessed January 2008.

IEEE (1990) *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Standard 610.12-1990, Institute for Electrical and Electronics Engineering, New York.

IEEE (2000) *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*. IEEE Standard 1471-2000, Institute for Electrical and Electronics Engineering, New York.

Imai M (1997) *Gemba Kaizen: A Commonsense, Low-Cost Approach to Management*. McGraw-Hill, New York.

Incropera FP, De Witt DP (1990) *Fundamentals of Heat and Mass Transfer* (3rd ed.). John Wiley, New York.

Innotec (2008) *Innotec – Product Overview*. Online available at http://www.innotec.com/produktuebersicht.html?&L=1. Accessed March 2008.

Intergraph (2008). *Intergraph: SmartPlant foundation*. Online available at http://www.intergraph.com/products/ppm/smartplant/default.aspx. Accessed March 2008.

Ion P, Miner R, eds. (1999) *Mathematical Markup Language (MathML) 1.01 Specification*. W3C Recommendation, revision of 7 July 1999. Online available at http://www.w3.org/TR/REC-MathML/. Accessed April 2007.

ISO (1994) Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. International Standard ISO 10303-11:1994, International Organization for Standardization, Geneva, Switzerland.

ISO (2003) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 2: Data model. International Standard ISO 15926-2:2003, International Organization for Standardization, Geneva, Switzerland.

ISO (2004) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 1: Overview and fundamental principles. International Standard ISO 15926-1:2004, International Organization for Standardization, Geneva, Switzerland.

ISO (2005) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 7: Implementation methods for data exchange and integration. International Standard under development ISO/CD TS 15926-7, International Organization for Standardization, Geneva, Switzerland.

ISO (2006) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 3: Ontology for geometry and topology. International Standard under development ISO/NP TS 15926-3, International Organization for Standardization, Geneva, Switzerland.

ISO (2007) Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 4: Initial reference data. International Standard ISO/TS 15926-4:2007, International Organization for Standardization, Geneva, Switzerland.

ISO 10303, Part 231 (1998) *Process Engineering Data: Process Design and Process Specifications of Major Equipment*. ISO TC 184/SC4/WG3 N740.

ISO/IEC 12207 (2008) *Systems and Software Engineering – Software life cycle processes.*

Jacobson I, Booch G, Rumbaugh J (2003) *The Unified Software Development Process: UML.* Addison-Wesley.

Jarke M, Marquardt W (1995) Design and Evaluation of Computer-Aided Process Modeling Tools. In: Davis J, Stephanopoulos G, Venkatasubramanian V (eds): *Intelligent Systems in Process Engineering,* AIChE Symp. Ser., **312** (92):97-109.

Jarke M, Gallersdörfer R, Jeusfeld MA, Staudt M, Eherer S (1995) ConceptBase – a deductive object base for meta data management. *J. Intell. Inf. Syst.* **4** (2):167–192.

Jarke M, List T, Weidenhaupt K (1999) A process-integrated conceptual design environment for chemical engineering. In: *Proceedings of the 18th International Conference on Conceptual Modeling*. Springer, Berlin:520–537.

Jarke M, List T, Köller J (2000) The challenge of process data warehousing. In: *Proceedings of the 26th International Conference on Very Large Databases*. Morgan Kaufmann:473–483.

Jarke M, Lenzerini M, Vassiliou Y, Vassiliadis P (2003) *Fundamentals of Data Warehouses*. Springer, Berlin.

Jarrar M, Meersman R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine SSGRR2002*.

Jensen AK (1998) *Generation of problem specific simulation models within an integrated computer aided system.* PhD Thesis, Department of Chemical Engineering, Technical University of Denmark.

Karnopp DC, Margolis DL, Rosenberg RC (1990) *System Dynamics: A Unified Approach*. John Wiley, New York.

Kifer M, Lausen G, Wu J (1995) Logical foundations of object-oriented and frame-based languages. *JACM* **42** (4):741–843.

Killich S, Luczak H, Schlick C, Weißenbach M, Wiedenmaier S, Ziegler J (1999) Task modelling for cooperative work. *Behaviour and Information Technology* **18** (5):325-338.

Kim Y, Kang S, Lee S, Yoo S (2001) A distributed, open, intelligent product data management system. *Int. J. Computer Integr. Manuf.* **14**:224–235.

Kitamura Y, Mizoguchi R (1999) Meta-functions of artifacts. In: Price C (ed.): *Proceedings of the 13th International Workshop on Qualitative Reasoning*. University of Aberystwyth:136–145.

Kitamura Y, Mizoguchi R (2003) Ontology-based description of functional design knowledge and its use in a functional way server. *Expert Syst. Appl.* **24** (2):153–166.

Kitamura Y, Koji Y, Mizoguchi R (2006) An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology* **1** (3-4):237–262.

Klein M (2002) Interpreting XML documents via an RDF schema ontology. In: *Proceedings of the 13th International Workshop on Database and Expert Systems Applications*:889–893.

Klinker G, Bhola C, Dallemagne G, Marques D, McDermott J (1991) Usable and reusable programming constructs. *Knowl. Acquis.* **3** (2):117–135.

Klir GJ (1985) *Architecture of Systems Problem Solving*. Plenum Press, New York.

Konda S, Monarch I, Sargent P, Subrahmanian E (1992) Shared memory in design: A unifying theme for research and practice. *Res. Eng. Des.* **4**:23–42.

Kopena J, Regli WC (2003) Functional modeling of engineering designs for the semantic web. *IEEE Data Eng. Bull.* **26** (4):55–61.

Kozaki K, Kitamura Y, Ikeda M, Mizoguchi R (2000) Development of an environment for building ontologies which is based on a fundamental consideration of “relationship” and “role”. In: Compton P, Hoffmann A (eds.): *Proceedings of the 6th Pacific Knowledge Acquisition Workshop*. University of New South Wales:205–221.

Krishna R, Taylor R (1993) Multicomponent mass transfer: theory and applications. In: Cheremisino NP (ed): *Handbook of Heat and Mass Transfer*, Gulf Publishing Company, **2**:259-432.

Krobb C, Lohmann B, Marquardt W (1998) *The Chemical Engineering Data Model VeDa. Part 6: The Process of Model Development*. Technical Report (LPT-1998-06), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Kumar A, Gupta R (1998) *Fundamentals of Polymers*. McGraw-Hill, New York.

Lang HJ (1947) Engineering Approach to Preliminary Cost Estimates, *Chem. Eng. (New York)*:130-133.

Lassila O, McGuinness D (2001) *The Role of Frame-Based Representation on the Semantic Web*. Technical Report (KSL-01-02), Knowledge Systems Laboratory, Stanford University. Online available at http://www-ksl. stanford.edu/KSL_Abstracts/KSL-01-02.html. Accessed October 2007.

Lauber J (1996) *Methode zur funktionalen Beschreibung und Analyse von Produktionsprozessen als Basis zur Realisierung leittechnischer Lösungen*. Dissertation, Lehrstuhl für Prozessleittechnik, RWTH Aachen University.

Lenat D, Guha RV (1990) *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison Wesley.

Levenspiel O (1999). *Chemical Reaction Engineering*. John Wiley & Sons, Inc., New York.

Linninger A (2000) Towards computer-aided model generation. In: *Proceedings of the JSPS International Workshop on Safety-Assured Operation and Concurrent Engineering*. Yokohama, Japan:C35–C49.

Linstrom PJ, Mallard WG, eds. (2005) NIST Chemistry WebBook. NIST Standard Reference Database Number 69, June 2005, National Institute of Standards and Technology, Gaithersburg, MD. Online available at http://webbook.nist.gov.

Little EG, Rogova GL (2005) *Ontology Meta-Model for building a situational picture of catastrophic events*. In: 8th International Conference on Information Fusion 2005, DOI: 10.1109/ICIF. 2005.1591935.

Lloyd CM, Halstead MDB, Nielsen PF (2004) CellML: its future, present and past. Progress in Biophysics and Molecular Biology **85** (2-3):433-450.

Lloyd JW (1987) *Foundations of Logic Programming, 2nd edition*. Springer, Berlin.

Lohmann B, Marquardt W (1998) Entwicklungsprozesse für den konzeptionellen Entwurf. In: Nagl M, Westfechtel B (eds.): *Integration von Entwicklungssystemen in Ingenieuranwendungen*. Springer, Berlin:253-264.

Lu ML, Batres R, Li HS, Naka Y (1997) A G2 based MDOOM testbed for concurrent process engineering. *Comput. Chem. Eng.* **21**:11–16.

Maier A, Schnurr HP, Sure Y (2003) Ontology-based information integration in the automotive industry. In: *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Florida:897-912.

Mangold M, Angeles-Palacios O, Ginkel M, Kremling A, Waschler R, Kienle A, Gilles ED (2005) Computer-aided modeling of chemical and biological systems: methods, tools and applications. *Ind. Eng. Chem. Res.* **44**:2579–2591.

Maropoulos PG (2003) Digital enterprise technology – defining perspectives and research priorities. *Int. J. Computer Integr. Manuf.* **16** (7/8):467–478.

Marquardt W (1992a) An object-oriented representation of structured process models. *Comput. Chem. Eng.* **16**:329–336.

Marquardt W (1992b) Rechnergestützte Erstellung verfahrenstechnischer Prozessmodelle. *Chem. Ing. Tech.* **64**:25-40

Marquardt W (1994a) Trends in Computer-Aided Process Modeling. *5th International Symposium on Process Systems Engineering, PSE'94,* Proceedings PSE'94:1-24

Marquardt W (1994b) Computer-aided generation of chemical engineering process models. *Int. Chem. Eng.* **34**:28–46.

Marquardt W (1995) Towards a Process Modeling Methodology. In: Berber, R: *Methods of Model-Based Control.* NATO-ASI E, Applied Sciences, **293**, Kluwer, Dordrecht:3-41.

Marquardt W (1996) Trends in computer-aided process modeling. *Comput. Chem. Eng.* **20** (6/7):591–609.

Marquardt W, Nagl M (2004) Workflow and information centered support of design processes – the IMPROVE perspective. *Comput. Chem. Eng.* **29** (1):65–82.

Marquardt W, Gerstlauer A, Gilles ED (1993) Modeling and Representation of Complex Objects: A Chemical Engineering Perspective, *6th Int. Conf. on Industrial and Engineering Applications to Artificial Intelligence and Expert Systems*, Proceedings:219-228.

Marquardt W, von Wedel L, Bayer B (2000) Perspectives on lifecycle process modeling. In: Malone MF, Trainham JA, Carnahan B (eds.): *Foundations of Computer–Aided Process Design.* AIChE:192–214.

Martinson WS, Barton PI (2000) A differentiation index for partial differential-algebraic equations. *SIAM J. Sci. Comput.* 21:2295–2315.

Matthes F (1959) Zur Systematik der chemischen Technologie – Teil 2, *Chem. Tech. (Leipzig)*:536-543.

McCabe WL, Smith JC, Harriott P (2004) *Unit Operations in Chemical Engineering*. 7th Edition, McGraw-Hill, New York.

McGuinness DL (2002) Ontologies come of age. In: Fensel D, Hendler J, Lieberman H, Wahlster W (eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press:171–194.

McLaughlin B, Bennett K (2005) *Supervenience*. Online available at http://plato.stanford.edu/entries/supervenience/. Accessed December 2006.

McMahon C, Lowe A, Culley SJ (2004) Knowledge management in engineering design, personalisation and codification. *J. Eng. Des.* **15** (4):307–325.

McNaught AD, Wilkinson A, eds. (1997) Compendium of Chemical Terminology, 2nd Edition. Blackwell Science, Oxford, UK. Electronic version online available at http://goldbook.iupac.org/.

Mesihovic S, Malmqvist J, Pikosz P (2004) Product data management system-based support for engineering project management. *J. Eng. Des.* **15** (4):389–403.

Miatidis M, Jarke M (2005) Integrating workflow extensions into a process-integrated environment for chemical engineering. In: *Proceedings of the 7th International Conference on Enterprise Information Systems*:255–260.

Miatidis M, Jarke M, Weidenhaupt K (2008) Using developers' experience in cooperative design processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:185–223.

Miller DC, Josephson JR, Elsass MJ, Davis JF, Chandrasekaran B (1997) Sharable Engineering Databases for Intelligent System Applications. *Comput. Chem. Eng.* **21**:77-82.

Minsky M (1975) A framework for representing knowledge. In: Winston PH (ed.): *The Psychology of Computer Vision*. McGraw-Hill, New York.

Mizoguchi R (2001) Ontological engineering: foundation of the next generation knowledge processing. In: Zhong N, Yao Y, Liu J, Ohsuga S (eds.): *Web Intelligence: Research and Development*. Springer, Berlin:44–57.

Mizoguchi R, Vanwelkenhuysen J, Ikeda M (1995) Task ontologies for reuse of problem solving knowledge. In: Mars N (ed.): *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam:46–57.

Mizoguchi R, Kozaki K, Sano T, Kitamura Y (2000) Construction and deployment of a plant ontology. In: Dieng R, Corby O (eds.): *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. Springer, Berlin:113–128.

Modell M, Reid RC (1983) Thermodynamics and its Applications. Second ed., Prentice-Hall, Englewood Cliffs.

Molitor R (2000) *Unterstützung der Modellierung verfahrenstechnischer Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken*. PhD thesis, Department of Computer Science, RWTH Aachen University.

Morbach J, Marquardt W (2006) Wissenssprache im Anlagenbau – die Erstellung von Branchenleistungsverzeichnissen mit Hilfe von Ontologien. In: Schenk M (ed.): *Industriearbeitskreis „Kooperation im Anlagenbau“: Arbeitsbericht 03/04*. Fraunhofer IRB Verlag, Stuttgart:1–17.

Morbach J, Marquardt W (2008) Ontology-based integration and management of distributed design data. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:647–655.

Morbach J, Yang A, Marquardt W (2007) OntoCAPE – a large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20** (2):147–161.

Morbach J, Theißen M, Hai R, Marquardt W (2008a) An introduction to application domain modeling. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:83–92.

Morbach J Bayer B, Yang A, Marquardt W (2008b) Product data models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:93–110.

Morbach J, Hai R, Bayer B, Marquardt W (2008c) Document models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:111–125.

Morbach J, Theißen M, Marquardt W (2008d) Integrated application domain models for chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:169–182.

Morbach J, Wiesner A, Marquardt W (2008e) OntoCAPE 2.0 – a (re)usable ontology for computer-aided process engineering. In: Braunschweig B, Joulia X (eds.): 18th European Symposium on Computer Aided Process Engineering. Elsevier:991–996.

Morbach J, Wiesner A, Marquardt W (2008f) *OntoCAPE 2.0 – The Meta Model*. Technical Report (LPT-2008-24), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-24.

Morbach J, Bayer B, Wiesner A, Yang A, Marquardt W (2008g) *OntoCAPE 2.0 – The Upper Level*. Technical Report (LPT-2008-25), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-25.

Morbach J, Yang A, Wiesner A, Marquardt W (2008h) *OntoCAPE 2.0 – Supporting Concepts*. Technical Report (LPT-2008-26), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-26.

Morbach J, Yang A, Marquardt W (2008i) *OntoCAPE 2.0 – Materials*. Technical Report (LPT-2008-27), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-27.

Morbach J, Yang A, Marquardt W (2008j) *OntoCAPE 2.0 – Mathematical Models*. Technical Report (LPT-2008-28), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-28.

Morgenstern L, Riecken D (2005) SNAP: An action-based ontology for e-commerce reasoning. In: *Proceedings of the 1st Workshop "FOMI 2005" – Formal Ontologies Meet Industry*.

Moss GP, ed. (1996) Basic terminology of stereochemistry. IUPAC Recommendations 1996, *Pure Appl. Chem.* **68**:2193-2222.

Moss GP, Smith PAS, Tavernier D, ed. (1995) Glossary of class names of organic compounds and reactive intermediates based on structure. IUPAC Recommendations 1995, *Pure Appl. Chem.* **67**:1307-1375.

Müller P, ed. (1994) Glossary of terms used in physical organic chemistry. IUPAC Reccomendations 1994, *Pure Appl. Chem.* **66**:1077-1184.

Nagl M, Marquardt W, eds. (2008) *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support*. Springer, Berlin.

Neches R, Fikes R, Finin T, Gruber T, Patil R, Senator T, Swartout WR (1991) Enabling technology for knowledge sharing. *AI Mag.* **12** (3):36–56.

Newell A (1982) The knowledge level. *Artif. Intel.* **18** (1):87–127.

Niles P (2001) Towards a standard upper ontology. In: Guarino N, Welty C, Smith B (eds.): *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*. ACM:2–9.

Nonaka I, Takeuchi H (1995) *The Knowledge Creating Company*. Oxford University Press, New York.

Noonan H (2006) Identity. In: Zalta EN (ed.): *The Stanford Encyclopedia of Philosophy (Winter 2006 Edition)*. Online available at http://plato.stanford.edu/archives/win2006/ entries/identity/. Accessed January 2008.

Noumonon Consulting Limited (2008) *Open Access to Intelligent Process Plant Models*. Online available at http://www.noumenon.co.uk/. Accessed April 2008.

Noy NF, Klein M (2004) Ontology evolution: not the same as schema evolution. *Knowl. Inform. Syst.* **6**:428–440.

Noy N, Rector A, eds. (2006) *Defining N-ary Relations on the Semantic Web*. W3C Working Group Note, 12 April 2006. Online available at http:// www.w3.org/TR/ swbp-n-aryRelations/. Accessed December 2007.

Oberle D, Volz R, Motik B, Staab S (2004) An extensible ontology software environment. In: Staab S, Studer R (eds.): *Handbook on Ontologies*. Springer, Berlin:311–333.

Odell JJ (1994) Six different kinds of composition. *Journal of Object-Oriented Programming* **5**(8). Online available at http://www.conradbock.org/ compkind.html. Accessed December 2006.

OLS – Ontology Lookup Service (2006) ChEBI Ontology Browser. Online available at http://www.ebi.ac.uk/ontology-lookup/browse.do?ontName=CHE BI. Accessed February 2007.

Ontoprise GmbH (2009) *OntoStudio*. Online available at http://www.ontoprise.de/en/ home/products/ontostudio/. Accessed April 2009.

OWL (2002) OWL representation ontology. Online available at http://www.w3. org/2002/07/owl. Accessed October 2007.

Pâslaru-Bontaş E (2007) *Contextual Approach to Ontology Reuse: Methodology, Methods and Tools for the Semantic Web*. PhD Thesis, FU Berlin.

Patel-Schneider PF, Horrocks I (2006) *OWL 1.1 Web Ontology Language Overview*. W3C Member Submission, 19 December 2006. Online available at http://www.w3.org/Submission/owl11-overview/. Accessed October 2007.

Paton NW, Goble CA, Bechhofer S (2000) Knowledge-based information integration systems. *Inform. Software Technol.* **42** (5):299–312.

Patzak G (1982) *Systemtechnik – Planung komplexer innovativer Systeme*. Springer, Berlin.

Perkins JD, Sargent RWH, Vazquez-Roman R, Cho JH (1996) Computer generation of process models. *Comput. Chem. Eng.* **20** (6):635–639.

Peters MS, Timmerhaus KD (1991) *Plant Design and Economics for Chemical Engineers*. McGraw-Hill, New York.

Pinto HS, Gomez-Perez A, Martins JP (1999) Some issues on ontology integration. In: *Proceedings of the IJCAI'99 Workshop on Ontologies and Problem Solving Methods*.

Pohl K, Weidenhaupt K, Dömges R, Haumer P, Jarke M, Klamma R (1999) PRIME: Towards Process-Integrated Environments. *ACM Transactions on Software Engineering and Methodology* **8** (4):343–410.

Polke M, ed. (1994) *Process Control Engineering*. VCH, Weinheim.

Preisig HA (1995) MODELLER – An object-oriented computer-aided modeling tool. In: Biegler LT, Doherty MF (eds.): *Foundations of Computer-Aided Process Design*. AIChE:328–331.

Prolist (2009) *Prolist*. Online available at: http://www.prolist.org.

PTC (2008) *Windchill*. Online available at http://www.ptc.com/products/windchill/. Accessed May 2008.

Pure-Systems (2008) *Pure::Variants*. Online available at http://www.pure-systems.com. Accessed September 2008.

Racer Systems (2006) *What is Racer Pro?* Webpage, http://www.racer-systems.com/products/racerpro/index.phtml. Accessed December 2006.

Racer Systems (2007) *RacerPro Reference Manual*. Online available at http://www.racer-systems.com/products/racerpro/reference-manual-1-9-2-beta.pdf. Accessed June 2008.

Raddatz M, Schlüter M, Brandt SC (2006) Identification and reuse of experience knowledge in continuous production processes. Presented at the *9th IFAC Symposium on Automated Systems Based on Human Skill and Knowledge*. Online available at http://www-i5.informatik.rwth-aachen.de/i5new/publications/pubs2006.html. Accessed May 2008.

Ramkrishna D (1985) The status of population balances. *Rev. Chem. Eng.* **3**:49-95.

Rector A (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Genari J (ed.): *Knowledge Capture 2003*. ACM Press:121–128.

Rector A, ed. (2005) *Representing Specified Values in OWL: "value partitions" and "value sets"*. W3C Working Group Note, 17 May 2005. Online available at http://www.w3.org/TR/swbp-specified-values. Accessed November 2007.

Rector A, Welty C, eds. (2005) *Simple part-whole relations in OWL Ontologies*. W3C Editor's Draft, 11 August 2005. Online available at http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. Accessed November 2007.

Rector A, Solomon WD, Nowlan WA, Rush TW (1995) A terminology server for medical language and medical information systems. *Methods Inf. Med.* **34** (1/2):147–157.

Rector A, Drummond N, Horridge M, Rogers J, Knublauch H, Stevens R, Wang H, Wroe C (2004) OWL pizzas: practical experience of teaching OWL-DL: common errors & common patterns. In: Motta E, Shadbolt N, Stutt A, Gibbins N (eds.): *Proceedings of the European Conference on Knowledge Acquisition (EKAW)*. Springer:63–81.

Russ T, Valente A, MacGregor R, Swartout W (1999) Practical experiences in trading off ontology usability and reusability. In: *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Sandler SI (1999) *Chemical and Engineering Thermodynamics*. Third ed., John Wiley, New York.

Sargent RWH (1998) A Functional Approach to Process Synthesis and its Application to Distillation Systems. *Comput. Chem. Eng.* **22**:31-45.

Schembra M (1991) *Daten und Methoden zur Vorkalkulation des Anlagekapitalbedarfs von Chemieanlagen*. PhD thesis, Technische Universität Berlin.

Schneider L (2003) How to build a foundational ontology: the object-centered high-level reference ontology OCHRE. In: Günter A, Kruse R, Neumann B (eds.): *KI 2003: Advances in Artificial Intelligence*. Springer, Berlin:120–134.

Schuler H, ed. (1999) *Prozessführung*. Oldenbourg, München.

Schulze-Kremer S (1998) Ontologies for molecular biology. In: *Proceedings of the Third Pacific Symposium on Biocomputing*. AAAI Press:693–704.

Schummer J (1998) The chemical core of chemistry I: a conceptual approach. HYLE – *Int. J. Philosophy Chem.* **4**(2):129-162.

Seader JD, Henley EJ (1998) *Separation Process Principles*. John Wiley, New York.

Seipel D, Baumeister J (2004) Declarative methods for the evaluation of ontologies. *Künstliche Intelligenz* **18** (4):51–57.

Sheremetov L, Batyrshin I, Chi M, Vergara E, Rosales A (2007) Knowledge-based collaborative engineering of pipe networks in the upstream and downstream petroleum industry. In: Shen W, Yang Y, Yong J, Hawryszkiewycz I, Lin Z, Barthes JPA, Maher ML, Hao Q, Tran MH (eds.): *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007)*. IEEE:458–463.

Shreve RN (1978) *Chemical Process Industries*. McGraw-Hill, New York.

Siemens PLM Software (2008) *Teamcenter*. Online available at http://www.ugs.com/ products/teamcenter/. Accessed May 2008.

Simmons B (2007) *Mathwords* – website. Online available at http://www.mathwords.com/. Accessed October 2007.

Simons P (1987) *Parts: A Study in Ontology*. Oxford University Press.

Smith B (1996) Mereotopology: a theory of parts and boundaries. *Data Know. Eng.* **20** (3):287–303.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Smith EG (1968) *The Wiswesser Line-Formula Chemical Notation*. McGraw-Hill, New York.

Smith JM (1981) *Chemical Engineering Kinetics.* 3rd Edition, McGraw-Hill, New York.

Smith JM, Van Ness HC (1975) Introduction to Chemical Engineering Thermodynamics, 3rd Edition. McGraw-Hill, New York.

Smith MK, Welty C, McGuinness DL, eds. (2004) *OWL Web Ontology Languages Guide*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-guide/. Accessed October 2007.

Souza D, Marquardt W (1998a) *The Chemical Engineering Data Model VeDa. Part 2: Structural Modeling Objects*. Technical Report (LPT-1998-02), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Souza D, Marquardt W (1998b) *The Chemical Engineering Data Model VeDa. Part 3: Geometrical Modeling Objects*. Technical Report (LPT-1998-03), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Sowa JF (1995) Top-level ontological categories. *Int. J. Hum Comput Stud.* **43** (5/6):669–685.

Stanford Center for Biomedical Informatics Research (2008) *The Protégé Ontology Editor and Knowledge Acquisition System.* Online available at http://protege.stanford.edu/. Accessed January 2008.

Stein SE, Heller SR, Tchekhovski D (2003) An open standard for chemical structure representation – the IUPAC Chemical Identifier. In: *Proceedings of the 2003 Nimes International Chemical Information Conference*:131-143.

Stephanopoulos G, Henning G, Leone H (1990) MODEL.LA. A modeling language for process engineering. I. The formal framework. *Comput. Chem. Eng.* **14** (8):813–846.

Stuckenschmidt H, Klein M (2003) Integrity and change in modular ontologies. In: Gottlob G, Walsh T (eds.): *IJCAI-03 – Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann:900–905.

Studer S, Benjamins VR, Fensel D (1998) Knowledge engineering principles and methods. *Data Knowl. Eng.* **25** (1/2):161–197.

Subramanian E, Rachuri S (2008) Guest editorial: special issue on "engineering informatics". Trans. ASME, J. Computing and Information Science in Engineering. **8**:010301-4.

Szykman S, Sriram RD, Bochenek C, Racz JW, Senfaute J (2000) Design repositories: engineering design's new knowledge base. *IEEE Intell. Syst.* **15** (3):48-55.

Szykman S, Sriram RD, Regli WC (2001) The role of knowledge in next-generation product development systems. *J. Comput. Inf. Sci. Eng.* **1** (1):3–11.

TGL 25000 (1974) *Chemical Engineering Unit Operations – Classification.* Departmental Standard of the German Democratic Republic.

Teijgeler H (2007) *InfowebML, OWL-based Information Exchange and Integration based on ISO 15926*. Online available at http://www.infowebml.ws/. Accessed November 2007.

Theißen M, Marquardt W (2008) Decision models. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:153–168.

Theißen M, Hai R, Marquardt W (2008a) Computer-assisted work process modeling in chemical engineering. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:656–666.

Theißen M, Hai R, Marquardt W (2008b) Design process modeling in chemical engineering. *J. Comput. Inf. Sci. Eng.* **8** (1), 011007 (9 pages).

Theißen M, Hai R, Morbach J, Schneider R, Marquardt W (2008c) Scenario-based analysis of industrial work processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:433–450.

Theißen M, Hai R, Marquardt W (2009) A framework for work process modeling in the chemical industries. *8th World Congress of Chemical Engineering (WCCE8)*. Montréal, August 2009.

Thomé B, ed. (1993) *Systems Engineering: Principles and Practice of Computer - based Systems Engineering*. John Wiley, New York.

Tiller MM (2001) *Introduction to Physical Modeling with Modelica*. Springer, Berlin.

Tränkle F (2000) *Rechnerunterstützte Modellierung verfahrenstechnischer Prozesse für die Simulationsumgebung DIVA*. Fortschritt-Berichte VDI 309, Reihe 20, Nr. 309. VDI-Verlag, Düsseldorf.

Tränkle F, Gerstlauer A, Zeitz M, Gilles ED (1997) Application of the modeling and simulation environment PROMOT/DIVA to the modeling of distillation processes. *Comput. Chem. Eng.* **21**:841–846.

Tsarkov D, Horrocks I (2007) *FaCT++*. Online available at http://owl.man.ac.uk/factplusplus/. Accessed January 2008.

Ullman D (2002) Toward the ideal mechanical engineering design support system. *Research in Engineering Design* **13** (2), 55– 64.

Unbehauen H (1989) *Regelungstechnik I*. Vieweg, Braunschweig.

Uschold M, Grüninger M (1996) Ontologies: principles, methods and applications. *Knowl. Eng. Rev.* **11** (2):93–136.

Uschold M, King M, Moralee S, Zorgios Y (1998) The enterprise ontology. *Knowl. Eng. Rev.* **13**:31–89.

Valente A, Breuker J (1996) Towards principled core ontologies. In: Gaines BR, Mussen M (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SDRG Publications.

Valente A, Russ T, MacGregor R, Swartout W (1999) Building and (re)using an ontology of air campaign planning. *IEEE Intell. Syst.* **14** (1):27–36.

van Heijst G, Schreiber AT, Wielinga BJ (1997a) Using explicit ontologies in KBS development. *Int. J. Hum Comput Stud.* **46** (2/3):183–292.

van Heijst G, Schreiber AT, Wielinga BJ (1997b) Roles are not classes: a reply to Nicola Guarino. *Int. J. Hum Comput Stud.* **46** (2/3):311–318.

Varzi A (2006) Mereology. In: Zalta EN (ed.): The Stanford Encyclopedia of Philosophy (Winter 2006 Edition). Online available at http://plato.stanford.edu/archives/win2006/entries/mereology/. Accessed January 2007.

Venkatasubramanian V (2009) Drowning in data: informatics and modeling challenges in a data-rich and networked world. *AIChE J.* **55**:2–8.

Venkatasubramanian V, Zhao C, Joglekar G, Jain A, Hailemariam L, Suresh P, Akkisetty P, Morris K, Reklaitis GV (2006) Ontological informatics infrastructure for pharmaceutical product development and manufacturing. *Comput. Chem. Eng.* **30** (10/12):1482–1496.

VIM (1993) *International Vocabulary of Basic and General Terms in Metrology (VIM), 2nd Edition*. Jointly prepared by ISO, IEC, BIPM, IFCC, IUPAC, IUPAP and OIML. Published by ISO, Geneva, as ISO Guide **99**:1993.

Visser PRS, Cui Z (1998) Heterogeneous ontology structures for distributed architectures. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:112–119.

Visser U, Stuckenschmidt H, Wache H, Vögele T (2000) Enabling technologies for interoperability. In: Visser U, Pundt H (eds.): *Workshop: Information Sharing: Methods and Applications at the 14th International Symposium of Computer Science for Environmental Protection*:35–46.

Vogt M (1996) *Neuere Methoden der Investitionsrechnung in der Chemischen Industrie*. Diploma thesis, Technische Universität Berlin.

von Wedel L (2002) *CapeML – A Model Exchange Language for Chemical Process Modeling*. Technical Report (LPT-2002-16), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

von Wedel L, Marquardt W (1999) *The Chemical Engineering Data Model VeDa. Part 5: Material Modeling Objects*. Technical Report (LPT-1998-05), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

von Wedel L, Marquardt W (2000) ROME: a repository to support the integration of models over the lifecycle of model-based engineering. In: Pierucci S (ed.): *Proceedings of the European Symposium on Computer Aided Process Engineering – ESCAPE 10*. Elsevier:535–540.

W3C (2000) *The RDFS representation ontology*. Web resource. Online available at http://www.w3.org/2000/01/rdf-schema. Accessed June 2008.

W3C (2002) *The OWL representation ontology*. Online available at http://www.w3.org/2002/ 07/owl. Accessed October 2007.

W3C (2006) *Extensible Markup Language (XML)*. Online available at http://www.w3.org/XML/. Accessed December 2007.

Wache H, Vögele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hübner S (2001) Ontology-based integration of information – a survey of existing approaches. In: Goméz-Pérez A, Gruninger M, Stuckenschmidt H, Uschold M (eds.): *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*. CEUR Workshop Proceedings:108–117.

Weininger D (1988) SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules *J. Chem. Inf. Comput. Sci.* **28**:31-36.

Weisstein E (2007) *MathWorld* – website. Online available at http://mathworld.wolfram.com/ Accessed October 2007.

Wesselingh JA, Krishna R (2000) *Mass transfer in multicomponent mixtures*. Delft University Press.

Westerberg AW, Subrahmanian E, Reich Y, Konda S, the n-dim group (1997) Designing the process design process. *Comput. Chem. Eng.* **21**:1–9.

Wiesner A, Morbach J, Marquardt W (2007) An overview on OntoCAPE and its latest applications. In: *Proceedings of the 2007 AIChE Annual Meeting*.

Wiesner A, Morbach J, Bayer B, Yang A, Marquardt W (2008a) *OntoCAPE 2.0 – Chemical Process System*. Technical Report (LPT-2008-29), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-29.

Wiesner A, Morbach J, Marquardt W (2008b) Semantic data integration for process engineering desgin data. In: *Proceedings of the 10th International Conference on Enterprise Information Systems – ICEIS 2008*:190-195.

Wilhelm Jr RG (1996) In Search of the Cheshire Cat: The Invisible Automation System Paradox. *ISA Transactions* **35**:321-327.

Windream GmbH (2008) *Managing Documents by Windream*. Online available at http://www.windream.com/. Accessed May 2008.

Wüsteneck KD (1963) Zur philosophischen Verallgemeinerung und Bestimmung des Modellbegriffs. *Deutsche Zeitschrift für Philosophie* **1963**:1504 et sqq.

Yang A, Marquardt W (2004) An ontology-based approach to conceptual process modeling. In: Barbarosa-Póvoa A, Matos H (eds.): *Proceedings of the European Symposium on Computer Aided Process Engineering – ESCAPE 14*. Elsevier:1159–1164.

Yang A, Schlüter M, Bayer B, Krüger J, Haberstroh E, Marquardt W (2003) A concise conceptual model for material data and its applications in process engineering. *Comput. Chem. Eng.* **27** (4):595–609.

Yang A, Morbach J, Marquardt W (2004a) From conceptualization to model generation: the roles of ontologies in process modeling. In: Floudas CA, Agrawal R (eds.): *Sixth International Conference on Foundations of Computer-Aided Process Design*. Omnipress:591–594.

Yang A, Marquardt W, Stalker I, Fraga I, Serra M, Piñol D, Paen D, Roux P, Braunschweig B (2004b) *Principles and Informal Specification of OntoCAPE. COGents*. Technical Report, COGents Project, Information Society Technologies Program IST-2001-34431.

Yang A, Braunschweig B, Fraga ES, Guessoum Z, Marquardt W, Nadjemi O, Paen D, Piñol D, Roux P, Sama S, Serra M, Stalker I (2008) A multi-agent system to facilitate component-based process modeling and design. *Comput. Chem. Eng.* **32** (10):2290–2305.

Index

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1,
© Springer-Verlag Berlin Heidelberg 2010

Index of Concept Descriptions