CONSTRUCTION OF ENGINEERING ONTOLOGIES FOR KNOWLEDGE SHARING AND REUSE

Cover illustration:

Police Constable Rob Piper of the police of Kidderminster, England, displays the plug for his new electric patrol car on Monday Nov. 18, 1996. The battery powered Peugeot 106 car, with a top speed of 56 miles per hour (90 km/h), and a range of 60 miles (96 km), on one overnight charge of electricity, will be used for short journeys around the area.

Computer programs are often used in the design of technical devices. For the design of the electric Peugeot 106, software developed in the OLMECO *project (Chapter 4) has been used. With this software it is possible to predict properties (such as the range) of the device that is being designed without having to build a prototype. Because these programs use sophisticated (engineering) knowledge they are called "knowledge-based systems". Ontologies are specifications of this knowledge and can be used in the development of knowledge-based systems. This thesis is about ontologies, how ontologies can be constructed and how they can be of help for knowledge-based system development.*

The rightmost window appearing on the cover shows a graphical overview of the ontology of technical components that is described in Chapter 3. It has been used to develop the software in the OLMECO *project. The window in the middle shows a knowledge-based system that can determine how a product (in this case a coffee machine) can best be disassembled. It has been developed with an ontology and is described in Chapter 5. The leftmost window gives an example of the output of a knowledge based system. In this case it shows a prediction of the behaviour of a central heating system of a large hospital (Chapter 4).*

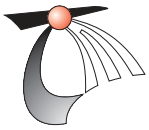
Photograph of car and police constable: Associated Press Photo/David Jones.
Photograph of background: Pim Borst.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Borst, Willem Nico

Construction of Engineering Ontologies for Knowledge Sharing and Reuse /
W. N. Borst – [S.l. : s.n.]. – Ill.
Thesis Enschede – With ref. – With summary.
ISBN: 90-365-0988-2
ISSN: 1381-3617 (CTIT Ph. D-series No. 97-14)
Subject headings: ontologies / knowledge-based systems / engineering design / modelling

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.



CTIT Ph. D-thesis series No. 97-14

P.O. Box 217 - 7500 AE Enschede - The Netherlands
telephone +31-53-4893779/4892100 / fax +31-53-4894524

Centre for
Telematics and
Information
Technology

CONSTRUCTION OF ENGINEERING ONTOLOGIES FOR KNOWLEDGE SHARING AND REUSE

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. F. A. van Vught,
volgens besluit van het College van Promoties
in het openbaar te verdedigen
op vrijdag 5 september 1997 te 15.00 uur.

door
Willem Nico Borst
geboren op 27 augustus 1969
te Akersloot

Dit proefschrift is goedgekeurd door de promotor, prof. dr. J. M. Akkermans.

Contents

Summary

This thesis describes an investigation into the practical use of *ontologies* for the development of information systems. Ontologies are formal descriptions of shared knowledge in a domain. An ontology can be used as a specification of an information system because it specifies the knowledge that is required for the tasks the information system has to perform. Sharing and reuse of ontologies across different domains and applications can therefore improve information systems design.

Ontologies have been a subject for a lot of research carried out in the artificial intelligence community. Although many ontologies have been developed, they fail to demonstrate that ontologies for large and complex domains can be developed that can be *used and reused across different applications*. There are three reasons for this: (i) many ontologies have not been used to develop a real-life application, (ii) many ontologies have not been *reused* for different applications in different domains and (iii) many ontologies are merely taxonomies of domain concepts and fail to capture meta-level and tacit background knowledge.

As a result, the question whether ontologies can be used and reused for different real-life applications remains open. The aim of our research has therefore been to find the answer to this question.

The aim of our research is to investigate the usability and reusability of ontologies by construction and validation of an ontology for a large and complex domain, that

1. is usable for application development,
2. is reusable across different applications and
3. captures meta-level and tacit background knowledge.

Chapter 2 explains what ontologies exactly are, how they can be specified and how they can be used. To make usable ontologies, the domain knowledge must be carved up into modules containing different kinds of knowledge. This makes it possible to construct large and complex ontologies out of smaller and more reusable ones.

Chapter 3 discusses the way a domain ontology for the modeling of physical systems can be constructed. The ontology is called PHYSSYS and describes the domain knowledge required to make simulation models of devices like heating systems, automotive systems and machine tools.

We found out that a large and complex ontology like PHYSSYS can be constructed out of smaller ontologies by carving up the domain knowledge in smaller pieces. Because of this internal structure, the ontology will be easier to understand and be well suited for reuse. The ontologies that form the building blocks of PHYSSYS can be categorized in three types:

1. *`Super'theories* which are general and abstract ontologies.
2. *Viewpoint or base ontologies* that formalize a conceptual category of concepts in a domain.
3. *Domain ontologies* that form an integral and coherent conceptualization of a domain and can be constructed by combining viewpoint ontologies.

To construct a large ontology from smaller ontologies, the dependencies between concepts and relations in different ontologies are formalized as *ontology projections*. Three types of ontology projections were used and are named according to the way they can be implemented.

1. *Include and extend:* An imported ontology is extended with new concepts and relations.
2. *Include and specialize:* An abstract theory is imported and applied to the contents of the importing ontology. Doing this, abstract concepts are specialized.
3. *Include and map:* Different viewpoints on a domain are joined by including the views in the domain ontology and formalization of their interdependencies.

Chapter 4 shows the way PHYSSYS could be used to develop a library of *model fragments*. The library allows engineers to construct large simulation models by combining model fragments from the library. The library has been filled with model fragments for various enginering domains, including the model fragments for thermodynamic systems we developed (see also Appendix B). A modelling and simulation experiment of the existing heating system of the Schieland Hospital (a general hospital in Schiedam, The Netherlands) served as a validation of the library as well as PHYSSYS.

In Chapter 5 is investigated what happens when, instead of an ontology about the knowledge for simulation of technical devices, an ontology for a totally different task in a different engineering domain is constructed. This new task is the ecological impact assessment of product disassembly. It is demonstrated that parts of PHYSSYS can be reused and extended with other (reusable) ontologies to form a new ontology formalizing a novel approach to product disassembly analysis. A knowledge based system called PROMOD has been developed based on the new ontology. PROMOD serves as a prototype for a future extension of commercial software for ecological impact assessment.

In Chapter 6 we draw the conclusion that our experiences with PHYSSYS, the OLMECO library and PROMOD demonstrate that large ontologies can be specified that can indeed be used and reused across applications in different domains. We also summarize the results of our research for ontology construction and describe the roles ontologies may play in the future.

Acknowledgements

Many people have, directly or indirectly contributed to this thesis. Here, I would like to take the opportunity to thank them.

First of all, I would like to thank my promotor Hans Akkermans who initiated and supervised my research. I thank him for being an inspiring promotor and for giving me the opportunity to participate in interesting projects.

I would like to thank the people who helped me to develop the ideas presented in this thesis. First of all, I would like to thank Jan Top for his contributions. I thank Anita Pos, Arno Breunese, Jan Broenink, Pierre-Joseph Gailly, Roger Fisset, Jacques Guyot, Bob Wielinga, Mert Alberts, Jan Benjamin, Willem Wortel and Wim Zeiler for discussions concerning various parts of this work.

Furthermore I would like to thank my colleagues in the Information Systems department for the weekly games of soccer, the miniborrels and the indispensable coffee breaks.

This work has been supported in part by the Commission of the European Communities as Esprit-III project P6521 `OLMECO' (Open Library for Models of MEchatronic COmponents). The partners in the OLMECO project are PSA Peugeot-Citroën (France), BIM (Belgium), FAGOR (Spain), Ikerlan (Spain), Imagine (France), the University of Twente (UT, Enschede, The Netherlands) and the Netherlands Energy Research Foundation (ECN, Petten, the Netherlands). The thermal systems library has benefited from earlier work in the EBIB project, supported by Senter, and with Kropman BV Installatietechniek (The Netherlands), ECN and the University of Twente as partners. I am grateful to Kropman BV for providing data for the Schieland hospital heating system simulation reported in this paper.

Part of this work has been carried out in the SUSTAIN project, a cooperation between PRé Product Ecology Consultants (Amersfoort, The Netherlands), ECN and the University of Twente (Faculty of Computer Science). The SUSTAIN project has been supported by the Dutch Ministry of Economic Affairs SENTER-IT Programme as project nr. ITU95038. I would like to thank the members of the SUSTAIN project, Mark Goedkoop, Vincent Cleij, Cor Warmer, Jan Braam, Hans Akkermans, Wilco Burghout and Martijn de Jong for their enthusiasm and useful comments. I thank Martijn de Jong for making the nice graphical user interface for the PROMOD prototype that appears on the cover of this thesis.

The work on the OLMECO library and the SUSTAIN project were part of the contribution of the Netherlands Energy Research Foundation to these projects. I would like to thank ECN for financing my PhD position at the University of Twente.

I am grateful to my parents for their support and for giving me the opportunity to go to university. Also many thanks to my sister Annemarie Borst for her excellent drawing of the animals of Chapter 2.

Finally, I would like to thank my girlfriend Karen Daman who has supported and encouraged me in writing this thesis. She had to go hiking in the woods of Twente on her own (which is fortunately not very dangerous in this part of the country) because I was too busy writing this thesis.

Chapter 1

Mechanisms for Knowledge Sharing and Reuse

In this chapter, the motivation and aim of our research will be explained. Furthermore, the application domain and context in which the research has been carried out will be described. We will conclude with an overview of the structure of this thesis and a list of publications that have reported on parts of our research.

Section 1.1 gives an introduction into knowledge sharing and reuse. We will see how it can be helpful for the development of information systems. Ontologies have been proposed as a mechanism for knowledge sharing and reuse and have been studied and developed in artificial intelligence. Despite this research, the question whether ontologies can be used and reused for different real-life applications remains open. Finding the answer to this question has been the aim of this research (see Section 1.2). In Section 1.3 we will give an indication of the type of knowledge and applications that served as test cases to develop our theories and ideas. The remaining sections will describe the context in which the research has been carried out, the organization of this thesis and the publications that reported on parts of our research.

1.1 Motivation: Knowledge Sharing and Reuse

From the beginning of the computer era, programmers have sought for ways to reduce the effort of software development. The earliest attempt to achieve this has been the sharing and reuse of pieces of software. The instrument for distribution and sharing were extensive *libraries of reusable software modules*. Good examples of libraries that were in the forefront of these developments are the Fortran subroutine libraries for numerical computation. The desire to improve the abilities for reuse of software modules has been one of the main incentives for the development of structured and modular programming languages like Pascal and Modula.

The next step to improve the reuse of software modules was to share not only the modules themselves, but also *knowledge about the modules*. This approach originated in the mid 1970s when knowledge-based systems (KBSs) like Dendral (Buchanan and Feigenbaum 1978) for elucidating chemical structures and Mycin (Shortliffe 1976), a system to diagnose infectious blood diseases were developed. These systems employed a separation between the domain knowledge they reasoned about (often expressed as rules or frames) and the inference engine (the methods) they used to draw conclusions from the knowledge. Because these systems stored their knowledge in a format that was independent of the content of the knowledge, KBS developers realized that the inference engines could be used relatively independent of the domain.

This has led to *generic tasks* (Chandrasekaran 1988) and *problem- solving methods* (McDermott 1989). A generic task is a task that can be used in applications in different domains of expertise. Examples of generic tasks are classification, interpretation, diagnosis, planning and design. A problem-solving method is a specification of how a task can be performed. An example of a problem-solving method for design is *propose and revise*. This method defines that a design can be found by first proposing a partial solution and then resolving violated constraints by revising this partial solution. Identification of the tasks to be performed by an information system can direct the developer to off the shelf problem-solving methods that can be implemented. Tasks and problem-solving methods are specified at the knowledge level (Newell 1982), i.e. they are independent of the computer language the methods are implemented in. Thus, sharing and reuse of knowledge about problem-solving methods leads to more effective development of knowledge-based systems.

A recent approach to support information-systems development is to address the domain knowledge side of the knowledge–method dichotomy. Instead of identification of generic and reusable structures in the inference engine of a KBS, *reusable pieces of the domain knowledge* are specified and reused. *Ontologies* have been proposed as a specification mechanism to enhance this type of knowledge sharing and reuse across different applications (Neches, Fikes, Finin, Gruber, Senator, and Swartout 1991). We will give examples of ontology development below.

Naive Physics and Commonsense Knowledge An example of an ontology about naive physics is the ontology for liquids developed by Hayes (1985). Naive physics can be hard to formalize: the problem with knowledge about liquids is that they have no definite shape and can merge split mix in mysterious ways. Formalizations of knowledge about physical objects can be found in (Clarke 1981; Simons 1987; Cohn, Randell, and Cui 1995; Borgo, Guarino, and Masolo 1996b). Ontologies of microscopic and macroscopic views on the electrical domain are combined by Liu (1992). The aim of the Cyc project (Lenat and Guha 1990) is to build up a large knowledge base with commonsense knowledge. To help structuring the knowledge in the knowledge base, an ontology of common-sense top-level concepts has been developed. Other formalizations of naive physics and commonsense knowledge can be found in (Hobbs and Moore 1985; Davis 1990; Hobbs 1995).

Natural-Language Processing The Penman Upper Model (Bateman, Magnini, and Rinaldi 1994), is a general model about natural language that can be used for the generation and processing of different languages (Italian, German and English). Other ontologies formalize the semantics of the part–whole relation in natural language (Gerstl and Pribbenow 1995). Natural language about movement in the French language is formalized in (Sablayrolles 1993). Ontologies are also used for the development of systems for extraction of knowledge from abstracts of scientific articles (van der Vet, Speel, and Mars 1994; van der Vet, Speel, and Mars 1995). An ontology used for the development of a consultation system for financial investment can be found in (Horacek 1994).

Engineering and Technical Applications Many ontologies have been developed for engineering and technical applications. An ontology for the Sisyphus elevator design problem (VT) is described in (Schreiber and Terpstra 1996). In the KACTUS project (Laresgoiti, Anjewierden, Bernaras, Corera, Schreiber, and Wielinga 1996; Bernaras and Laresgoiti 1996), ontologies for diagnosis of electrical networks and for the exchange of knowledge about ship design and oil platforms have been written. The YMIR ontology (Alberts 1993) is a domain independent, sharable ontology for the formal representation of engineering design-knowledge, based on systems theory. The PHYSSYS ontology (Borst, Akkermans, and Top 1997) described in Chapter 3 has the same objectives, but is less biased to a mathematical representation. Knowledge formalized in PHYSSYS has been used to develop a number applications: 007 (Pos 1997), a model revision assistant, the OLMECO library of model fragments for simulation (Chapter 4) and, in parts, a prototype system for ecological product disassembly analysis (Chapter 5). EngMath (Gruber 1994) is an ontology for mathematical modelling in engineering. It has been reused many times, for instance in PHYSSYS and in CML. CML (Falkenhainer, Farquar, Bobrow, Fikes, Forbus, Gruber, Iwasaki, and Kuipers 1994) is an ontology about time, continuity, object properties etc. to enable the sharing of models based on compositional modelling (Falkenhainer and Forbus 1991; Forbus 1984). The CML ontology has been used to develop an ontology for thermodynamic systems and an ontology for VT.

Corporate Memory and Enterprise Modelling The TOVE ontology (Fox, Chionglo, and Fadel 1993; Gruninger and Fox 1994), formalizes knowledge about production/communication processes, activities, causality, resources, quality and cost in business enterprises. Ontologies have also been developed for the implementation of knowledge bases for formalization and conservation of the knowledge of experts in enterprises. An example of such an ontology is the KONE ontology (Kühn 1994) that deals with conservation of corporate knowledge about crankshaft design.

Medical Diagnosis Knowledge in the medical domain about diagnosis, therapy planning and patient monitoring has been formalized in the GAMES-II project (Falasconi and Stefanelli 1994; van Heijst, Schreiber, and Wielinga 1997).

Information-System Specification and Design As we have sketched in Section 1.1, knowledge about generic tasks and problem-solving methods can support information system design. A group of researchers in AI have worked on ontologies that formalize what the knowledge about tasks and methods look like (Angele, Decker, Perkuhn, and Struder 1996; Fensel, Schönegge, Groenboom, and Wielinga 1996). Takagaki, Wand and Weber (Takagaki and Wand 1991; Wand and Weber 1990) have worked on ontologies that formally describe information systems. Their goal is to overcome the lack of theoretical foundations of information-system design methodologies.

Knowledge Acquisition As an ontology formally specifies meta-level domain knowledge, it can be an excellent specification for tools that acquire knowledge from domain experts. A good example of a project in this field is the PROTÉGÉ-II project (Puerta, Egar, Tu, and Musen 1992).

The examples mentioned above have in common that the domains and applications they are about are knowledge intensive. Many applications have the additional problem that they have to combine diverse knowledge from different disciplines to perform their tasks. It is exactly for these types of applications that developers can benefit from knowledge sharing and reuse with ontologies.

1.2 Aim: Investigate the Usability and Reusability of Ontologies

In view of the large number of ontologies that have been developed it is tempting to conclude that the use of ontologies is a widely accepted and well founded practice. Unfortunately, it is still too premature to draw this conclusion. There are three reasons for this:

1. Many ontologies have not been *used* to develop a real-life application.
2. Many ontologies have not been *reused* for different applications in different domains.
3. Many ontologies are merely taxonomies of domain concepts and fail to capture meta-level and tacit background knowledge.

Research into the extent to which ontologies can be used and reused for application development is therefore justified.

> The aim of our research is therefore to investigate the usability and reusability of ontologies by construction and validation of an ontology for a large and complex domain, that
>
> 1. is usable for application development,
> 2. is reusable across different applications and
> 3. captures meta-level and tacit background knowledge.

1.3 Domain: Physical System Engineering

If ontologies are to be used as a specification mechanism for knowledge sharing and reuse across different applications (Neches, Fikes, Finin, Gruber, Senator, and Swartout 1991), they must capture the intended meaning of concepts and statements in a domain. Sharing and reuse imply two additional requirements: (i) ontologies must aim at a maximum level of genericity and thus bring out the commonalities within extensive bodies of detailed and specialized knowledge, and (ii) they must be able to explicate tacit and meta-level knowledge, as significant parts of domain expertise are highly implicit and have a background nature.

These aspects are all clearly present in the area we consider in this thesis: intelligent support for physical-system engineering. Take as a simple example the expression $F = ma$. Many people will immediately associate this with Newton's law stating that force is the product of mass and acceleration. But this is a highly non-trivial association, because it can only be made by invoking a lot of background knowledge. First, we have to know that we are dealing here with a mathematical expression, and we have to understand the related concepts of equations, parameters and variables. However, this is far from enough: the intended meaning of $F = ma$ may now still be that electrical voltage is the product of resistance and current (which, instead, many people would call Ohm's law and typically write as $V = IR$). To distinguish between such possible interpretations we need more knowledge about, for example, the concept of physical dimensions of variables. To capture the intended meaning of $F = ma$ in the context of its *use in problem solving*, we have to additionally invoke a significant body of expert knowledge. For example, we have to understand that in this context of problem-solving use, physical objects are abstracted to and parameterized in terms of a concept called `mass', that this mass acts as a kind of storage place for movement, that this movement does not change when the mass is undisturbed, and that it does change under certain external influences and circumstances which are abstracted to and parameterized in terms of a concept called `force', and so on. That is, in specifying intended meaning we unavoidably have jumped into a background body of specialist knowledge known as classical mechanics.

If ontologies are to enhance knowledge sharing and reuse by capturing intended meaning, the above-mentioned issues have to be confronted. Current information systems supporting complex tasks and domains typically do not possess the body of knowledge necessary for generating adequate interpretations, but instead rely on the fact that the user does. So, they place most of the burden on the user. Intelligent support implies that this burden must be shifted back as much as possible towards the information system. Ontologies are a promising candidate to help achieve this, but to realize this potential we need a better understanding both of their role in complex problem solving and of their construction.

1.4 Context of the Research

Our research has been carried out at Department of Computer Science of the University of Twente, Enschede, The Netherlands. This section describes the project group in which the research took place and two projects we participated in.

1.4.1 The TWIST Project Group

The TWIST project group develops concepts and methods for advanced information systems in science and engineering applications. The aim is to enhance computer support for tasks in engineering and physical science, in particular concerning the model construction process of physical systems, simulation and other forms of large-scale computational and mathematical analysis, and engineering design. A special research emphasis is on the utilization of ideas and methods from artificial intelligence for physics and engineering, and on their integration into conventional approaches and techniques in application fields.

1.4.2 The OLMECO Project

The ideas formalized in PHYSSYS (Chapter 3 of this thesis) provided the basis for the development of a library of reusable model fragments for engineering and design. This library has been developed in the European Union ESPRIT-II program OLMECO(`Open Library for Models of mEchatronic COmponents'). The program has been carried out in the period from September 1992 to November 1995.

The aim of the OLMECO project is to develop a modelling and simulation environment for industrial applications. The philosophy behind the project is that generic models fragments can be specified that can be reused in many simulation models. The reuse of these models can improve the modelling productivity and lead to a better competitive position,

The project participants provided three kinds of expertise for the project:

1. *Industrial expertise:* The project was initiated by the French car manufacturer PSA Peugeot Citroën. Further know-how about industrial applications was provided by the Spanish machine tool manufacturer Fagor. Both PSA and Fagor were assisted by technology transfer companies, respectively by Imagine (France) and Ikerlan (Spain).
2. *Methodological expertise:* The University of Twente and the Netherlands Energy Research Foundation (ECN) brought in their expertise about modelling and simulation methodology.
3. *Information technology expertise:* The Belgian software vendor BIM was involved in the project to implement the library system and to make it commercially available.

Chapter 4 describes work carried out in the OLMECO project.

1.4.3 The SUSTAIN Project

The research described in Chapter 5 has been carried out in the SUSTAIN project to support life cycle assessment. The aim of product life cycle assessment (LCA) is to determine the impacts of a product on the environment. Life cycle assessment entails the specification of a model,

comprising many different components, materials, energy, production, use and disposal processes, environmental impacts, with many interrelationships chaining all this together.

The aim of the SUSTAIN project is to make design for the environment more easy for the user. Recent advances in information technology give opportunities to make LCA software more intelligent, such that the user is better supported, needed specialist knowledge is accessed more easily, and quality LCA's become feasible for a wider range of interested parties, e.g., design engineers.

The SUSTAIN project is a cooperation between PRé Product Ecology Consultants (Amersfoort, the Netherlands), the Energy Research Foundation ECN (Petten, the Netherlands) and the University of Twente (Faculty of Computer Science, Enschede, the Netherlands). The SUSTAIN project has run from late 1995 to April 1997, and has been supported by the Dutch Ministry of Economic Affairs SENTER-IT Programme as project nr. ITU95038. The objective was to develop a number of innovative ideas regarding product and process modelling that simplify the end user's task, and can be built into the PRé SimaPro software package, the LCA software that is leading the market in Europe.

The project has produced three major deliverables, on interfacing Computer Aided Design (CAD) and LCA (Burghout, Akkermans, Borst, and Pos 1997), on new methods for product disassembly analysis (Borst and Akkermans 1997) (also Chapter 5) and on enhancements of LCA process modelling to make LCA software more intelligent and supportive (Warmer 1997). Some of the results of the SUSTAIN project have been reported at the SETAC Europe 1997 Meeting (Amsterdam, April 1997).

1.5 Organization of this Thesis

The organization of this thesis is as follows:

Chapter 2 We begin with a general chapter on ontologies. It will be explained what an ontology exactly is, what ontology specifications look like and which different types of ontologies can be distinguished. Emphasis will be put on explaining *which properties an ontology should have to be reusable*.

Chapter 3 Chapter 3 gives an overview of a general collection of ontologies for physical systems, whereby we attempt to clarify throughout how we can achieve genericity in ontological specifications, what general decomposition and structuring principles play a role, and how we can reuse existing other ontologies. We will introduce a number of *ontology construction operators* which are mechanisms to construct large and complex ontologies from smaller and reusable ontologies.

Chapter 4 Chapter 4 is more domain-specific and demonstrates the *practical relevance and use of ontologies* for demanding industrial engineering domains and tasks. We follow the complete route from formal ontology construction (PHYSSYS, Chapter 3), via the ontology-based design specifications of an implemented library of reusable models

(OLMECO, Section 4.1), to the daily task execution by domain experts (numerical system simulation, Section 4.3).

Chapter 5 Chapter 5 demonstrates how, using the principles of ontology construction, some of the ontologies that form PHYSSYS can be *reused and extended* with new ontologies to capture knowledge *for a totally different task*, namely that of ecological product disassembly. This chapter also describes the design of a prototype knowledge-based system, demonstrating in more detail the use of ontologies as software specifications.

Chapter 6 We believe that many of our experiences and results are independent of the considered domain, and have a general relevance for the engineering of ontologies. In Chapter 6 we will discuss the conclusions emerging from the present work and the role ontologies can play in the future.

Appendix A This appendix contains the complete Ontolingua 4.0 implementation of the PHYSSYS library of ontologies.

Appendix B This appendix contains a specification of the thermodynamic models we developed for the OLMECO library as well as a detailed description of the modelling experiment.

1.6 Publications

Parts of this work have been published and presented in international fora:

An article in the International Journal of Human–Computer Studies (Borst, Akkermans, and Top 1997) reported on the construction and validation of PHYSSYSas described in Chapters 3 and 4. A shorter version has been presented at the Bannf Knowledge Acquisition for Knowledge-Based Systems Workshop (Borst, Akkermans, and Top 1996).

Earlier versions of he PHYSSYS ontology (Chapter 3) have been presented at workshops of the European Conference on Artificial Intelligence (ECAI) in 1994 (Borst, Pos, Top, and Akkermans 1994) and the International Workshop on Qualitative Reasoning (Borst, Akkermans, Pos, and Top 1995).

Some ideas of ontology construction were addressed in (Benjamin, Borst, Akkermans, and Wielinga 1996) and have been presented at the European Knowledge Acquisition Workshop in 1996. The paper describes the way an ontology for heat exchangers could be constructed. The same topic, only then applied to the PHYSSYS ontology is covered in (Borst, Benjamin, Wielinga, and Akkermans 1996a) (presented at the ECAI'96 workshop on ontological engineering) and (Borst, Benjamin, Wielinga, and Akkermans 1996b) (presented at the Dutch Conference on Artificial Intelligence).

Our work in the SUSTAIN project on ecological product disassembly (Chapter 5) has also been described in a project deliverable (Borst and Akkermans 1997). A presentation about the project has been given at the Seventh Annual Meeting of the Society of Environmental

Toxicology and Chemistry in 1997. The project has also been presented at the Workshop on Product Knowledge Sharing and Integration on April 17–18, 1997 in Sophia Antipolis.

The conceptual schemata of the OLMECO library were published in (Akkermans, Borst, Pos, and Top 1995). The thermodynamic models we developed for the OLMECO library can be found in the OLMECO deliverable (Top, Borst, and Akkermans 1995) (see also Appendix B). Our experiences in the design and the use of the thermodynamic library have been described in (Borst, Top, and Akkermans 1997) and were presented at the International Conference on Bond Graph Modeling and Simulation in 1997.

On November 19, 1996, the Dutch national newspaper *NRC Handelsblad* published a photograph that also appears on the cover of this thesis (although with a different background). The picture shows the Peugeot 106 Electrique. This electrical car has been designed using the OLMECO library. At the final project review, a prototype of the car was demonstrated by engineers of Peugeot to the European Community representatives, referees and project participants who could take rides in it. A few months later, the car was introduced on the market and bought by the police in Kidderminster.

Chapter 2

What is a Useful Ontology?

In this chapter we will explain what ontologies exactly are, how they can be specified and how they can be used. We will see that different kinds of knowledge can be distinguished and that knowledge can be modularized in small, manageable pieces. This makes it possible to construct large and complex ontologies out of smaller and more reusable ones.

In Section 2.1 of this chapter will be explained what is meant by the term `ontology' by giving a definition. The following sections will describe how a very simple ontology of the classification of animal species in herbivores, carnivores and omnivores can be developed. With this example ontology we will show how ontologies can be specified (Section 2.2) and which different kinds of ontologies can be distinguished (Section 2.3). Section 2.4 explains how large ontologies can be constructed from smaller ontologies. The way ontologies can be used is discussed in Section 2.5 and Section 2.6 gives a summary of this chapter.

2.1 A Definition of Ontology

In philosophy, the word ontology means a theory about the nature of being, or the kinds of existence. Artificial intelligence (AI) has borrowed the word from philosophy and has given its meaning a twist. For AI the main question is not what the nature of being is, but what an AI system has to reason about to be able to perform a useful task. An often used and paraphrased definition of ontology is that of Gruber.

> An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented (Gruber 1994).

In order to understand this definition, it must be clear what a conceptualization is. A conceptualization is a structured interpretation of a part of the world that people use to think and communicate about the world. For a biologist such a conceptualization may include that animals can be classified in groups called species and that the animals belonging to a species have similar eating habits. Based on these eating habits the species can be subcategorized into herbivores, carnivores and omnivores.

The AI definition of ontology resembles the interpretation of the philosopher Quine (Quine 1961): what exists is that what can be quantified over. Just as in the field of AI, he is more interested in the concepts required for useful reasoning in a domain, and not so much in the question whether a concept exists in the physical world or not. The question whether the animal species of elephants exists in reality or only in the mind of people is an interesting question, but for the practical use of ontologies less important.

A lot of debate has been going about what is the best definition of an ontology. Most researchers generally agree on the definition of Gruber, but find it too broad. The main differences between the definitions lie in ways that ontologies that satisfy Gruber's definition, but are not useful for application development are excluded. The debate is similar to that of the difference between data and information. Although it seems to be impossible to define the exact difference between the two, computer engineers are still able to write successful information systems. We will therefore give a definition of ontologies that suites us the best and continue this section with explaining how to make a good ontology.

> An ontology is a formal specification of a shared conceptualization.

This definition emphasizes the fact that there must be agreement on the conceptualization that is specified. The reason for including this is that the ability to reuse an ontology will be almost nil when the conceptualization it specifies is not generally accepted.

2.2 How to Specify an Ontology?

For reuse of ontologies it is essential that they are formally specified. There are many formalisms that can be used. Examples are first-order predicate logic (Hayes 1985), MODEL (Tu, Erikson, Genari, Shahar, and Musen 1995) and CML (Schreiber, Wielinga, Akkermans, van de Velde, and Anjewierden 1994). Most specification languages are based on predicate logic or meta logic. The ontologies described in this thesis have been specified in the language of the Ontolingua system (Gruber 1992; Gruber 1993; Farquar, Fikes, and Rice 1996).

The syntax of Ontolingua definitions is based on a standard notation and semantics for predicate calculus called Knowledge Interchange Format (KIF) (Genesereth and Fikes 1992). KIF is a monotonic first order logic that has been slightly extended to support reasoning about relations. KIF is intended as a language for the publication and communication of knowledge.

The current version of the Ontolingua system is accessible only through the World Wide Web (http://ontolingua.stanford.edu). The system supports browsing, construction and modifica-

tion of ontologies by distributed groups of people. The system can check the consistency of definitions and the use of the terms in an ontology. Ontologies can be presented in a structured way and stable ontologies can be made public by submitting them to a library. Furthermore, ontologies can be translated to various knowledge representation languages such as IDL (Mowbray and Zahavi 1995), Prolog, CLIPS, LOOM (MacGregor 1990), Epikit (Genesereth 1990) and KIF.

In Ontolingua terms, an ontology is called a *theory*, a term borrowed from logic. A theory consists of definitions of classes, relations, functions, class instances and axioms. Furthermore, a standard Ontolingua theory called the *Frame Ontology* defines concepts like frames, slots, and slot constraints that enable ontological engineers to express knowledge in object-oriented and frame-language terms.

Throughout this thesis excerpts of our ontologies will be presented. The excerpts are written using a slightly altered syntax. The basic structure and the types of the Ontolingua definitions has been kept, but the syntax has been changed to conform closer to the syntax of predicate logic. This to improve readability for people unfamiliar with Ontolingua. The ontologies have been parsed and checked by version 4.0 of the Ontolingua system. Version 4.0 is a previous version of Ontolingua that was still a stand alone system without Internet capabilities.

2.3 What is in a Useful Ontology?

Figure 2.2 gives an example of (a first attempt to) an ontology of animal forage. Line 1 defines the name of the theory that is specified. Usually this name gives an indication of the domain the theory is about. Line 2 defines the existence of a concept called *species* in the domain of animal forage. On the lines below, marked *a* to *c*, *axioms* regarding the class of species and its instances are defined. Axioms are definitions of things that are always true in the domain. Axioms 2a to 2c define that ferrets, polecats and chamois [1] are instances of the class species. In Line 3 the relation eats is defined. It associates a type of food to animal species. The axioms in Line 3a to 3c define properties of the relation that are always true. The knowledge that a species are herbivorous, carnivorous or omnivorous is formalized in Line 4.

Based on the forage ontology, a small computer program can be written that can tell whether ferrets, polecats and chamois are herbivores, carnivores or omnivores. The ontology provides the information to design the data structures of the application. For applications written in a programming language like C or Pascal, an array of records with fields for the name of the species, the kind of food and the kind of species suffices. For database applications, a table with these three fields can be used.

An important observation about the ontology is that it defines terms in the domain and relationships between these terms in a formal way, but it does not specify the meaning of the

[1] The species used in the example are species that frequently appeared in the hilarious sketches about a pet shop in the Dutch TV programme *Jiskefet* (which means trashcan in the Frisian language).

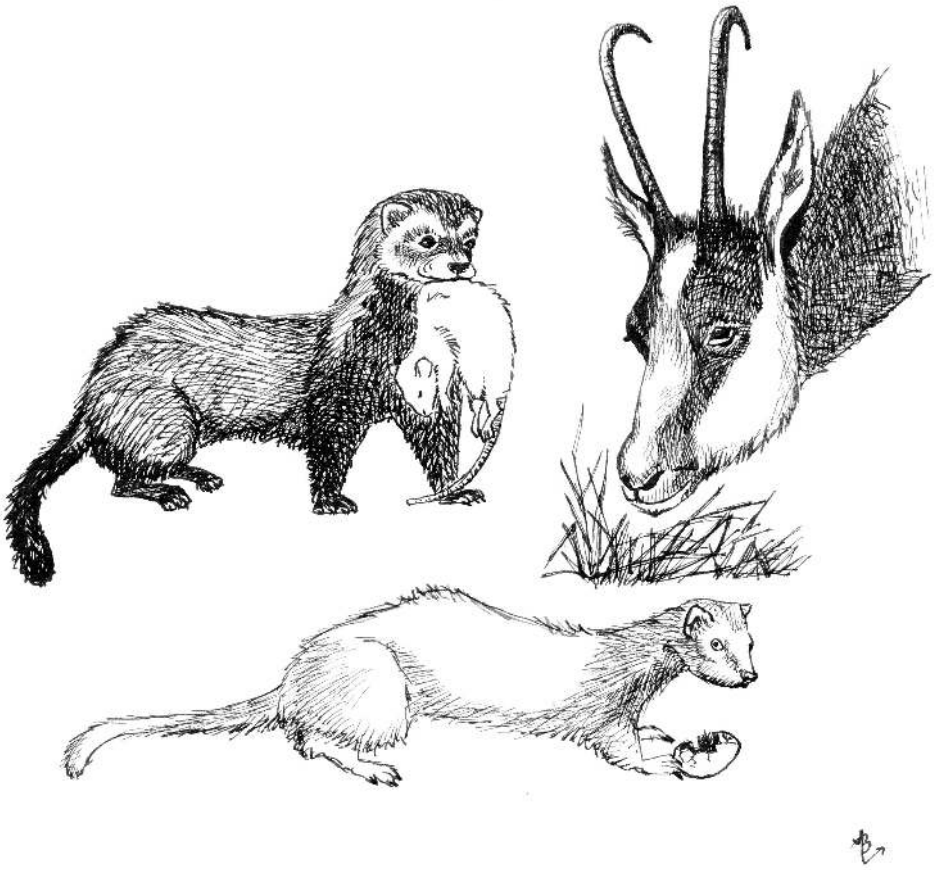


Figure 2.1: The species appearing in the ontology of animal forage: Polecats (top left-hand side), Chamois (top right-hand side) and Ferrets (bottom). The drawing has been made by Annemarie Borst.

```
1  define-theory animal-forage

2  define-class species(x)
a    instance-of(ferrets,species)
b    instance-of(polecats,species)
c    instance-of(chamois,species)

3  define-relation eat(s,f)
a    eat(ferrets,animals)
b    eat(polecats,animals)
c    eat(chamois,plants)

4  define-relation kind-of(s,f)
a    kind-of(ferrets,carnivorous-species)
b    kind-of(polecats,carnivorous-species)
c    kind-of(chamois,herbivorous-species)
```

Figure 2.2: First attempt to an ontology of animal forage.

terms. What species, ferrets or herbivorous species actually are, is assumed to be known to the group of persons for which the ontology was written (biologists). An ontology therefore is a formal specification of *a part of* a conceptualization. The names of the concepts and the description of the ontology in natural language is therefore important for the ability to understand and use an ontology.

If an agent, information system or group of people agrees on an ontology, it is said that they commit themselves to it. The choice of terms and axioms are therefore called the *ontological commitments*. There seems to be a general agreement (Gruber 1995) that the ontological commitments in an ontology need to be kept to a minimum, because overcommitment reduces the reusability of an ontology. We would rather say that there is a tension between under, and overcommitment. Overcommitment reduces the *reusability*, but undercommitment reduces the *usability* of an ontology. In this view, the forage ontology of Figure 2.2 is a good example of a useless ontology. In one sense it is overcommitted and in another it is undercommitted. This will become clear below when we look at the imaginary application that is based on this ontology.

The functionality of the application is limited because in committing to the ontology, it committed to three animal species. As a result, when the user asks about the forage habits of rhinoceroses the system is unable to answer. The ontology should therefore not mention the factual situation (also called *state of affairs*) in a domain, but describe the meta-knowledge in the domain. For the forage ontology this means that it should specify the relationship between the eating habits of a species and the type of species. In failing to do this, we could say that the ontology is also undercommitted. Although this remark on undercommitment may seem obvious, many of the ontologies found in the literature are just classifications of domain instances, just as the forage ontology in Figure 2.2. Incorporating meta-knowledge in the forage ontology leads to the ontology in Figure 2.3.

In the new ontology, meta-knowledge about the domain is specified instead of the state of affairs. The fact that animals of a certain species can eat one or two types of food is specified in Line 3. Depending on the type of food the species eats, it can be decided what kind of

```
1  define-theory animal-forage

2  define-class species(x)

3  define-relation eat(s,f)
a    forall s,f: eat(s,f) -> instance-of(s,species)
       and (f=plants or f=animals)

4  define-relation kind-of(s,f)
a    kind-of(s,herbivorous) <->
       instance-of(s,species) and eat(s,plants) and not eat(s,animals)
b    kind-of(s,carnivorous) <->
       instance-of(s,species) and eat(s,animals) and not eat(s,plants)
c    kind-of(s,omnivorous) <->
       instance-of(s,species) and eat(s,plants) and eat(s,animals)
```

Figure 2.3: Second attempt to an ontology of animal forage.

animal species it is. This is formalized in Line 4. Note that in Lines 4a to 4b, the free variable `s` is unbound. Where there are unbound variables in an axiom, it is assumed that they are all universally quantified, just as in Line 3.

It is of course possible to incorporate some domain facts in a domain ontology, but generally this is only done when there is good reason to believe that the facts are exhaustive. This is the case with the types of food because there are only two types: plants and animals. For non exhaustive facts, it is left open to the information system developers to decide which facts to include in the system. Because the specifications in Figure 2.2 is not much more than just a specification of domain facts, some researchers would not consider it an ontology at all.

Because the ontology now includes meta-knowledge, its (re)usability has been increased. It can not only be used for simple programs that know about three animal species, but also for database systems where information about new species can be added and checked for consistency (using meta-data derived from the ontology). Also, a more intelligent knowledge based system can be built that, when it is queried about a species it has not heard of, can ask the user whether he knows the type of food this species eats. Based on that information the system can classify the species and update its factual knowledge.

The structure of an ontology can be presented in an OMT conceptual schema (Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen 1991). Figure 2.4 gives an overview of the OMT notation. The conceptual schema of the forage ontology can be found in Figure 2.5. Such a schema gives an overview of the concepts and relations in an ontology. Conceptual schemata cannot replace an ontology, because their expressiveness is too limited. For instance, the domain knowledge that species that only eat plants are herbivorous (Axiom 4a–c) cannot be represented. In cases where relations between concepts are restricted by important axioms, we will represent this using a labelled dotted line between the relations. The labels correspond to the relevant axioms.

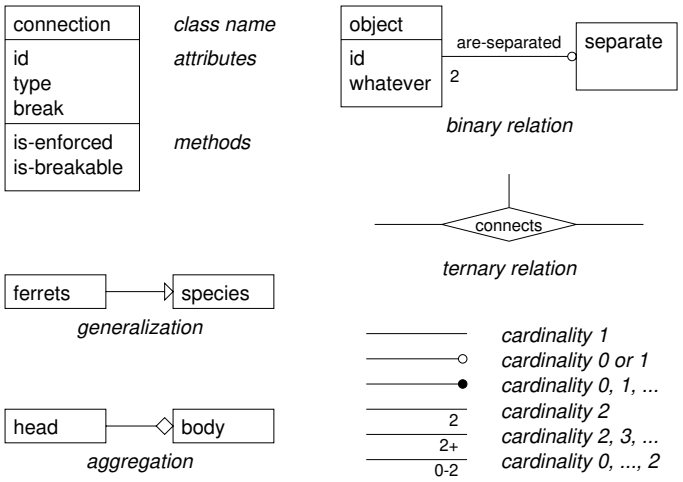


Figure 2.4: Notational conventions of OMT.

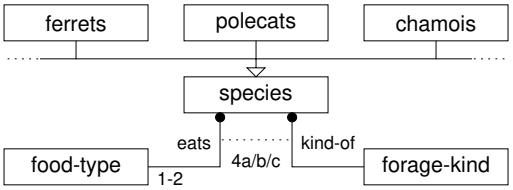


Figure 2.5: Conceptual schema of the forage ontology extended with three animal species.

2.4 The Problem: Useful Ontologies will be Big

It is evident that knowledge sharing and reuse is most profitable for applications in large and complex domains. It can therefore be expected usable ontologies will be also large and complex. The approach to handle such large ontologies is similar to the approach to handle large pieces of software in software engineering: divide and conquer. Domain knowledge is divided into small, manageable pieces with strong internal coherence but relatively loose coupling. These small bodies of knowledge are specified in separate ontologies that form the building blocks to construct larger ontologies. In this section we will explain several ways to modularize large bodies of knowledge and the way ontology construction takes place.

2.4.1 Divide: The Ontological Building Blocks

Division of knowledge into small pieces is based on recognition of different kinds of ontologies. We will briefly describe these different kinds, the properties they have and the way they can be identified in large bodies of knowledge.

Natural Viewpoints The first way to modularize domain knowledge is to partition it in pieces in which the concepts are centered around natural viewpoints or base categories. This results in relatively small *viewpoint* ontologies. For instance, the knowledge in the domain of biology can be partitioned in the subdomains of botany and zoology. Both subdomains can be combined in the general domain ontology of biology. This ontology could then define a taxonomy of all organisms, botanical and zoological on top of it. Other examples of viewpoints are the ones distinguished in Chapter 3 of this thesis. There, three viewpoints on technical devices are distinguished: the way they are constructed from components, their behaviour in terms of physical processes and the way this behaviour can be described mathematically. The advantage of modularization into viewpoints is that a viewpoint has less ontological commitments and is therefore more reusable than the entire domain ontology.

Abstract Ontologies A second way for modularization is to make ontologies that define abstract concepts. These abstract abstract concepts can be used to define more specific concepts in different domains. An example of abstract concepts are taxonomies. A taxonomy is a structured overview of classes, subclasses and instances. Taxonomies can not only be found in biology, but also in engineering domains. Chapter 4, for instance, describes a library of reusable simulation models in which the component models can be accessed using a component taxonomy. Other examples of abstract ontologies are mereology (part-of relationships), topology (connected-to relationship), and systems theory which are discussed in Chapter 3. Ontologies of graph-theory and of state-space are briefly described in Chapter 5.

Method Ontologies The third type of ontologies are *method ontologies*. Method ontologies define the way domain knowledge can be used to perform certain tasks. For the forage on-

tology we have already seen that there are various ways in which domain knowledge (facts) can be used to classify an animal species. In one application, information in a database was searched for. In another application, facts (the type of food a species eats) were acquired from the user and reasoned with to infer the kind of species. Both for searching and for logical reasoning general problem solving methods exist. For searching, there are for example sequential and binary search methods. Examples In the field of logical reasoning are inductive and abductive reasoning. These methods are general in the sense that they are independent of the domain that is reasoned about.

A method ontology includes the definition of a terminology for expressing the competence and the knowledge requirements of a method (Fensel, Schönegge, Groenboom, and Wielinga 1996). For search methods this terminology could include the concept of *records*, which are group of related domain facts, *keys* on which all facts in a record uniquely depend and *queries* that define the records we are interested in. The two search methods that have a different competence. Sequential search generally takes more time than binary search, but the latter has an additional knowledge requirement. For binary search there must be an ordering relation defined for the keys, and the records must be sorted accordingly. The way method ontologies can be linked to domain ontologies is explained in (Gennari, Tu, Rothenfluh, and Musen 1994) and in Chapter 5 of this thesis.

Application Ontologies Ontologies that are used for the design of an application are called *application ontologies*. Generally, they consist of a domain ontology and methods from a method ontology. The methods specify the way the functionality of the application is achieved. Therefore, application ontologies are less suited for reuse for other applications.

The degree to which knowledge can be formalized in reusable ontologies is strongly affected by the interaction problem, presented by Bylander and Chandrasekaran (1988):

> Representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem (Bylander and Chandrasekaran 1988).

Interpreting this problem in a pessimistic way, one could say that it is hardly possible to write a domain ontology that can be reused across many applications because each application will have different tasks and use different methods to accomplish the tasks.

We argue however, that although it is not feasible nor desirable to write *one* domain ontology that is appropriate for all tasks in a domain, it is possible to write a domain ontology that can be shared across large groups of applications. Again, the key factor is modularity. Domain knowledge that is shared by all applications should be specified in a separate core ontology. Task dependent extensions to, or axioms restricting this core knowledge can then be specified in additional ontologies. This provides us another principle for modularization: separate knowledge that is specific for a small group of tasks from knowledge that is less task specific.

2.4.2 Conquer: Ontology Construction

Modularizing domain knowledge in manageable ontologies is one part of the solution, but in order to assemble them into large and complex domain ontologies, a mechanism to combine ontologies is required. We call this assembling of small ontologies *ontology construction*.

In the Ontolingua system version 4.0, ontology construction is supported by ontology inclusion. Inclusion of an ontology in another means that definitions of the first theory are added to the second. The current version of Ontolingua offers additional facilities to resolve naming conflicts when an ontology that defines a term includes an ontology that defines a conceptually different term with the same name. Furthermore, there is a mechanism to handle polymorfism. This allows existing operators to be extended to work on new classes (for instance the + operator means addition for numbers and concatenation for strings).

What we think is a disadvantage of the present inclusion mechanism, is that it is a pure syntactical operation. If large ontologies are constructed by combining smaller ontologies, the inclusion operator should make explicit the relationship between the combined knowledge: are different viewpoints combined, are abstract concepts used to define more specific ones or is the knowledge merely extended? This kind of informative mechanisms would help to understand large ontologies. This is important, because in order to reuse an ontology, the contents of it must be understood by the ontological engineer.

2.4.3 Libraries of Ontologies

To deal with large ontologies, large and complex domain ontologies are modularized in smaller, manageable ontologies of different kinds. Many of these ontologies will be reusable for different applications.

Domain ontologies can be reused because they define domain knowledge that can be used for a large group of applications and problem solving methods.

Domain viewpoints can be candidates for reuse because they define compact and coherent pieces of knowledge in a domain. Because subdomains generally consist of a subset of the views of an entire domain, it is likely they can be shared across many subdomains.

Abstract ontologies are especially suited for reuse. Many domain concepts can be defined as specializations of abstract concepts. The domain concepts will then comply to the same properties as the abstract concepts.

Method ontologies can be reused because they define generic problem-solving methods that can applied in different domains.

To allow that the above mentioned ontologies can be reused by different people they need to be shared. A common way to share knowledge is by using a library. The Ontolingua system supports such a publicly accessible library of ontologies. Ontological engineers can submit

ontologies to this library or use ontologies from the library for ontology construction. To access the ontologies in the library, the library system offers different kinds of information such as an inclusion lattice of the ontologies in the library. Furthermore, automatically generated reports of ontologies can be inspected. These reports give an overview of the terms defined in the ontology as well as the terms included from other ontologies.

One aspect in which ontology libraries need to be improved is their support to retrieve an appropriate ontology in the library. Although the library system presents some information about the ontologies, we think this is not enough to support reuse sufficiently. The relationships between the contents of the ontologies have to be presented. Differences between alternative ontologies for a domain or domain view have to be made explicit. For method ontologies, similarities and differences between the problems they solve should be visible. Abstract and method ontologies should give indications in which domains they can be of use. With this information an appropriate ontology can be retrieved without having to study the definitions in every ontology in detail.

2.5 The Role of Ontologies in Information Systems Development

The usual way ontologies are used in information systems development is by construction of an application ontology. This application ontology then serves as a specification of the application's data structures and the inference engine that performs the application's tasks.

In the first part of this section we will describe the design process in which the application ontology is constructed. The second part gives a comparison between ontology-based development and other approaches in (non-AI) computer science.

2.5.1 In Artificial Intelligence

Most of the current approaches (for instance PROTÉGÉ-II (Gennari, Tu, Rothenfluh, and Musen 1994)) to construct an application ontology include the following steps:

1. construct the domain ontology
2. determine which problem solving methods perform the application's tasks
3. relate the knowledge required by the selected methods to domain knowledge
4. add factual knowledge about the domain

Some steps can be executed concurrently by different persons, for instance Step 1 and 2. Step 1, 2, and 3 may need to be iterated for the domain ontology to converge to the selected methods. The addition of domain knowledge can sometimes be supported by knowledge

acquisition tools based on the domain ontology constructed in Step 1. The result is the application ontology for the information system.

In (van Heijst, Schreiber, and Wielinga 1997), Van Heijst discusses the problem of knowledge-based integration of representation formalisms. The off the shelf problem solvers that are used to implement the application often require different knowledge representation formalisms. The solution he proposes is to use the application ontology as a specification of a common knowledge representation. Inferences of problem solvers using other representations are translated to and from this common representation. As we will see in Chapter 5, the same solution has been used for the development of a prototype knowledge based system. There, we used functions that translate data from C data structures to the form required for the problem solving methods.

2.5.2 In Information Systems Development and Object Orientation

The role of application ontologies for information systems design does not differ much from more conventional system design principles in software design, such as semantic modelling (entity/relationship diagrams and object models), meta-data, design and analysis patterns and the use of libraries of reusable software modules. We will discuss the main differences between these approaches and the ontological approach and explain the added value of the latter.

Reusable Software Modules Libraries of problem-solving methods are very similar to libraries of reusable software modules. Both contain pieces of software that can be used for different tasks in different domains. Method ontologies serve as a specification for the competence and knowledge requirements of the methods in these libraries. Therefore, the application ontology constructed in the ontological approach makes explicit how the domain tasks are related to the generic task performed and which domain knowledge is used.

Semantic Modelling An entity/relationship diagram is a conceptual specification of the data in an information system. It defines the concepts that are distinguished, the properties they can have and the relations between entities. Furthermore, a class–subclass hierarchy between entities can be defined which causes inheritance of properties and relations. The Ontolingua frame ontology is a formal definition of this way of specification of knowledge. Therefore, every entity/relationship diagram can be captured in an ontology. Ontologies however have the ability to specify domain knowledge in a much richer and structured way.

Object models can be viewed upon as entity/relationship diagrams with an extension that attaches methods to objects (inheritance also applies to these methods). In the ontological approach, these methods specified in method ontologies can also be related to domain objects. Application ontologies can therefore be used to express object oriented models (Takagaki and Wand 1991).

Meta-Data In database systems development, specifications of the properties of objects and relations is called *meta-data*. Meta-data can be used to check the consistency of the data in the system. Because domain ontologies are developed to capture this meta-knowledge, meta-data can be easily derived from a domain ontology. The ontological approach has the benefit that the principles of ontology construction can be used to define meta-knowledge in a structured way.

Object Oriented Design and Analysis Patterns A very interesting development in the field of object oriented analysis and design is the use of patterns (Coad 1992; Gamma, Helm, Johnson, and Vlissides 1995; Partridge 1996; Fowler 1997). Analysis patterns are meta-level descriptions of pieces of object models that frequently occur in object oriented analysis. Design patterns are similar, but have pieces of code attached to it, so they can be used for object oriented design. Analysis patterns are similar to domain ontologies and design patterns to application ontologies. The benefit of using ontologies instead of patterns is that ontology construction offers a way for constructing complex ontologies from smaller ones. Furthermore, using abstract ontologies it is possible to specify knowledge at higher knowledge levels. In pattern terminology this means that patterns of patterns can be specified.

The great benefit of design patterns is that the methods defined for them are tuned to the object oriented knowledge representation. Thus, the problem of knowledge representation integration is got around. Application development using design patterns is therefore less time consuming than ontology-based development. On the other hand, ontologies are not restricted to object oriented knowledge and methods. We think that the fields of object orientation and ontological engineering can learn a lot from one another's achievements.

Analysis and design patterns are collected in libraries. Like ontology libraries, these pattern libraries also lack good support for accessing the contents of large libraries.

2.6 Ontologies as Information System Specifications

We conclude this chapter by summarizing the most important findings.

Ontologies are formal specifications of shared conceptualizations. They can be the instruments for knowledge sharing and reuse. Ontologies can support information systems design because they specify the knowledge an information system must capture to perform its tasks. To be (re)usable, ontologies should not capture facts about instances in the domain, but make explicit tacit and meta-level knowledge. Ontologies are usually specified in representation languages based on predicate logic.

Ontologies can be constructed from smaller modules that are ontologies themselves. Different types of these smaller ontologies can be distinguished: domain ontologies, domain viewpoint ontologies, abstract ontologies and method ontologies. Many of these ontologies can be reused across different domains and applications. A good way to support sharing and reuse of ontologies is to include them in a publicly accessible ontology library.

By combining a domain ontology and different method ontologies and adding domain facts, an application ontology can be constructed. An application ontology specifies how the application's functionality can be implemented and which domain knowledge is required. As such, application ontologies play a similar role in information systems design as entity-relationship diagrams, object models, and object patterns. The benefit of ontologies is that they are better suited for piecemeal specification of complex structured domain knowledge, that they make explicit the way domain tasks are performed and that they are independent of the application's implementation language.

In the next chapters we will leave the animals in this chapter in peace and start the investigation of the use of ontologies for complex applications in a large, technical domain.

Chapter 3

Engineering Ontologies

In this chapter we will discuss the way a domain ontology for the modeling of physical systems can be constructed from smaller, reusable ontologies. The ontology is called PHYSSYS *and describes the domain knowledge required to make simulation models for devices like heating systems, automotive systems and machine tools.*

In Section 3.1 we will see that the knowledge about physical systems can be centered around three conceptual viewpoints: technical components, physical processes and mathematical relations. Furthermore, generic abstract relations that play a role in both component and process viewpoints can be distinguished. This allows us to modularize the ontology and construct it out of small, reusable ontologies using *ontology projections*. Section 3.2 gives an extensive comparison of the work presented in this chapter and related research. We will end this chapter with an overview of our findings about ontology construction and ontology projections (Section 3.3).

3.1 The PHYSSYS Ontology for Physical Systems

PHYSSYS is a formal ontology based upon system dynamics theory as practiced in engineering modelling, simulation and design. It forms the basis for the OLMECO library, a model component library for physical systems like heating systems, automotive systems and machine tools. The ontology expresses different conceptual viewpoints on a physical system. To demonstrate what these viewpoints are, we carry out the small exercise of determining the knowledge that is required to understand the formula $F = ma$. Anybody who paid attention during physics class at highschool knows that this formula is Newton's law that describes the acceleration of an object under the influence of a force. Unfortunately, for a computer this is not obvious at all.

When a user types in the formula on the console, it is just a string of characters. Assuming the computer knows about mathematics, this string will be parsed and identified as a mathematical formula, a relation between variables. The mathematical knowledge must include the facts that a variable stands for a certain value that may or may not change in time (or another free variable) and that possibly has a certain dimension. Knowing all this, $F = ma$ just means that the value of one variable is equal to the product of the value of two other variables, at any time. The system still knows nothing about its meaning in terms of physics.

In order to make the computer understand the physical implication of the formula it must know about physical processes, energy and physical domains. It must know that a formula can be a mathematical description of a physical process like the inertial effect of a mass. At this point it also becomes clear that a mathematical variable represents a certain physical quantity. In the $F = ma$ example, F is a force quantity, m stands for a mass and a for an acceleration. Without this knowledge $F = ma$ could also have meant that the voltage F is equal to a resistance m multiplied by the electrical current a. With this interpretation, the equation would have been another famous physical law called Ohm's law (usually written down as $V = IR$) that describes the process of electrical resistance.

The final step is to introduce the relation between the physical processes and the real world physical system. For this, knowledge of how people look upon physical systems is required. In engineering it is customary to think of the system as a configuration of components which on their turn can be decomposed into smaller components. Connections between components are the means for interaction. In these terms, the mass could be a heavy object hoisted by a crane. The load and cable would be two components connected to each other by a mechanical connection. Each component is the carrier of physical processes. The load component is the carrier of the inertial effect and its interaction with the cable component implies an energy flow between the physical processes modelling the two components.

Accordingly, it is clear that three conceptual viewpoints on physical systems can be distinguished: (i) system layout, (ii) physical processes underlying behaviour and (iii) descriptive mathematical relations. This can be seen in Figure 3.1, which gives an overview of the structure of the PHYSSYS ontology (Borst, Pos, Top, and Akkermans 1994; Borst, Akkermans, Pos, and Top 1995). Boxes represent separate ontologies whereas labeled arrows indicate ontology inclusion. The labels next to the arrows show the kind of inclusion. As can be seen in the figure, the basis of PHYSSYS is formed by three primary ontologies which are formalizations of the three views on the physical domain.

What can also be noticed is the use of the abstract ontologies *mereology*, *topology* and *systems theory* in the construction of the component and process views. One particular viewpoint on a physical system is that it is a *system* in the sense of general systems theory. That is, it constitutes an entity that (i) can be seen as separate from the rest of the world —so it has a boundary and an outer world, the environment— and that (ii) has internal structure in terms of constitutive elements and subsystems maintaining certain mutual relationships. Clearly, this system theoretic view can be applied for device components for the entire concept of components is based on the fact that they are separate entities that can be connected to form a device. In Section 3.1.5 we will see that systems theory can also be applied for the definition of physical process descriptions.

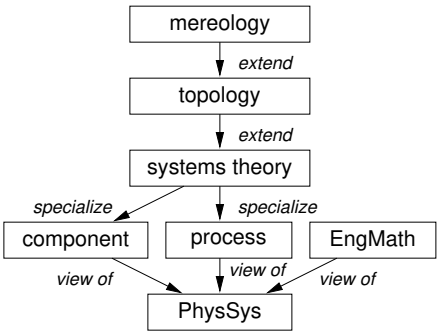


Figure 3.1: Inclusion lattice of the PHYSSYS ontology.

For physical systems this implies that we focus on the *structural* aspects, and abstract from what kind of dynamic processes occur in the system and from how it is described in terms of mathematical constraint equations. Within such a purely structural view, we can express the following knowledge about the system:

Mereological relationships: a system has a certain *part-of decomposition* into subsystems, which on their turn can be decomposed into more primitive components.

Topological relationships: the various constituents of a system (subsystems, components) are linked to one another through certain *connections*. For a physical system, this usually provides information on the spatial topology of the system, but in general, the connections indicate the paths for physical interactions between the constituents.

System theoretic concepts: with use of the mereological and topological relationships system theoretic concepts such as *system boundary*, *system environment* etc. can be defined.]

The PHYSSYS ontology consists of the three engineering ontologies formalizing the three viewpoints on physical devices. These viewpoints themselves are constructed from smaller abstract ontologies. The interdependencies between these ontologies are formalized as ontology projections. This gives us a set of ontologies of varying genericity and abstractness. Identifying these separate ontologies not only makes it easier to understand the domain because classes and ontological commitments are added incrementally, it also increases the ability to share and reuse parts of PHYSSYS.

In the next sections these ontologies will be presented. The three abstract ontologies will be described first because they are used in the construction of the component and process views. Special attention will be given to the *ontology projections*, which are the formalizations of the interdependencies between included ontologies.

3.1.1 Mereological Ontology

This section describes the mereological ontology that defines the part-of relation and its properties. The mereological ontology of PHYSSYS is simply an Ontolingua implementation of the Classical Extensional Mereology as described in (Simons 1987). In at the end of this section the Ontolingua Implementation will be presented, but first we will discuss Classical Extensional Mereology in detail to show what an abstract ontology is and why they are difficult to develop.

Classical Extensional Mereology

Mereology literally means *`science or theory of parts'* and stems from the Greek word for *part*, $\mu\epsilon\rho o\varsigma$. It is also the name of the formal theory of parts and associated concepts developed by Leśniewski. To distinguish his theory from other mereological theories, the theory of Leśniewski is referred to as `Mereology' written with a capital M.

At first glance, the part-of relation does not seem to be hard to define. A component for instance is part of a another component when the smaller component has been used to built the larger. Mereology claims to be not only a theory about the part-of relationship between components, but rather between a large class of `things' for which parts and wholes are relevant. Some examples of these things, which we will call *individuals* from now on, can be found in Table 3.1. It is only when we study in detail what may be concluded from the statement that individual A is part of individual B and what this means for the part-of relationships between A, B and other individuals that the difficult aspects of Mereology become appearant.

whole	**parts**
body	organs
organism	cells
device	components
house	roof, walls
book	chapters

Table 3.1: Examples of part-whole relationships.

In this section we will investigate these aspects by presenting Leśniewski's Classical Extensional Mereology in the way it is explained in (Simons 1987). We will see that we have to consider aspects of the theory that may be irrelevant in the context of components but are essential for choosing the right ontological commitments. This will give us some clues of how a library of ontologies needs to be organized.

The logical notation used in this section and Section 3.1.2 corresponds to the notation used in (Simons 1987) and can be found in Table 3.2. Furthermore, we have to mention that it is assumed that unbound variables are universally quantified. The survey of Mereology in this section and that of the mereo-topology in Section 3.1.2 is based on the work of Simons.

symbol	meaning	symbol	meaning
$\supset$	logical implication	$\circ$	overlap
$\equiv$	logical equivalence	$\mid$	disjointness
$\wedge$	logical and	$+$	binary sum
$\vee$	logical or	$-$	mereological difference
$\sim$	logical negation	$\cdot$	binary product
$\exists$	existential quantifier	ι	definite description
$\forall$	universal quantifier	σ	general sum
$\ulcorner \ldots \urcorner$	scope of quantifiers	π	general product
$=$	equality	$\asymp$	connectedness
$\approx$	identity	$]\![$	disconnectedness
$<$	part-of	$\nsim$	external connectedness
$\ll$	proper part-of		

Table 3.2: Logical notation for Mereology and Clarke's mereo-topology.

Part-of, Proper-part-of and Equality Like in mathematics, where the usage of the relation $\leq$ leads to shorter expressions than the combined use of $<$ and $=$, in the formalization of the properties of the part-of relation it is often convenient to express that the logical variable x signifies a single individual that is part-of or equal to the single individual signified by y. For this reason, there are two part-of relations in Mereology, one meaning part-of-or-equal and the other meaning a-real-part of. The relation for a-real-part-of is called the *proper part-of* relation ($\ll$) and the relation for part-of-or-equal somewhat confusingly *part-of* ($<$). The relation between the three relations is expressed by the following definition of part-of in terms of the more intuitive proper-part-of and equal relations.

$$x < y \equiv x \ll y \vee x = y$$

Asymmetry and Transitivity In Mereology, axioms define the properties the part-of relation must have to exclude situations that do not make sense. These properties include asymmetry and transitivity:

$$x \ll y \supset\sim y \ll x$$

$$x \ll y \wedge y \ll z \supset x \ll z$$

Asymmetry makes it impossible to say that an individual is a proper part of itself. Transitivity states that when an individual is a proper part of a second individual that is a proper part of a third individual, the first is also a proper part of the third. Although transitivity is a sensible property when we regard a whole as being `assembled' from its parts, it can sound a bit strange in cases where the parts are named after their function. Take for instance the sentence: The house has a door, the door has a handle, so the house has a handle. This different view on the part-of relation is discussed in (Winston, Chaffin, and Herrmann 1987) and (Gerstl and Pribbenow 1995).

Overlap and Disjointness We define overlapping ($\circ$) as "sharing a common part" and disjointness (l) as the logical negation of this:

$$x \circ y \equiv \exists z^{\ulcorner} z < x \wedge z < y^{\urcorner}$$

$$x \,\mathrm{l}\, y \equiv\sim x \circ y$$

It can easily be seen that overlap is reflexive and symmetric but not transitive. More interestingly, we can see that $x \circ y$ can mean that either x or y is a part of the other, or that they *share a proper part*. In the last situation we speak of *proper overlap*. Proper overlap may appear strange, because usually we carve up the world in disjoint pieces, but it is not ruled out in Mereology. In some situations proper overlap is very useful. Take for example the territory of a nation. Although a nation can be divided into disjoint counties or provinces, the northern part of a nation (properly) overlaps the eastern part (being the north-eastern part). With national territories at sea something different is the matter. There it is acceptable that a sea or ocean is divided into national territories that do not overlap and even have stretches of no man's land in between. For different types of individuals, proper overlap may, or may not be allowed. For device components, proper overlap is forbidden.

Weak Supplementation Principle Mereology contains an axiom that compels the Weak Supplementation Principle:

$$x \ll y \supset \exists z^{\ulcorner} z \ll y \wedge z \,\mathrm{l}\, x^{\urcorner}$$

It means that when an individual has a proper part, it must have another proper part disjoint from the first. The philosophy behind this principle is that an individual cannot be distinguished from the sum of its parts. An individual that consists of only one part therefore cannot be distinguished from that part because the part and the sum of the part are the same. Put into a component context, the weak supplementation principle says that a device cannot be composed out of one single component, because there is nothing to compose with just one component. In such a situation, the device *is* the component.

Binary Sum, Product and Difference By definition, two overlapping individuals share at least a part. Each of these common individuals is called a lower bound of the overlapping individuals. Mereology demands that for each pair of overlapping individuals x and y there exists an individual that has all lower bounds as parts. This individual is called the greatest lower bound, or *binary product* and is denoted as $x \cdot y$ (for non-overlapping individuals x and y, $x \cdot y$ is an invalid description). The reason for demanding the existence of binary products is that when two individuals x and y share a group of individuals (parts), the shared individuals can be distinguished from both x and y. This suggests that this group is an individual itself. A good example are the regions of a country mentioned above. The fact that the northern part of a country overlaps the eastern part suggests that the overlapped part forms region itself: the north-eastern part.

The mereological sum of individuals x and y, $x + y$ is the individual that overlaps all and only those individuals that overlap at least one of x and y. Informally, it is that individual that has only x and y and the parts of x and y as parts. The assumption of the existence of a binary sum for every pair of mereological individuals is the most controversial property of classical extensional mereology. For overlapping individuals the existence of the binary sum may be appropriate since they are `connected' into one individual by their shared parts. Also, for individuals of the same kind (Jack and Jill) or individuals that are spatially close, the existence of a sum is plausible, but for individuals in general it is a too strong axiom. Two arbitrary individuals may be spatially remote and of completely different types, so it is strange to say that they form an individual that is the sum of both. The reason that the existence of binary sums is nevertheless an axiom of Mereology will be made clear in the following paragraphs.

For individuals that properly overlap, the mereological difference ($-$) exists. For individuals x and y, $x - y$ is the largest proper part of x that does not share a part with y.

General Sum and Product By taking binary sums of binary sums and binary products of binary products, sums and products of finite numbers of individuals can be taken. Furthermore, the existence of finite sums and products is guaranteed because of the existence, commutativity and associativity of binary sums and products. However, classes of individuals may be infinite, and then the existence of their sums is not guaranteed. Likewise, products for classes of individuals sharing common parts are also not guaranteed. To account for this, a notation and definition for general sums and products must be introduced.

The sum of each individual x in class F is denoted as $\sigma x^{\ulcorner} F x^{\urcorner}$ and the product of each individual x in a class F as $\pi x^{\ulcorner} F x^{\urcorner}$. Their definitions make use of the definitional notation $\iota x \ldots$, which means *the individual x which is such that* $\ldots$:

$$\sigma x^{\ulcorner} F x^{\urcorner} \approx \iota x \forall y^{\ulcorner} x \circ y \equiv \exists z^{\ulcorner} F z \wedge z \circ y^{\urcorner\urcorner}$$

$$\pi x^{\ulcorner} F x^{\urcorner} \approx \iota x \forall y^{\ulcorner} y < x \equiv \forall z^{\ulcorner} F z \supset y < z^{\urcorner\urcorner}$$

By introduction of the class of all individuals that are common parts of the individuals of class F, the general product can be expressed in terms of a general sum:

$$\pi x^{\ulcorner} F x^{\urcorner} \approx \sigma x^{\ulcorner} \forall y^{\ulcorner} F y \supset x < y^{\urcorner\urcorner}$$

In the same way, by introduction of additional classes of individuals (the class of pairs), binary sums, products and differences can be defined in terms of general sums:

$$x \cdot y \approx \sigma z^{\ulcorner} z < x \wedge z < y^{\urcorner}$$

$$x + y \approx \sigma z^{\ulcorner} z < x \vee z < y^{\urcorner}$$

$$x - y \approx \sigma z^{\ulcorner} z < x \wedge z \,\mathrm{l}\, y^{\urcorner}$$

Axioms The great strength of Classical Extensional Mereology is that by expressing binary sums, products and differences in terms of general sums and products, only one additional axiom has to be added to the axioms of irreflexivity, transitivity and the Weak Supplementation Principle to get a full axiomatization of the theory. This axiom is called the General Sum Principle that guarantees the existence of general sums:

$$\exists x F x \supset \exists x \forall y \ulcorner y \circ x \equiv \exists z \ulcorner F z \wedge y \circ z \urcorner \urcorner$$

Note that this axiom is in fact is an axiom rule. It defines an axiom for each class of individuals that exists.

Extensionality One of the theorems that can be derived from the axioms mentioned above is that of Extensionality. Just like the question whether an individual could be distinguished from its only part, we can ask ourselves the question whether two individuals that have exactly the same parts can really be distinguished. In Extensional Mereology, this is impossible. Although for some kinds of individuals, such as events and singular objects (classes or masses) this is not a problem, there are kinds of individuals for which it can be.

Consider for instance the situation where the same group of people form two different commissions. This situation can occur in everyday life, but is not allowed in extensional theories. Other occasions where problems arise is when temporal aspects of individuals are considered. It is common that parts of individuals change over time, but by using an extensional theory, we commit ourselves for instance to the dramatical point of view that a child that looses a milk-tooth is not the same child anymore.

Atomism In an atomistic mereological theory every individual is made out of individuals that are building blocks, or atoms. An atom is an individual that has no proper parts:

$$\mathrm{At}\, x \equiv\sim \exists z \ulcorner z \ll x \urcorner$$

Some kinds of individuals are definitely atomistic and some are not. The off-the-shelf components where a device is assembled from are the atoms of the device. An example of non-atomistic individuals are intervals of time: a time interval can always be split in shorter intervals. Whether atomicity is appropriate depends on the type of individuals that are described.

Atomicity can be enforced with the axiom

$$\forall x \exists y \ulcorner \mathrm{At}\, y \wedge y < x \urcorner$$

and atomlessness with

$$\forall x \exists y \ulcorner y \ll x \urcorner$$

There are also variants of Mereology where for different kinds of individuals, different atoms can be defined. In such a theory one can express for instance that off-the-shelf components are the atoms of all devices and that cells are the atoms of organisms.

Although it might be *convenient* to think of continua such as time and space as being made out of points in time/space (the atoms), it is debated in philosophy that this is the *`correct'* way to describe reality. So instead of regarding points in time or space as the atomic *parts* of individuals, other theories have been developed where they form the *boundaries* between individuals (Clarke 1981; Galton 1996). In these theories, a point is the boundary of two (or more) continuous individuals that are connected (touch each other). We will look further into this subject in Section 3.1.2 where topology will be discussed.

The importance of assuming atomicity or atomlessness is that in the atomistic case, the theory can be simplified due to the fact that an individual will be part of another iff (if and only if) all its atoms are also atoms of the second individual. With this property, the axioms of Mereology can be simplified a great deal. Furthermore, the identity relation (=) can be defined as `having the same atoms as parts'.

Summary We have seen that even for a theory such as Mereology that claims to be a general theory of parts and wholes, we still have to check carefully whether the theory is appropriate to describe the individuals we are interested in. Furthermore, in studying Mereology, we encountered axioms and properties that may have been overlooked if we would have written an ontology of mereology from scratch.

We have also seen that there is not just one mereological theory, but there are flavours of mereology that are different in the ontological commitments they make (the axioms chosen). Each theory is appropriate for different kinds of individuals. This suggests that a *library* of mereological ontologies can be made, where a simple ontology of mereology is specialized into the different flavours of mereology that are appropriate for specific classes of individuals.

After the addition of ontological commitments to create a new flavour, simplifications of axioms and definitions can sometimes be made. Therefore, it can be the case that the differences between the old and the new theory (the extra commitments) may not be clearly visible. It is therefore necessary to include (descriptions of) these commitments as meta-information about the ontologies in the ontology library. Only when this meta-information is accessible, an appropriate ontology can be chosen from the library and knowledge sharing and reuse can be realized,

Ontolingua Implementation

Our mereological ontology is simply an Ontolingua implementation of the Classical Extensional Mereology (Simons 1987) as described above. Two relations define part-of decompositions. The relation `equal(x,y)` defines which individuals are to be considered mereologically equal. An individual `x` is a mereological individual when `equal(x,x)` holds. When a mereological individual `x` is a part of a mereological individual `y`, the relation

`proper-part-of(x,y)` holds. With these relations it is possible to write down a variety of axioms specifying desirable properties any system decomposition should have. Examples are the asymmetry and transitivity of the `proper-part-of` relation.

```
1  define-theory mereology

2  define-class m-individual(x)
a    m-individual(x) <-> equal(x,x)

3  define-relation proper-part-of(x,y)
a    proper-part-of(x,y) -> not proper-part-of(y,x)
b    proper-part-of(x,y) and proper-part-of(y,z) -> proper-part-of(x,z)

4  define-relation direct-part-of(x,y)
a    direct-part-of(x,y) <-> proper-part-of(x,y) and
       not exists z: proper-part-of(z,y) and proper-part-of(x,z)

5  define-relation disjoint(x,y)
a    disjoint(x,y) <-> not(equal(x,y) or
       exists z: proper-part-of(z,x) and proper-part-of(z,y))

6  define-class simple-m-individual(x)
a    simple-m-individual(x) <-> m-individual(x) and
       not exists y: proper-part-of(y,x)
```

Figure 3.2: Excerpt from the mereological ontology. This ontology defines the means to specify decomposition information and the properties any decomposition should have.

Figure 3.2 shows an excerpt from the mereological ontology. Definition 2 defines the class of mereological individuals that has been described above. The (partial) definition of the `proper-part-of` relation clearly shows the asymmetry (3a) and transitivity (3b) axioms. The ontology furthermore defines the relation `disjoint(x,y)` which holds for individuals that do not share a part and `simple-m-individual`, the class of individuals that have no decomposition (the atoms). Note that the definitions only serve as an illustration and are not meant to be complete. For the entire implementation we refer to Appendix A.

3.1.2 Topological Ontology

With the part-of relation of Mereology we are capable to express that devices are assembled out of components, which on their turn, can be made out of smaller components. But to describe the physical behaviour of these devices we must also describe the way components interact. In Section 3.1.5 we will see that component behaviour is a result of energetic physical processes that occur in a component. Interaction between components can therefore be described as energy flowing from one component to another or vice versa. When two components are able to exchange energy they are said to be (energetically) connected.

Just as mereology is a theory for the part-whole relation in general, there is also a theory of the *is-connected-to* relation in general. This theory is called *topology*, after $\tau o \pi o \varsigma$ which is the Greek word for place. As is the case with mereology, there are also different flavours of topology, depending on the kind of individuals connected and the meaning of being connected.

For obvious reasons, we are interested in connectivity of individuals for which the part-whole relation applies. There are two approaches to make such a theory. One is to extend an existing theory of mereology with the topological is-connected-to relation. The other is to integrate mereological and topological concepts and relations into one mereo-topological theory. While the first approach is more intuitive, the second usually leads to more compact theories because axioms can be combined.

In the next section, the popular mereo-topological theory of Clarke will be described. We will not give a complete axiomatization of the theory but will only describe its main principles and peculiarities. After that, in Section3.1.2, we will introduce our own topological theory. This theory is an extension of Classical Extensional Mereology and will serve as an example for one of the types of ontology projection.

Clarke's Mereo-topology

The mereo-topology of Clarke is based on the relation $x \asymp y$ which means "x and y are connected". The easiest way to understand the theory is to assume that connected individuals are spatial regions and that *being connected* means the same as *touches*. With this in mind three axioms concerning the $\asymp$ relation can be defined as follows.

$$x \asymp x$$

$$x \asymp y \supset y \asymp x$$

$$\forall z^{\ulcorner} z \asymp x \equiv z \asymp y^{\urcorner} \supset x = y$$

The first axiom states that the $\asymp$ relation is reflexive, i.e. an individual is always connected to (touches) itself. Furthermore, the relation is symmetric. The third axiom defines an extensionality principle for connected individuals: when two logical variables signify individuals that are connected to exactly the same individuals, the two variables signify the same individual.

Based on the $\asymp$ relation, the relations *disconnected* $]\![$, *part-of* $<$, *proper part-of* $\ll$, *overlap* $\circ$ and *externally connected* $\prec\!\succ$ are defined.

$$x \,]\![\, y \equiv \sim x \asymp y$$

$$x < y \equiv \forall z^{\ulcorner} z \asymp x \supset z \asymp y^{\urcorner}$$

$$x \ll y \equiv x < y \wedge \sim y < x$$

$$x \circ y \equiv \exists z^{\ulcorner} z < x \wedge z < y^{\urcorner}$$

$$x \prec\!\succ y \equiv x \asymp y \wedge \sim x \circ y$$

Something is part of an individual when everything that is connected to the part is also connected to the whole. The definition of proper-part-of relation is slightly different compared

to Mereology. Together with the axiom of extensionality this ensures that in situations where individuals only touch their (proper) parts the theory is equivalent with mereology. In particular the Weak Supplementation Principle holds: when individual x has a proper part y, there must be a second proper part z. This follows from the fact that x must be part of y *but* y *may not be part of* x. When there is no second proper part, x and y touch the same individuals and this would clash with the extensional axiom that says that they would signify the same individual. When there *is* a second part z, y will touch z but x will not so there is no discrepancy.

An individual x is *externally connected* to y when they touch, but they do not overlap. The problem with Clarke's theory is that for being externally connected, the connected individuals must have proper parts. When individuals x and y are externally connected, they touch. But when they do not have proper parts, each individual touches the same individuals (both x and y) so they are identical, and identical individuals overlap. When x has a proper part z, y does not touch z so there is no problem.

To work around the problem it is assumed that every individual has an *interior*, which is the mereological sum of its proper parts. When an individual is externally connected with another, the first touches the second, but not the interior of the second. This results in the strange situation that the Weak Supplementation Principle holds for individuals without external connections, but for individuals with external connections the principle is turned a blind eye on.

The advantage of Clarke's mereo-topology is that it allows to express topological relationships between individuals without having to worry about the nature of the connections. In other topological theories connections are points or surfaces that are shared between the connected individuals. In this viewpoint, connected individuals share a common part, so mereologically speaking, they overlap. This does not sound intuitive, for objects that touch do not share anything, they are just positioned in a way that they make contact. A solution to this problem has been adopted in so called `pointless' theories (Clarke 1981; Cohn, Randell, and Cui 1995; Borgo, Guarino, and Masolo 1996b; Galton 1996). In these theories connections are regarded as the boundaries between objects that touch. Boundaries are different entities than mereological individuals. They cannot exist on their own. Their existence is solely due to the topological configuration of individuals. A complicating factor in theories of this kind is that they have to deal with the dimensional aspects of the boundaries. A surface can only connect *two* objects, whereas points and lines can connect more than two objects. Dealing with these constraints makes these theories rather complex (Galton 1996).

Therefore, in Clarke's theory, points, lines and surfaces are all moved into the boundary of the individuals, which is the difference between the individuals and their interiors. This leads to a compact theory that can be used in many situations but has some peculiarities. One of the peculiarities is the already mentioned inconsistency in the validity of the Weak Supplementation Principle. Another is the fact that there is a difference between a non-atomic individual and the sum of its proper parts (its interior) whereas in Classical Extensional Mereology, a non-atomic individual *is* the sum of its proper parts.

A Topological Extension of Mereology

To illustrate the concept of ontology construction we will now present a topological extension of Mereology. Again, it will be clarifying to imagine that the connected individuals are spatial regions. Connections are regarded as boundaries between connected individuals but we will not distinguish between points, lines and surfaces. We will restrict ourselves to connections that connect a pair of individuals. The theory is presented by explaining its Ontolingua implementation that can be found in Figure 3.3.

```
1  define-theory topology

2  include-theory mereology

3  define-class connection(c)
a    connection(c) -> not m-individual(x)
b    connection(c) <-> exists x,y: connects(c,x,y)

4  define-relation connects(c,x,y)
a    connects(c,x,y) -> connects(c,y,x)
b    connects(c,x,y) -> not(part-of(x,y) or
                            part-of(y,x))
c    connects(c,x,y) and part-of(x,z) and
       disjoint(z,y) -> connects(c,z,y)
d    connects(c,x,y) and not simple-m-individual(y)
       -> exists z: part-of(z,y) and connects(c,x,z)
e    connects(c,x1,y1) and connects(c,x2,y2)
       -> not(disjoint(x1,x2) and disjoint(x1,y2))
```

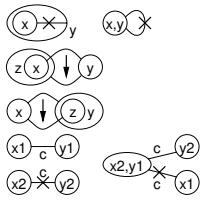


Figure 3.3: Excerpt from the topological ontology. This ontology provides the means to express that individuals are connected. Axioms ensure that only sound connections can be made. These axioms take into account the possible part-of decomposition an individual can have.

The projection performed in this ontology is of the type *include and extend*. Inclusion is done in Line 2 and the extension takes shape by definition of new concepts and relations that use mereology in their axioms. This yields an ontology that has the same level of abstraction as the included mereology.

Line 3 introduces the class of connections. Axiom 3a defines that connections are different things than mereological individuals. The fact that a connection connects two individuals is represented by Axiom 3b. Line 4 defines the `connects` relation. This relation is symmetric in the last two arguments (Axiom 4a) and a connection cannot connect an individual to one of its parts, including itself (Axiom 4b). The rationale behind this is that we do not want to allow self connectedness and, because a non-atomic individual is the sum of its proper parts, also want to disallow connectedness to proper parts. Axiom 4c defines the conditional transitivity in the last two arguments. It states that when connection `c` connects a part `x` to an individual `z` that is disjoint from its whole `y`, the whole `y` is also connected to individual `z` by `c`. Because we assumed that a non-atomic individual is the sum of its proper parts, we must ensure that a connection to a non-atomic individual is also a connection to one of its proper parts (Axiom 4d). Finally we have to disallow that a connection can connect two mereologically separated pairs of individuals (Axiom 4e). This also excludes connections that fork.

3.1.3 Ontology of Systems Theory

On top of the topological ontology the standard system-theoretic notions such as system, subsystem, system boundary, environment, open/closedness etcetera can be defined. Some of these definitions can be found in Figure 3.4. The ontology projection is of the *include and extend* type, just like in the topological ontology.

```
1  define-theory systems-theory

2  include-theory topology

3  define-class system(s)
a    system(s) -> m-individual(s)

4  define-relation in-system(x,s)
a    in-system(x,s) <-> proper-part-of(x,s) and
       system(s) and simple-m-individual(x)

5  define-relation in-boundary(c,s)
a    in-boundary(c,s) <->
       connection(c) and system(s) and
       exists x,y: connects(c,x,y) and
                   part-of(x,s) and not part-of(y,s)

6  define-relation subsystem-of(sub,sup)
a    subsystem-of(sub,sup) <-> system(sub) and system(sup) and
                               proper-part-of(sub,sup)

7  define-class open-system(s)
a    open-system(s) <-> system(s) and exists c: in-boundary(c,s)

8  define-class closed-system(s)
a    closed-system(s) <-> system(s) and not open-system(s)
```

Figure 3.4: Excerpt from the systems theory ontology. This ontology introduces system-theoretic notions on top of the topological ontology.

Definition 3 states that a system is a mereological individual (but not every mereological individual is a system). The relation `in-system(x,s)` holds for individuals that are the atoms of the system. This is different from the relation `subsystem-of(sub,sup)`, where the part can be a non-atomic system. A connection is in the boundary of a system when it connects an individual that is part of the system to an individual outside the system. With this definition, the classes `open-system` and `closed-system` can be defined easily.

3.1.4 Component Ontology

In the component ontology we focus on the *structural* aspects of devices, and abstract from what kind of dynamic processes occur in the system and from how it is described in terms of mathematical constraint equations.

An example of a structural-topological diagram for a physical system, i.c. an air pump, is shown in Figure 3.5. This structural view on physical systems is based upon what we call a *component ontology*.

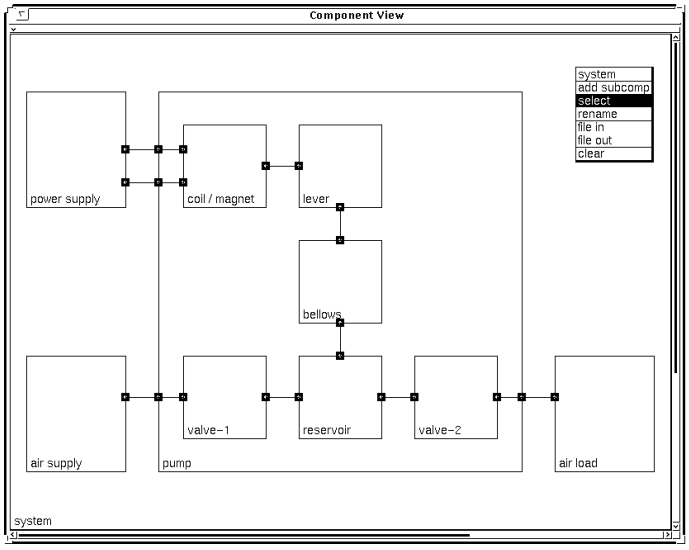


Figure 3.5: The component view on a physical system, showing a two-level part-of decomposition and the system topology for an air pump. Sub-components are drawn inside the area defined by their super-component. The small solid blocks are the interfaces through which components are connected.

Our component ontology is constructed from mereology, topology and systems theory. The ontology projection of the abstract systems theory ontology to the component ontology to accomplish this is slightly more complex than the one introduced in the previous sections. The component ontology defines the structural view on physical systems engineers have as depicted in Figure 3.5, i.e. components that can have subcomponents and terminals. The terminals are the interfaces of the components to the outer world. Therefore, connections hook onto terminals instead of components. This interpretation of components and connections is a bit more complex than the networks of abstract individuals and connections in systems theory. Nevertheless, the definition of these concepts can be kept simple due to a projection of the abstract systems theory on the definitions of engineering components and connections, thus enforcing the components to comply to the rules of systems theory. We will describe the way this projection takes place below. Because this projection makes abstract concepts more specific, this type of projection is called *include and specialize*.

Figure 3.6 shows some definitions from the component view ontology. The important classes are the classes `component`, `terminal` and `physical-system`. The relations `comp.subcomp`, `comp.term` and `conn.term` relate components to their subcomponents, terminals to components and connections to terminals. The general structure of the ontology can also be presented as a conceptual schema like in Figure 3.7.

Only the definitions contributing to the ontology projection are shown in the excerpt. Ontology projection consists of inclusion (Line 2) and the definition of axioms that specify the

```
1  define-theory component-view

2  include-theory systems-theory

3  define-class component(c)
a    component(c) -> m-individual(c)

4  define-relation comp.subcomp(c,s)
a    comp.subcomp(c,s) <-> component(c) and
       component(s) and direct-part-of(s,c)

5  define-relation conn.term(conn,term)
a    conn.term(conn,term) and comp.term(comp1,term)
       -> exists comp2: connects(conn,comp1,comp2)
b    component(comp1) and component(comp2) and
       connects(conn,comp1,comp2)
       -> exists term: conn.term(conn,term) and comp.term(comp1,term)

6  define-class phys-system(s)
a    phys-system(s) <-> system(s) and (in-system(c,s) -> component(c))
```

Figure 3.6: Excerpt from the component ontology. This ontology formalizes the component view of engineers on physical systems. Note that the ontology can be kept relatively simple because systems theory is *projected* onto it.

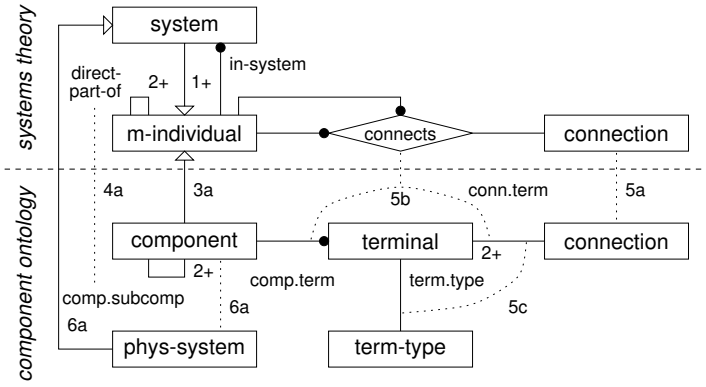


Figure 3.7: Conceptual schema of the component ontology.

abstraction of components to system theoretical concepts. Definition 3 shows how the ontological commitments for abstract mereological individuals are projected onto components. Definition 4 defines the meaning of the `comp.subcomp` relation in terms of mereology. The projection of topological connections onto component connections is performed by Definition 5. Definition 6 defines the modelled device as a system of components. The fact that connections can be of a certain type has been left out of the excerpt to keep it simple.

3.1.5 Physical Process Ontology

Our physical process ontology specifies the behavioural view on physical systems. In the general case it is quite difficult to formalize what the notion of a dynamic process precisely entails. Fortunately, for a certain part of physics this has been done to a level where one can define really primitive process concepts. The approach we take here is known in engineering as system dynamics theory, which also forms the theoretical background of the bond graph method (Karnopp, Margolis, and Rosenberg 1990). The basic idea behind this theory is that the dynamics of a system can always be captured by looking at the change of different kinds of *stuff*. This change of stuff is also called *flow*. For instance, in electrical systems, dynamic behaviour consists of the change of *electrical charge*, i.e. *electrical current*. Likewise, in the mechanical domain the stuff is called *location* and change of location is *velocity*. The thing required to bring about a flow is called *effort*. Table 3.3 lists the types of stuff, flow and effort of some of the physical domains defined in the ontology.

domain	**stuff**	**flow**	**effort**
electrical	charge	current	voltage
mechanical	location	velocity	force
hydraulical	volume	volume flow	pressure

Table 3.3: Some examples of physical domains. In each domain, dynamic behaviour is described as flow, i.e. change of stuff. Effort is that what is required to bring about a flow.

The interesting aspect about this table is that the product of a flow with its related effort has the dimension *energy / time*, i.e. such a pair defines an *energy flow*. Physical behaviour can therefore be defined in terms of energy flows. The process ontology introduces physical mechanisms which are applications of physical laws or principles to one or more energy flows. An important feature of these mechanisms is that they exploit in detail the analogies that exist between different physical domains. For example, the principle of conservation of momentum in mechanics is analogous to induction in the electrical domain. Many more of these analogies exist. This approach is valid for standard classical, deterministic physics, covering such diverse fields as mechanics, electricity and magnetism, hydraulics, acoustics, and thermodynamics.

Complex process descriptions can be formed by making a network of mechanisms, linked by energy flows. This abstraction is used to construct the process ontology. The process ontology *includes* systems theory *and specializes* the system theoretic concepts to processes. Just like the component ontology, the process ontology defines relatively simple concepts

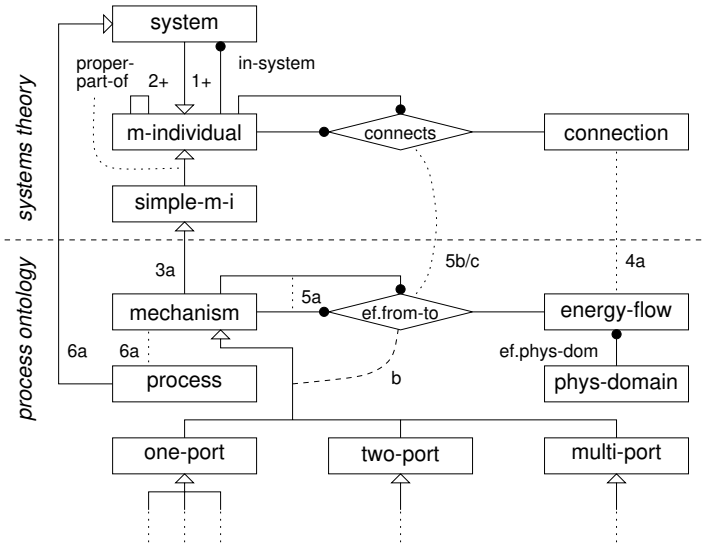


Figure 3.8: Conceptual schema of the physical process ontology

and relations onto which the system ontology is projected. Figure 3.8 gives the conceptual schema of the physical process ontology.

Figure 3.9 gives an excerpt of the physical process ontology. Mechanisms are defined as simple mereological individuals. Energy flows, which have a certain direction, flow from one mechanism to another. The projection is performed by stating that an energy flow is a topological connection that connects the two mechanisms. A process description can now simply be defined as a system of mechanisms. The definition of physical domains as depicted in Table 3.3 is not shown in Figure 3.9.

Figure 3.10 shows the taxonomy of mechanisms as defined in the process ontology. The classes in the figure are present as classes in the ontology and class-subclass relations are defined for every line in the figure. The discriminating properties used to construct this taxonomy are the number of energy flows linked by a mechanism (connectivity), the mechanism type, whether effort or flow plays the most important role with respect to the mechanism type and the physical domain (e.g. mechanics, electricity, hydraulics, thermodynamics). Note that some discriminating properties are not useful for certain types of mechanisms.

The order in which the discriminating properties are applied here is the opposite of the order used in the typical engineering education. There, the distinction between physical domains is made first: there are separate courses in mechanics, electrical engineering and thermodynamics. Only when students have mastered all courses, they are able to see the analogies between the domains that makes the process ontology as compact and elegant as it is.

```
1  define-theory process-view

2  include-theory system-theory

3  define-class mechanism(m)
a    mechanism(m) -> simple-m-individual(m)

4  define-class energy-flow(ef)
a    energy-flow(ef) -> connection(ef)

5  define-relation ef.from-to(ef,f,t)
a    ef.from-to(ef,f,t) -> not ef.from-to(ef,t,f)
b    ef.from-to(ef,f,t) -> connects(ef,f,t)
c    energy-flow(ef) and connects(ef,x,y)
       -> ef.from-to(ef,x,y) or ef.from-to(ef,y,x)

6  define-class process(p)
a    process(p) <-> system(p) and (in-system(m,p) -> mechanism(m))
```

Figure 3.9: Excerpt from the process ontology. This ontology formalizes a large part of physics. It defines how process descriptions can be formed by making a network of domain-independent mechanisms and energy flows. The ontology is relatively simple because systems theory is used for the definition of the networks.

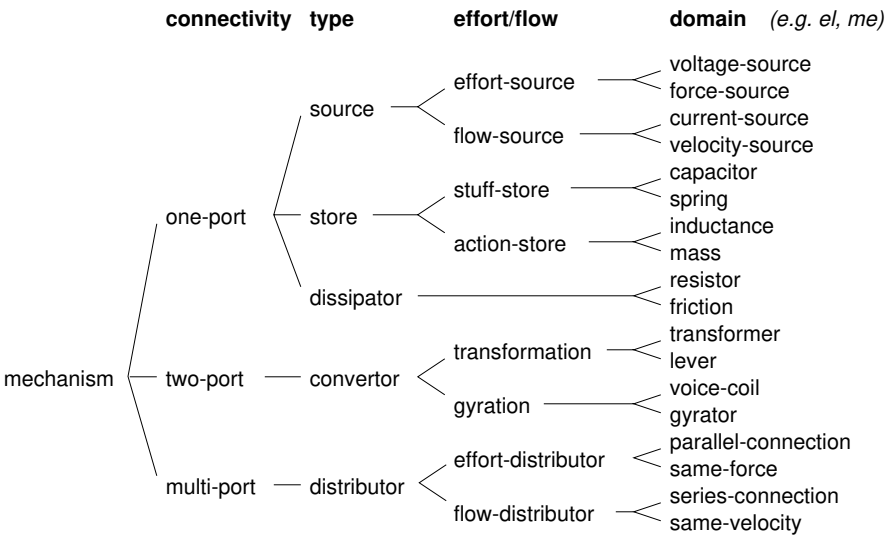


Figure 3.10: The taxonomy of physical mechanisms. The properties discriminating between the classes after branching are printed above the branch points. The classes on the right give some examples of mechanisms in the electrical and mechanical domain.

3.1.6 Mathematical Ontology

The mathematical ontology defines the mathematics required to describe physical processes. The EngMath ontology (Gruber 1994), available in the Ontolingua ontology library, is perfectly suited for this job and has therefore been (re)used for this. In this section, only a very short description is given that should be sufficient to understand the projection of the process ontology onto mathematics. For detailed information on EngMath see (Gruber 1994).

The EngMath ontology formalizes mathematical modelling in engineering. The ontology includes conceptual foundations for scalar, vector, and tensor quantities, physical dimensions, units of measure, functions of quantities, and dimensionless quantities.

A *physical quantity* is a measure of some quantifiable aspect of the modelled world characterized by a *physical dimension* such as length, mass or time. Quantities in the EngMath ontology can be expressed in various *units of measure* e.g. meter, inch, kilogram, pound, etc. The ontology defines the relationships between quantities, units of measure and dimensions. A special class of physical quantities are *time-dependent quantities*. These are in fact continuous functions from a quantity with the time dimension to another physical quantity, and can therefore be interpreted as dynamic quantities, varying over time.

Another important EngMath class for the PHYSSYS ontology are the Ontolingua (KIF) expressions that serve as a meta-level description of mathematical relations between physical quantities. For instance, a relation `r` between two time-dependent physical quantities `x` and `y` can be defined as:

```
define-relation r(x,y)
  r(x,y) <=> x = 2 * y
```

Here, the expression `x = 2 * y` is the (infix form of the) Ontolingua expression used to define the EngMath relation `r`. In the PHYSSYS ontology, two time-dependent physical quantities are used to mathematically describe an energy flow and relations similar to `r` define the mathematical relationships between these quantities.

3.1.7 Ontology Projections

The PHYSSYS top-level ontology is an ontology that only performs ontology projections. It includes the component, process and EngMath ontologies and relates them to each other. Figure 3.11 shows an example of the relations between instantiated views on a physical system. Basically, it defines that components are the carriers of physical processes that can be mathematically described with physical quantities and mathematical relations.

Figure 3.12 gives an overview of the way the mapping relations have been defined. In the next section we will first describe the mapping from components to physical process descriptions. After that, the mapping from physical mechanisms to mathematical relations will be explained.

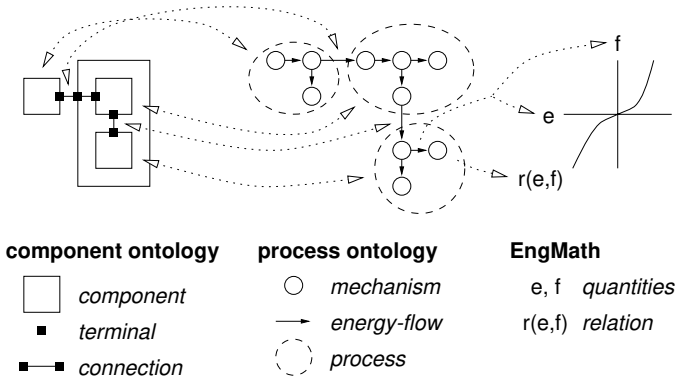


Figure 3.11: Interdependencies between the component, process and mathematical ontologies. Roughly it states that a component is the carrier of physical processes that are described by mathematics.

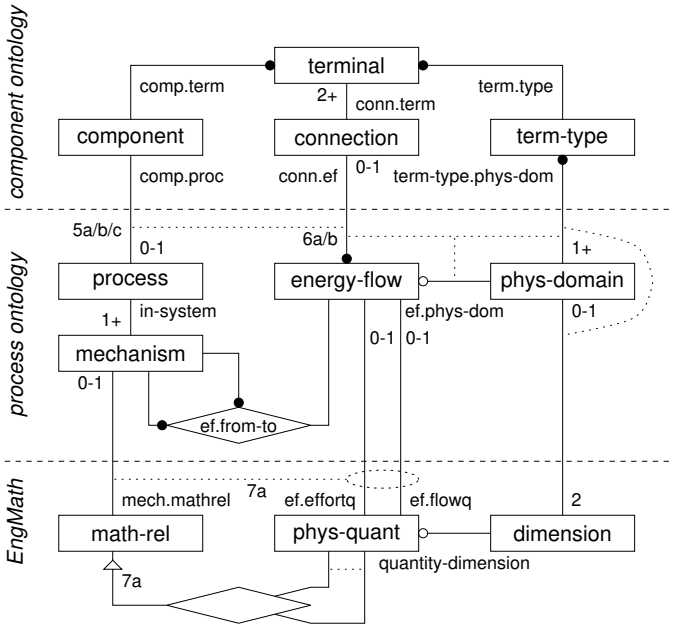


Figure 3.12: Conceptual schema of PHYSSYS' top-level ontology

Mapping of Components to Processes

Figure 3.13 shows the first part of the PHYSSYS ontology that includes the three viewpoints (lines 2, 3 and 4) and relates the component and process views (definitions 5 and 6). The relation `comp.proc` (Definition 5) implements the projection of simple components to process descriptions. Axiom 5a states that every atomic component must have a process description and Axiom 5b that each mechanism must be part of the process description of a component. Axiom 5c ensures that a mechanism can only be part of one process description of one component. The relationship between component connections and the energy flows between the process descriptions of these components is expressed by Definition 6. For each connection between components, the process descriptions of these components must interact via an energy flow (Axiom 6a). Vice versa, Axiom 6b defines that an energy flow between the process descriptions of two components goes through a connection. Note that the relationship between the type of a connection and the number and domain of the energy flows of this connection has not been included in the excerpt.

```
1  define-theory PhysSys

2  include-theory component-view

3  include-theory process-view

4  include-theory EngMath

5  define-relation comp.proc(c,p)
a    component(c) and simple-m-individual(c)
       -> exists p: process(p) and comp.proc(c,p)
b    mechanism(m) -> exists c,p: process(p) and
       in-system(m,p) and comp.proc(c,p)
c    comp.proc(c1,p1) and comp.proc(c2,p2) and
       c1 != c2 -> disjoint(p1,p2)

6  define-relation conn.ef(c,ef)
a    conn.term(c,t1) and conn.term(c,t2) and comp.term(c1,t1) and
       comp.term(c2.t2) and comp.proc(c1,p1) and comp.proc(c2,p2)
         -> exists ef: conn.ef(c,ef) and in-boundary(ef,p1) and
                                         in-boundary(ef,p2)
b    energy-flow(ef) and process(p1) and
       in-boundary(ef,p1) and comp.proc(c1,p1) and
       process(p2) and in-boundary(ef,p2) and comp.proc(c2,p2)
         -> exists c: comp.term(c1,t1) and conn.term(c,t1) and
                      comp.term(c2,t2) and conn.term(c,t2)
```

Figure 3.13: Excerpt from the first part of the PHYSSYS ontology where components are projected onto physical processes. This is an example of an ontology that only contains formalizations of the interdependencies between the viewpoints it includes.

Mapping of Processes to EngMath

Mapping of the process ontology to EngMath is depicted in the right-hand side of Figure 3.11. Informally, the mapping states that for each energy flow there are two time-dependent physical quantities, one for the effort and one for the flow. The domain of the

energy flow determines the dimension of the quantities. For instance, an electrical effort quantity has the dimension `energy/electrical-current` (voltage) and the flow quantity the `electrical-current` dimension. For each mechanism there is a mathematical relation that relates the values of the physical quantities of the energy flows connecting the mechanism to each other. The mapping also imposes constraints on the relation. These constraints only depend on the mechanism type and they are independent of the domains of the energy flows (Beaman and Rosenberg 1988). The mathematical relation in Figure 3.11 belongs to a dissipator mechanism. The constraints on such a relation are that it is a continuous function $r : e \mapsto f$ that lies in the first and third quadrant and that $r(0) = 0$. For an electrical energy flow, this can be an instantiation of Ohm's law $V = I \times R$ whereas in the mechanical domain it can model some kind of friction with $F = k \times v$.

```
7  define-relation mech.mathrel(m,r)
a    dissipator(m) and ef.from-to(ef,n,m) and
     ef.effortq(ef,e) and ef.flowq(ef,f) ->
       exists r: mech.mathrel(m,r) and
         forall t: value-at(e,t) = et and value-at(f,t) = ft ->
           r(et,ft) and (et * ft >= 0 * energy-dimension) and
           zero-quantity(et) <-> zero-quantity(ft)
```

Figure 3.14: Excerpt from the second part of PHYSSYS ontology where physical processes are projected onto mathematical relations. This is an example of a domain ontology that only contains formalizations of the interdependencies between the viewpoints it includes.

Figure 3.14 shows an excerpt of the second part of PHYSSYS, the part that performs the process to mathematics projection described above. Only a part of the definition of the relation `mech.mathrel`, the relation that relates a mathematical relation to a process, is shown. Axiom 7a states that for every dissipator a relation between the effort and flow quantities of the energy flow to the dissipator must exist. In the axiom the relations `ef.effortq` and `ef.flowq` are used. These relations link each energy flow to physical quantities for the effort and the flow. Axioms not incorporated in Figure 3.14 ensure that these quantities have the proper physical dimension. The constraints on the relation for dissipators have been formalized by stating that the effort is zero if and only if the flow is zero and that the product of effort and flow, i.e. the energy flow, must be positive. In other words, the dissipator must *dissipate* energy. Furthermore, it is probably needless to say that PHYSSYS contains axioms like axiom 7a for each type of mechanism defined in the process view.

3.2 Related Research

In comparing the contents, design principles and application of the PHYSSYS ontology with related work, we will consider several aspects:

- Domain theories concerning physical systems and engineering embodied in the ontology.

- Top-level, general `super' theories (as we have called them) incorporated in the ontology.
- Constructive aspects of ontologies, and in particular the notion of minimum ontological commitment as a design principle.
- The role of ontologies in the knowledge engineering process.

Our PHYSSYS ontology intends to capture widely applicable concepts and background theories in physical systems engineering, an area which has stimulated quite a significant effort in ontology development in various application directions (Top and Akkermans 1994; Neches, Fikes, Finin, Gruber, Senator, and Swartout 1991; Alberts 1993; Gruber 1994; van der Vet, Speel, and Mars 1995; Bernaras and Laresgoiti 1996; Benjamin, Borst, Akkermans, and Wielinga 1996). A feature of our approach is the postulated existence of different conceptual viewpoints on the domain objects and reasoning about them, that is, a grouping of properties of the domain objects into separate `natural' categories. This has the advantage that it leads to a corresponding strict separation of ontologies (describing these properties) as a basic structuring principle. The ontological viewpoints selected here — technical system components, physical processes, mathematical descriptions — have been adopted from (Top and Akkermans 1994), but are much further worked out and operationalized in the present work. It is gratifying that the EngMath ontology for engineering mathematics (Gruber 1994) could be reused here and integrated into a wider physical systems ontology.

We do not at all want to imply that these are the only possible or relevant viewpoints. In the next chapter we will see that the determination of exogenous model and design parameters often proceeds on the basis of geometric and material properties. This suggests to extend the set of ontologies. We are currently developing a geometry view, which has to be compatible with the already available topology theory, and can be linked to the relevant part of the EngMath ontology by means of an ontology projection. Spatial ontologies are discussed e.g. in (Cohn, Randell, and Cui 1995). Also, work on separate ontologies for material properties is ongoing, such as the Plinius ontology (van der Vet, Speel, and Mars 1995). In tasks like analyzing the environmental impact of alternative designs for engineering systems, the topic of Chapter 5, modelling the material properties is a key issue.

In other work, we also find the idea of using different conceptual viewpoints to modularize ontologies but usually in a much looser way. In (Ushold 1992), two viewpoints containing such meta-knowledge are considered: an ontology for the domain of ecological modelling and a mathematical ontology based on lambda calculus to execute (simulate) the models. Macroscopic and microscopic viewpoints for electronics have been combined by Liu (1992). The problems of mereo-topology regarding time aspects are being worked upon by Borgo (1996b). He defines three viewpoints on objects: a viewpoint of conceptual physical objects, a viewpoint of chunks of matter and a viewpoint of regions of space.

In the Ontolingua library, also a thermal systems ontology has been specified (Neches, Fikes, Finin, Gruber, Senator, and Swartout 1991). Basically, it contains a number of thermal components (such as `boiler'), for which then mathematical equations are given that specify the component behaviour. In our approach, such models are not part of the ontology, but are

found in the OLMECO library (see Chapter 4). The PHYSSYS ontology rather specifies what such models should look like *in general*. Thus, our ontology expresses *meta-level knowledge* concerning modelling and simulation. This conforms to the view in (Schreiber, Wielinga, Akkermans, de Velde, and de Hoog 1994; Schreiber, Wielinga, and Jansweijer 1995), where ontologies are viewed as meta-level specifications of a set of possible domain theories or models.

This, by the way, does have practical consequences, since in our case there are clearly more constraints than in the KSE library on how components can or cannot be connected at the component level, and what implications follow for the assembly of mathematical models. Closest to our approach is probably the YMIR ontology of (Alberts 1993), which is also based upon general systems theory. The systems part is essentially the same as the PHYSSYS component ontology, although it is not constructed out of smaller ontologies about mereology and topology. YMIR pays more attention than both PHYSSYS and EngMath to possible abstraction steps from larger to simpler models at the mathematical level. A major difference is the absence in YMIR of a process ontology. Like in the KSE physical systems library, physical processes are in fact equated with their mathematical descriptions. This is also a typical situation in AI qualitative reasoning frameworks that are device- and constraint-oriented, *cf.*, (Kuipers 1994) and references therein.

We have not made this choice for fundamental reasons: (i) it is common in knowledge acquisition to encounter forms of conceptual or qualitative reasoning (also by experts) about physical processes without mathematics; (ii) in general the relationship between physical processes and mathematical descriptions is *n-to-n*. Both our ontology and the OLMECO library that will be described in the next chapter cater for this, leading to more flexible modelling. Thus, the process ontology is a salient feature of PHYSSYS. Forbus' qualitative process theory (Forbus 1984) and the associated modelling framework (Falkenhainer and Forbus 1991) are much in the same spirit, but there are important differences in the underlying ontology. In contrast, the PHYSSYS process ontology is formally built in a compositional way on a set of primitive physical mechanisms, that in addition satisfy generic ontologies concerning mereology, topology and (network) systems theory. All this is left much more open (as well as much more informal) in the ontology underlying QPT, resulting in less commitment and less guidance. One side of the coin is that QPT allows to specify processes according to, say, medieval, Aristotelian or commonsense physics. This is not possible in PHYSSYS as it commits to modern physical science [1]. The other side of the same coin is that, due to this lack of commitment, it is much easier in QPT to come up with nonsensical process models. Here, PHYSSYS provides more physics knowledge and guidance — that is, according to current scientific standards.

A unique, strongly unifying, characteristic of PHYSSYS is that it formally specifies and exploits the analogies between different fields in physics. This makes it much more widely applicable and reusable than first selecting a physics subdomain (e.g. thermodynamics) and

[1] To allow types of physics other than the current scientific one, we would have to fundamentally revise the axioms pertaining to the elementary physical mechanisms. It seems to us a very interesting exercise, by the way, to try and find a similar comprehensive axiomatization of very different notions concerning physics, like the medieval impetus theory.

restricting the ontology to this subdomain as is usually done. This again has practical consequences: many modern engineering systems are inherently multidisciplinary —mechatronic systems but also heating systems are good examples— and restricting ontologies in such situations to the standard physics subdomains not only hampers reusability but also usability.

In some ontologies for technical domains, see (Bernaras and Laresgoiti 1996; Benjamin, Borst, Akkermans, and Wielinga 1996) and other ontologies from the KACTUS project, where we do find a separate notion of physical processes or phenomena, it resembles a function-oriented abstraction of our process notion. The reason is that it depends on the task how much detail one needs. In our type of tasks, control- and design-oriented prediction, process detail is required for making adequate modelling decisions. In other tasks, such as electrical network diagnosis and service recovery as in (Bernaras and Laresgoiti 1996), (dys)function abstractions of underlying processes are sufficient to do the job.

We now turn to aspects of what (van Heijst, Schreiber, and Wielinga 1997) calls generic ontologies, representing theories that are supposed to be valid across many fields. Devising a satisfactory top-level categorization of generic concepts (such as thing, object, state, event, etc.) has turned out to be extremely hard (Lenat and Guha 1990; Sowa 1995; Skuce 1993; Benjamin, Borst, Akkermans, and Wielinga 1996). On the other hand, the present work has indicated that it is practically feasible and useful to use and reuse generic theories such as mereology, topology and systems theory in domain ontology building. Hobbs (Hobbs 1995) comes to a similar conclusion (he calls it `core theories') in the context of language understanding. These generic ontologies are abstract theories that define particular kinds of relations (part-of, connected-to, etc.) over abstract entities. A standard top-level concept taxonomy for such entities apparently is not a requirement for the reuse of generic ontologies. What happens is that these abstract entities are projected onto the relevant domain objects. After this, the way is open for further extension and specialization by adding axioms expressing more specific domain knowledge.

Concerning the contents of the generic ontologies that have been reused in PHYSSYS, we note that we have employed rather classical theories of mereology and topology. Alternatives are being discussed also in the ontology literature (Gerstl and Pribbenow 1995; Guarino 1995; Eschenbach and Heydrich 1995). One of the efforts in ontology research is to *combine* mereology and topology in one theory that expresses the part-of relation in terms of connectedness (Clarke 1981). In PHYSSYS we have followed an approach similar to what is described in (Eschenbach and Heydrich 1995) where mereology is *extended* with topological relations. This is because we are not primarily interested in the philosophical question whether part-whole and connectedness relationships can be expressed using only one relation within a single theory. Rather, we want to reflect the engineering practice where components are thought to be decomposed first and connected later on (often as off-the-shelf components) as a step in configuration design. Furthermore, an ontology of mereology without topology imposes less ontological commitments. Our approach is based on incremental specification which yields more structure and is also easier to understand. Once more, we would like to stress that other alternatives are less preferable than our ontologies, they just differ in the way they are constructed, their compactness and the situations in which they can be used. The important thing is that the differences between alternative ontologies should be clear.

In this chapter we have barely touched upon the issue of method ontologies: ontologies that are oriented towards problem solving methods, specifying information requirements that must be fulfilled by domain models such that inference methods can be executed (Gennari, Tu, Rothenfluh, and Musen 1994; Tu, Erikson, Genari, Shahar, and Musen 1995; van Heijst, Schreiber, and Wielinga 1997). This subject will be elaborated on in Chapter 5.

Gruber has listed in (Gruber 1995) a number of design principles for ontologies — clarity, coherence, extensibility, minimal encoding bias, minimal ontological commitment. In general, these turn out to be valid design principles as far as the PHYSSYS ontology is concerned. The principle of `minimum ontological commitment' deserves however some further discussion. In (van Heijst, Schreiber, and Wielinga 1997) it is suggested to operationalize this principle as the minimization of the number of theory inclusions in the ontology. Guarino *et al.* (1995) propose a formalization of ontological commitment in a modal-logic style. Informally and roughly stated, statements of an ontological theory must be true in every possible world; ontological commitment comprises the set of possible worlds thus allowed by the ontological theory specification. The minimum commitment principle favours the weakest theory (maximum number of models) and tends to emphasize the danger of overcommitment by excluding allowable worlds. In our opinion, there are two practical dangers: excluding acceptable possible worlds, but also including undesired ones. Overcommitment leads to reduction of reuse and sharing, but undercommitment diminishes end-user guidance and support. For example in the component part of our ontology, an issue is whether or not one wants to have typed connections between components. No typing means less commitment, but it also implies that end users are not prevented from making undesirable system models whereby an electrical outlet of one component is directly connected to a mechanical plug of another component. Our experience in the PHYSSYS ontology is that the specification is a balancing act between over- and undercommitment.

Van Heijst *et al.* (1997) discuss a number of ways to categorize and organize ontologies, and what role they play in the knowledge engineering process. In the categorization of Van Heijst *et al.*, the PHYSSYS ontology is a knowledge-modelling ontology. They point to the need to explicate ontological commitments early in the process. As a method to modularize and organize ontologies, they suggest, first, to single out basic concepts (such as patient, disease, therapy) on the basis of `natural categories' of the field to construct some widely usable base ontologies; to specialize these concepts with respect to various relevant (here, medical) subdomains; and then add method-oriented extensions. These are steps needed in achieving modularity of ontologies, which is seen as a key principle in ontology library organization (see especially their Section 3). We will consider the impact of our work in this regard in the next paragraphs.

There is in our opinion no doubt that *modularity* is indeed a key success factor to ontology library construction. In any large-scale application we face what Van Heijst *et al.* call the hugeness problem: the enormous amount of domain knowledge that is involved in expert tasks. However, as these authors point out, concepts involved come in different levels of generality, and this gives a handle on organizing an ontology library. This is clearly visible in the structure of the GAMES-II core library, and we have deployed a very similar approach, as depicted in Figure 3.1. What Van Heijst *et al.* call `generic concepts' is very akin to our

reusable *`super'theories*. They claim that partitioning should be based on two considerations: (i) definitions are to be centered around available `natural categories' of concepts that belong together, and (ii) the number of theory inclusions must be kept to a minimum.

Although we are generally in agreement with these views, our PHYSSYS and OLMECO work offers some different perspectives as well as extensions. As we have argued earlier in this section in relation to the work of (Gruber 1995), minimization of theory inclusions to achieve minimum ontological commitment is a phrase that is in danger of simplifying the real picture in applications. Rather, we would phrase it as *piecemeal ontological commitment*: starting from (indeed) the minimal side, one needs to incrementally build up the ontological commitments until the right degree of commitment for the particular application is achieved. The organization of an ontology library must be modular in such a way that this can be realized.

In (van Heijst, Schreiber, and Wielinga 1997), the proposed `natural categories' are groups of concepts that naturally belong together, reflecting the social consensus of a certain expert community. We share the observation that such natural categories do exist (although they may be so self-evident to a community that they are implicit for outsiders), and that they provide a good handle for partitioning ontologies. This has also been a major structuring principle in PHYSSYS. In our case, we have called these natural categories *viewpoints* (Top and Akkermans 1994), because our ontology organization has been based from the start on coherent categories of properties of the same class of real-world objects, rather than on categorizing different real-world objects. In modeling and simulation, engineers view the same object (say, a hospital heating system under design) sometimes as a collection of connected components, or as a collection of interacting physical processes, etc., depending on the type of information that is to be extracted or decisions that are to be made.

Talking about partitioning and modularity, naturally leads us to the question how ontological modules can be connected again to meaningful assemblies. Here, our work offers an important new extension. The standard mechanism for configuring ontologies is theory inclusion, as it is used in most ontology work including (van Heijst, Schreiber, and Wielinga 1997). We have found in our applications that richer and more flexible means for linking ontologies together are necessary, that go beyond relatively simple inclusion, specialization and extension operations. To this end we have developed what we call *ontology projections*: connections between two different ontologies realized by a mapping, that is highly knowledge-intensive itself and therefore assumes the form of an ontology in its own right. A good example is our specification of the connection between physical process knowledge and mathematical theory concepts.

Finally, we want to emphasize that using explicit ontologies yields benefits for a much wider range of information systems than KBS only. The PHYSSYS ontology is a formalization of domain knowledge of the QUBA modeling assistant (Top and Akkermans 1994) and its successor IMMS, the KBS 007 for automated model revision (Pos, Borst, Top, and Akkermans 1996; Pos, Akkermans, and Top 1996), and the OLMECO library of reusable mechatronic models reported in Chapter 4.

007 (Pos 1997) is a KBS in which model revision is actively carried out by the system on the basis of repair plans. Model revision itself is based upon a (considerable) extension of

the Propose-and-Revise method (Marcus and McDermott 1989). What is interesting in the present context is that some of the repair plans are able to automatically adapt models, such that they conform to the requirements of given simulation methods. Thus, some repair plans function on the basis of knowledge about available method ontologies and about method-oriented revisions of domain models.

The OLMECO library is, at least in its implementation, a conventional database. Explicit ontologies are helpful in two ways here. First, they support and even enforce a sharply defined conceptualization of the information in the system in a way that is natural to the user (some might perhaps want to view this as a formal and high-quality `data dictionary'). Highly important is the experience that ontologies are a great help in clarifying the many tacit and implicit aspects involved. Second, from the modular organization of ontologies the modular structure of the information system itself quite easily follows. This proved to be strongly beneficial in the OLMECO library work, resulting in a design and demonstrator system that was appreciated by end users. Hence, also in a conventional implementation setting, this approach leads to a knowledge-oriented system design that reflects the way users view their world during task execution.

3.3 Ontology Construction

In developing PHYSSYS we found out that an ontology for a large and complex domain can be constructed out of smaller ontologies by carving up the domain knowledge in smaller pieces. Because of this internal structure, the ontology will be easier to understand and be well suited for reuse. The ontologies that form the building blocks of PHYSSYS can be categorized in three types:

`Super'theories which are general and abstract ontologies such as mereology, topology, systems theory.

Viewpoint or base ontologies that formalize a conceptual category of concepts in a domain. For the physical domain at least three of such categories exist: that of a configuration of components, physical processes underlying behaviour and the engineering mathematics that is used to describe the processes.

Domain ontologies that form an integral and coherent conceptualization of a domain. The conceptualization of the domain of physical systems offered by PHYSSYS is nothing other than a combination of the three viewpoints plus the formalization of the interdependencies between the concepts in different viewpoints.

Distinguishing pieces of knowledge of these three types can modularize a large domain in small, reusable modules. To construct a large ontology from smaller ontologies, the dependencies between concepts and relations in different ontologies are formalized as *ontology projections*. Three types of ontology projections were used and are named according to the way they can be implemented.

Include and extend: An imported ontology is extended with new concepts and relations. The result is at the same level of abstraction as the included ontology. An example is the extension of mereology to topology that was described in this chapter.

Include and specialize: An abstract theory is imported and applied to the contents of the importing ontology. Doing this, abstract concepts are specialized. An abstract `super' theory can be considered generic when there are many useful specializations that can be made. For instance, systems theory is used twice in PHYSSYS. It is used as an abstraction of system components as well as an abstraction of physical process descriptions, and it is easy to imagine its application in other domains.

Include and map: Different viewpoints on a domain are joined by including the views in the domain ontology and formalization of their interdependencies. In contrast to the previous ontology projections, these projections contain a great deal of domain knowledge and can therefore be considered to be ontologies of their own.

From our study of mereology and topology, we found out that a library of ontologies may contain abstract ontologies that define similar abstract concepts and relations, but are applicable in slightly different situations. To be able to retrieve the right ontology for an application, the library system should make clear the cases in which an ontology can be used and describe the differences between similar ontologies.

Chapter 4

Using an Ontology as an Information System Specification

The previous chapter described the PHYSSYS *ontology that specifies what a valid physical model looks like. In this chapter a library of model fragments is presented that allows engineers to construct large simulation models by combining model fragments from the library. We will demonstrate the way in which* PHYSSYS *played a role in the design of the library. A modelling and simulation experiment carried out using thermodynamic models from the library, serves as a validation of the library as well as* PHYSSYS.

The design of the OLMECO library is described in Section 4.1. Model fragments of thermodynamic systems will serve as illustrations. Section 4.2 discusses the design of the thermodynamic model fragments in the library, which can be found in Appendix B. Section 4.3 describes the modelling and simulation of the existing heating system of a general hospital. Section 4.4 discusses the main bottlenecks in the use of the library in more detail and suggests a possible extension of OLMECO and PHYSSYS that could resolve the remaining problems. In the final section of this chapter we will draw conclusions about the OLMECO library and the usability of PHYSSYS.

4.1 The OLMECO Library of Simulation Models

In this section we will first examine the way in which the physical models formalized by PHYSSYS are constructed by engineers. This provides us with the information about what kind of data a model library should contain to support the modelling process. The remainder of this section will describe the OLMECO model library that has been designed in accordance to these observations.

4.1.1 Evolutionary Modelling

The explicit separation between conceptual levels (Top and Akkermans 1994) described in Chapter 3 defines a way of organizing models which is different from the traditional approach, and it is depicted in Figure 4.1. Here, each ontological viewpoint corresponds to an information layer which has a strong internal cohesion and a relatively narrow coupling with other layers. The left part of the figure shows a physical model of the kind described by PHYSSYS. Arrows indicate the relationship between parts of the model and the model fragments in the library they can be chosen from.

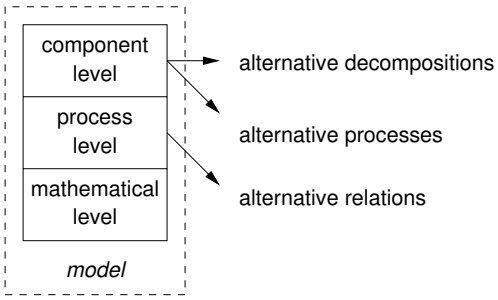


Figure 4.1: Modular structure of a model composed from model fragments in an engineering library, based on the separation of ontological viewpoints. The arrows indicate links to alternative model fragments in the library that are available to revise the model.

When modelling in a top-down manner, first *components* will be identified, since they are linked to the `real' objects, more or less independent from physical processes considered. Components are decomposed into subcomponents until the internal layout of the subcomponents becomes irrelevant. Next, at the conceptual-physical level for each component the assumed physical concepts or processes are described. This description is independent of the internal layout of the components, but does depend on the underlying physical assumptions made. Finally, the resulting mathematical equations are specified and processed for computer analysis and simulation.

Once a fully instantiated model has been constructed, the result can be assessed by means of analysis of the model or the simulation data derived from the model. This may lead to the conclusion that the model, or parts of the model are inadequate. In such a case, the modeller will revise the model by choosing alternative model fragments from the library for parts of the instantiated model. The model granularity can be changed by choosing alternative decompositions, alternative process descriptions may contain additional secondary physical processes that were neglected first and alternative relations may be non-linear instead of linearized. This approach to structured engineering modelling is depicted in Figure 4.2; an associated modelling support system prototype called QUBA has been developed and is described in (Top and Akkermans 1994).

Each ontological level goes with its own characteristic questions that have to be answered by the modeller:

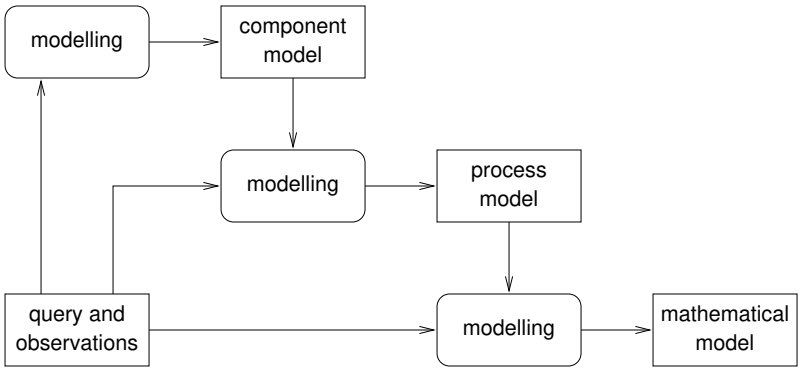


Figure 4.2: Evolutionary approach to physical modelling, with task decomposition based on ontological differentiations.

- Component level: Out of which concrete artefacts (device components) does the system that is to be designed exist, and how are they interconnected (system structure or layout)?
- Process level: How is the behaviour of the system components *realized* in terms of physical mechanisms?
- Mathematical level: How is the physical behaviour formally specified in terms of equations, such that system analysis and simulation can be carried out by computer?

Thus, in practice it will be quite evident during the modelling process what to put on what level, and when to transit from one level to another.

Given the modelling decisions described above, a model library should contain the following kinds of model fragments:

- A sufficient number of top-level components to support modelling in a certain domain and facilities like a taxonomy to help the modeller to find the right components.
- Different options for decomposition of the top-level components into smaller ones.
- Different physical process descriptions that are possible for a component. Usually, these descriptions have in common that they model one primary physical process and are only different in the secondary processes that are modelled.
- A physical process can often be described by different mathematical relations. Which one has to be chosen often depends on structural properties or the operating conditions of the modelled system. The library must therefore contain relations for each situation.

In the next sections, the general structure of the library will be explained by following the conceptual schema that was used as the basis for the implementation of the library. Model fragments from the thermodynamic domain will be presented as an illustration.

4.1.2 Structure of the Library

The above discussion represents a knowledge-level analysis of what engineering modelling and design actually `is'. In this section we will discuss how some of these aspects are being practically implemented in the OLMECO library for models of mechatronic design components. The core of the OLMECO software is a conventional (OO/relational) database for storage and retrieval of mechatronic model fragments; we will give an impression of its structure by considering the most important parts of the conceptual schema of the database. For more details, see (Top, Breunese, van Dijk, Broenink, and Akkermans 1994) and for some examples of its use see also (Top, Breunese, Broenink, and Akkermans 1995).

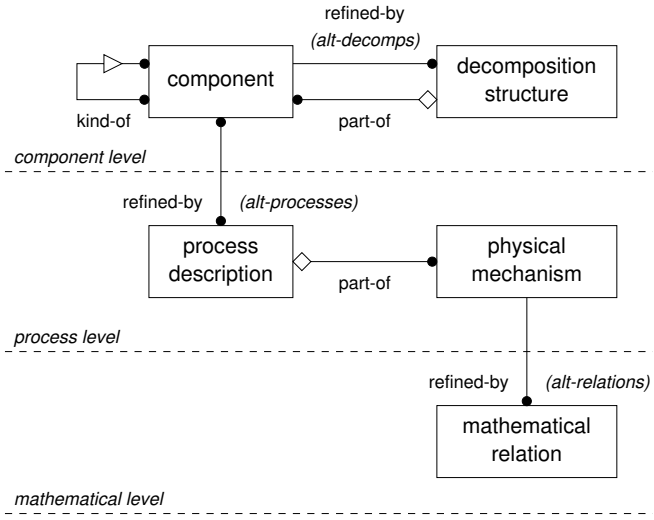


Figure 4.3: The general conceptual schema of the OLMECO library. It defines the database structure of the library and reflects the PHYSSYS ontology. rectangular boxes denote entity types or classes. Solid line connections, possibly with a diamond symbol carrying the relation name, stand for relations. Open and solid balls indicate the cardinality of the relation to be zero or one (optional association) and zero or more (one-to-many association), respectively. The triangle symbolizes the is-a or kind-of relationship (generalization/specialization), while the small diamond is employed for the part-of relationship (aggregation). Dotted lines are used to indicate constraints on/between entities or relations.

The OLMECO conceptual schema has been represented with the object-oriented modelling technique, called OMT, of (Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen 1991). The basic structure of the OLMECO library is shown in Figure 4.3.

It can be seen that the library structure follows the differentiation between ontological aspects, as discussed in the previous section. It is noteworthy that there are three different points, indicated by the *alt-...* links, where the user can make separate modelling choices. Systems can be decomposed in different ways, functions of which device components are the

carrier can be realized by different physical processes, and physical processes can be specified by different mathematical constraint expressions. Similar suggestions for structuring the modelling process not only come from AI, but are also proposed in the engineering literature (de Vries, Breedveld, and Meindertsma 1993).

The proposed generic structure of the OLMECO library has two important advantages:

1. It separates different groups of modelling decisions, thus giving handles for user support and facilitating a piecemeal approach to engineering model construction, and

2. It provides a breakdown of stored models into parts that have a generic nature, thus enhancing reusability and sharability of library models.

Component Taxonomy A component taxonomy can be stored in the library by means of the kind-of generalization/specialization relationship in Figure 4.3. This information can be used by the modeller to quickly access the components he or she wants to use. Figure 4.4 shows the taxonomy of the thermodynamic models in the library. For the components on the right, decompositions and/or process descriptions are available. Note that because the component taxonomy is meta-level information about components in instantiated models, the PHYSSYS ontology did not contain a formalization of it.

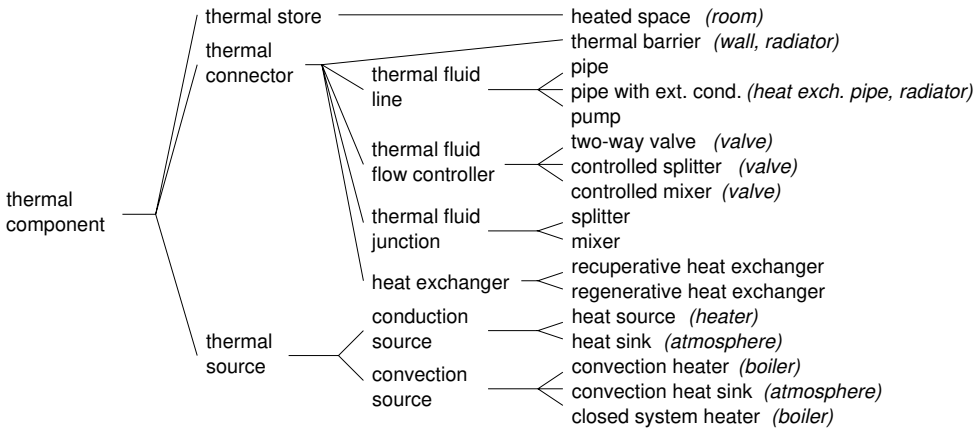


Figure 4.4: Taxonomy of thermal components. This taxonomy allows the modeller to quickly find components in the library. The library contains component models for the components on the right. Keywords printed in italics annotate specific situations in which a generic library component can be used, thus providing an alternative way to access components in the library.

The keywords printed in italics on the right cater for another way to index the library. The idea is that these keywords provide the link between specific component names used in the thermodynamic domain and the generic model components stored in the library. The keywords *wall* and *radiator* are examples of this. In both situations the *thermal barrier* component

can be used because it can model both heat flow through the metal of a radiator as well as heat flow from one side of a wall to the other.

For large libraries like the OLMECO library, the component taxonomy also becomes very large. A picture like the one in Figure 4.4 containing the model components of all OLMECO participants contains over 250 components and covers 16 pages! This shows that the taxonomy browsers in the library software must be able to cope with large amounts of components and be able to present parts of this taxonomy in a clear way to the user.

Decomposition Structure In physical modelling two types of decomposition can be distinguished. One type is the decomposition of components in pure *physical* subcomponents. An example of this is the decomposition of a *closed system heater* in Figure 4.5. A closed system heater is a device to heat circulating water. In the domain of thermodynamic systems, closed system heaters may, or may not include a pump that causes this circulation of the water. This has led to the three decompositions in the figure.

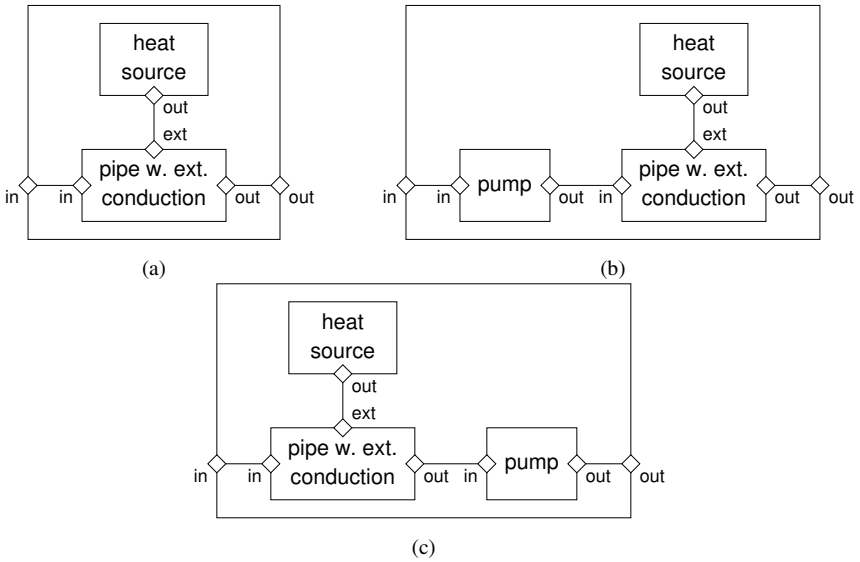


Figure 4.5: Decompositions of a closed system heater.

A second kind of decomposition establishes that physical models can be modelled more accurately. Because of the way physical processes are described, objects can only have a single value for state quantities like for instance temperature. This means that every part of for example a wall, is assumed to have the same temperature, instead of the continuous temperature distribution it has in reality. To approximate the real situation more accurately, the wall can be mentally decomposed into a number of wall *segments* which all can have a different temperature. This way, the continuous distribution will be approximated by a stepwise distribution. To facilitate this kind of decomposition, the library contains decompositions like the one in Figure 4.6.

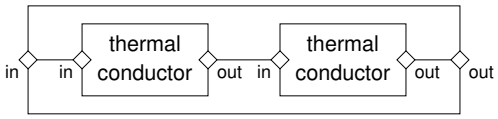


Figure 4.6: A possible decomposition of a thermal conductor.

The most apparent difference between the OLMECO library structure and PHYSSYS is that the first contains *decomposition structures*. In PHYSSYS it was sufficient to associate subcomponents to their parent with the `comp-subcomp` relation. For the library however, the relation between a generic component and all alternative ways for decomposition must be stored. Therefore, the concept of a decomposition was introduced. Thus, the association between a component and a possible subcomponent is made in two steps: first from the component to a possible decomposition structure (`alt-decomp`) and then from this decomposition structure to the subcomponent by a `part-of` relation.

Physical Process Description Figure 4.7 shows two process descriptions that can be used to model a pipe component. The process descriptions are presented in a graphical way using bond graphs. Bond graph representations are used in the modelling tool to display process descriptions in the library and as a way for the modeller to construct and edit instantiated process descriptions graphically. In a bond graph, the nodes represent physical mechanisms connected by half arrows, called bonds, which represent energy flows. The nodes are mnemonics indicating the type of the mechanism they stand for. A C node (short for capacity) for instance, indicates a store of stuff. Table 4.1 lists all bond graph nodes and the corresponding mechanisms. Full arrows in the bond graphs represent *information* flows which have not been formalized yet in PHYSSYS.

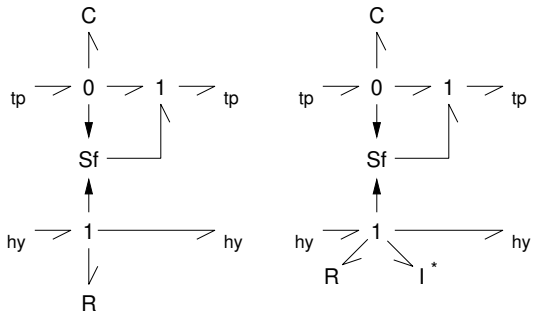


Figure 4.7: Bond graph process descriptions of a hot water flow through a pipe. A bond graph is a graphical representation of the process ontology as formalized in PHYSSYS. The nodes represent physical mechanisms of different kinds which are connected by energy flows. Both process descriptions model heat convection, whereby *(b)* contains an additional, secondary effect (the I node) due to the inertia of the water.

node	short for	mechanism
Se	source of effort	effort source
Sf	source of flow	flow source
C	capacity	stuff store
I	inertia	action store
R	resistance	dissipator
TF	transformer	transformation
GY	gyrator	gyration
0	-	flow distributor
1	-	effort distributor

Table 4.1: This table lists the bond graph nodes and the names of the mechanisms they represent. The nodes are mnemonics for the entries in the third column of the table.

Usually, the only difference between the process descriptions that model a library component are the secondary physical processes that are included. The process descriptions in Figure 4.7 for example, both model the *convection* and *hydraulic resistance* processes. The description on the left does not include *hydraulic inertia* (the process that makes your water pipes at home bang when you close the tap abruptly) whereas the one on the right-hand side does. This can easily be seen by the fact that the process description on the right includes an additional I node.

The relation between components and the alternative process descriptions that can model their behaviour can be found in the conceptual schema as the `alt-processes` relation that crosses the boundary between the component level and the physical process level. Note the analogy between this relation and the `comp.proc` relation in PHYSSYS. The difference is that in PHYSSYS the relation is between a component and one process description that models the component whereas `alt-processes` associates all alternative process descriptions to a generic component.

Mathematical Description For a number of reasons, a physical mechanism can be described mathematically in numerous ways. These reasons have been listed below, with examples from the thermodynamic domain. Some of the examples below concern the relations for hydraulic resistance (the R element in Figure 4.7). These relations can be found in Figure 4.8.

The reasons for different mathematical relations are:

Domain and nature of the process: Hydraulic friction requires relations different from those for mechanical friction. Different kinds of mechanical friction are described by different formulas.

Geometric and material properties of the modelled system: The relation for hydraulic friction in pipes with a smooth surface is different from the relation to be used in case of a rough surface (see Figure 4.8).

Operating conditions of the model: When the volume flow of water becomes higher, the nature of the water flow changes from *laminar* to *turbulent*. Hydraulic resistance for laminar flows is described differently compared to resistance in the turbulent case. This can be seen in Figure 4.8 where the Reynolds number Re quantifies the degree of turbulence. The simulator should check whether all relations are used in their range of validity and stop in case of a violation. Ideally, the violated relation could be replaced and simulation resumed.

Mathematical simplifications: The numerical simulation algorithm used may pose restrictions on the mathematical relations. This may require, for instance, linearizations of equations around a certain working point. In these cases, the same consideration for the validity of linearized relation holds as for the operating conditions mentioned above.

Hydraulic resistance

$$p = \xi \frac{l}{d_i} \frac{\rho}{2 A_f^2} V' |V'| \tag{4.1}$$

$$Re = \frac{V' d_i}{\nu A_f} = \frac{V' d_i \rho}{\eta A_f} \tag{4.2}$$

$$A_f = \frac{1}{4} \pi d_i^2 \tag{4.3}$$

Smooth pipe surfaces

$$\xi = \begin{cases} \frac{64}{Re} & Re < 2300 \\ \frac{0.3164}{\sqrt[4]{Re}} & 3000 < Re < 10^5 \\ 0.00540 + \frac{0.3964}{Re^{0.3}} & 2 \cdot 10^4 < Re < 2 \cdot 10^6 \end{cases} \tag{4.4}$$

Rough pipe surfaces

$$\xi = \begin{cases} \frac{64}{Re} & Re < 2300, K \leq 0.07 \\ \frac{1}{(2 \cdot 10 \log [d_i / K] + 1.14)^2} & \text{fully turbulent flow} \end{cases} \tag{4.5}$$

Figure 4.8: Some examples of mathematical relations that can be used to describe hydraulic resistance.

It is important to realize that all relations for a mechanism must conform to the constraints imposed upon them by the projection of physical processes onto mathematical relations defined in PHYSSYS. The ontology does not define the exact relation to be used, but only says what *class* of mathematical relations it must belong to. In theory, the library software could check these constraints, but to be able to check all possible relations requires powerful computer algebra.

In the conceptual schema of Figure 4.3, one or more mathematical relations are linked to a mechanism by the `alt-relations` relation crossing the line between the physical pro-

cess level and the mathematical level. The difference with the `mech.mathrel` relation in PHYSSYS again is that in an instantiated model a mechanism is described by *a single* relation but a generic mechanism can be described by one of *many* relations.

4.1.3 Instantiated Models

Software to support the modelling of physical systems consist of three main parts: a *model library*, a *modelling tool* and a *simulator*. The model library software allows the modeller to browse through the library and select the appropriate model fragments. The selected model fragments can then be assembled into an instantiated model using the modelling tool. Output of the modelling tool is the set of (differential) equations describing the behaviour of the modelled system. The simulator can accept these equations together with parameter values and arguments for the simulation algorithm (such as step sizes) and compute a simulation that predicts the behaviour of the modelled system.

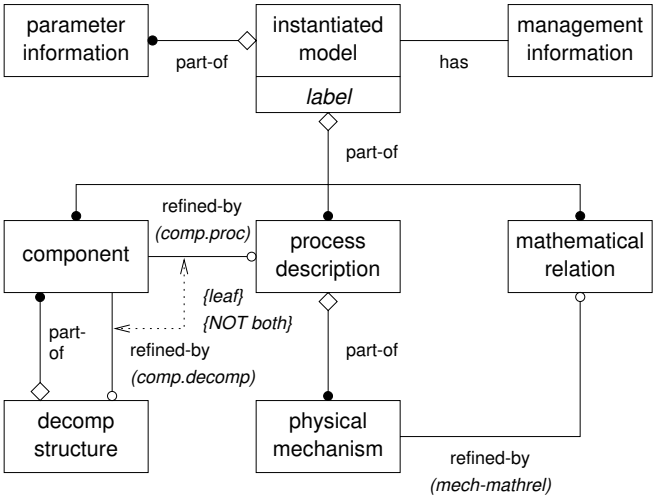


Figure 4.9: User-constructed instantiated models in the OLMECO library.

Up to now, only the general relationships that can exist between library model fragments were discussed. The situation is slightly different when we consider the user-built application models the modelling tool works with, since they represent an assembly of instantiations of generic model fragments. An instantiated model is a model constructed by an individual user or user group in the context of a specific application. The conceptual schema of an instantiated model is given in Figure 4.9. It contains references to the three different levels: (i) it identifies a selected component and the selected part-of hierarchy; (ii) it gives a set of bond graphs that collectively provide the physical description of the selected component, (iii) it gives a set of mathematical descriptions that describe the behaviour of each of the associated bond graph elements. Moreover, an instantiated model contains (iv) parameter

information. Note that the lower part of Figure 4.9 could almost serve as a conceptual schema of the PHYSSYS ontology. The only exception is the decomposition structure, which was introduced in OLMECO to store alternative decompositions and, for practical reasons, is also used in instantiated models. In instantiated models, it is completely equivalent to PHYSSYS' `comp.subcomp` relation.

In order to appreciate the differences between an instantiated model and the generic building blocks of the library, it is helpful to compare the schema of Figure 4.9 with Figure 4.3. Whereas the generic library fragments allow one-to-many refinement links —representing multiple modelling alternatives— instantiated models only contain one-to-one links corresponding to the specific selection made among alternatives. Furthermore, it is possible to have instantiated models that do not contain certain layers of information (see the solid balls in the schema). This is important to provide flexibility for the end user and to enhance interoperability with external software.

Namely, instantiation can be partial, *e.g.* if not all parameter data in the mathematical descriptions are specified. What type of instantiation is needed, depends on the requirements and capabilities of the external tools. Strictly speaking, for mathematical and computational purposes (analysis and simulation) only the mathematical description would be needed, and the component and/or physical levels of the model might be left empty. According to the conceptual schema, this constitutes a legal instantiated model — although perhaps not a preferred one from the viewpoint of structured modelling and sharability of models. (Note that even the empty model represents a legal instantiated model). This guarantees the interoperability with the external tools working with the library. For example, the OLMECO conceptual schema makes it even possible to work with simulation tools that don't know anything about components and bond graphs (*e.g.* ACSL or other Fortran-based mathematical software). All other information about components and bond graphs then provides *confidence building* documentation that can be used to backtrack the model construction process and to find suitable alternatives.

Components, decomposition structures, conceptual physical descriptions and mathematical descriptions know by which instantiated models they are used. Furthermore, each instantiated model has a user-definable label or keyword. This provides an important search facility.

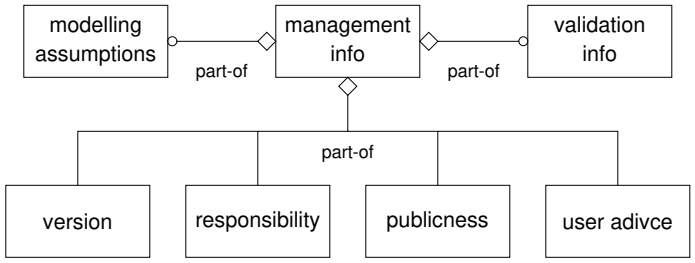


Figure 4.10: Model management information in the OLMECO library.

Finally, end user requirements with respect to the library have clearly pointed to the need for handles for knowledge management in sharing and reuse. Figure 4.10 shows a schema for this kind of information in the OLMECO library. The management information attributes: version, responsibility and publicness, represent quite straightforward database administration aspects. *Version* contains the information about which version of the model one is dealing with, and possibly about how and why the model has become what it is (the revision or update history). *Responsibility* refers to the owner(s) or administrator(s) who is/are responsible for the stored model information. The *publicness* attribute yields the status of the model as, say, a generally accepted one (within the user organization) or as a private `exercise'. This attribute might be used to introduce certain quality levels with respect to models and their degree of validation. *User advice* contains miscellaneous comments for using the model (such as hints for simulation algorithms or step sizes in tricky cases).

There are two management information elements that deserve special mention, because they contain crucial meta-level information regarding model construction and use:

Validation information: Any information that explains how the model has been or can be validated: literature references, measurement data, etc.

Modelling assumptions: The conditions under which the model is (not) applicable.

Both are very important attributes, that will need further attention in the future. Currently, the modelling assumptions and validation information are simply given in textual format, and thus comprise qualitative annotations concerning the model. Ideally, however, this meta-level information should be formalized in such a way that a support system for engineering modelling could actually *reason* about it. Steps in this direction have been made by (Addanki, Cremonini, and Penberthy 1991) in the so-called GoM approach, which contains some of this meta-level information, albeit in hardwired form.

To see whether the OLMECO library is able to support engineers in real-life situations, the library has been filled with model fragments and modelling and simulation experiments have been carried out.

4.2 The Thermodynamic Models in the OLMECO Library

To test the usefulness of the OLMECO library, each project partner has filled the library with model fragments for their domain of expertise and carried out a large scale modelling experiment. For the automotive domain, this has resulted in models for car bodies, gear boxes, ABS systems, hydraulical power steering, windshield wipers and electrical car components. Other partners in the consortium have modelled machine tools such as lathes, presses, milling and grinding machines. We have contributed to the library with thermodynamic models for components like pipes, valves, splitters and mixers, heaters and heat exchangers. Furthermore, the library contains models of electromagnetic transducers and general models of, for

instance, electrical components and mechanical rigid bodies. Table 4.2 gives an impression of the number of model fragments of the OLMECO library.

domain	process descriptions	mathematical relations	total
general models	30	95	125
rigid bodies	35	15	50
machine tools	55	54	109
thermodynamics	124	26	150
automotive	10	8	18
transducers	34	7	41
total	288	205	493

Table 4.2: Some statistics about the model fragments in the OLMECO library. For each domain, the the number of process descriptions and the number of mathematical relations are given.

We already have seen some of the thermodynamic models in examples used in the previous sections. It would take too much space to describe every thermodynamic model fragment in detail in this section. Instead, we will describe the most important design considerations for the thermodynamic models. The importance of these considerations lies in the fact that they concern constraints imposed on the models by the numerical simulation method used, so they are task specific extensions of PHYSSYS. A full description of all thermodynamic models can be found in (Top, Borst, and Akkermans 1995) and AppendixB.

The model fragments for convection in pipes are used to demonstrate the design considerations for the thermodynamic library models. We will see that to be able to make accurate models of pipes, we must place a number of models of pipe segments in a row. The problem that is encountered is that when the models of two pipe segments are connected, this leads to a set mathematical relations that cannot be handled by most numerical simulation methods. The problem is not so much that the constructed models are physically wrong, but rather that the generated set of mathematical relations contains two identical relations for one variable. In fact, one of these relations can be removed from the set of equations, but the problem is that this requires computer algebra. Therefore, another solution had to be found, as will be described next.

4.2.1 The Problem with Models for Convection

The problem occurs in components in which the physical process of thermal convection takes place. In the process of convection, thermal energy that is contained in material (usually a gas or liquid), is transported by movement of the material.

Figure 4.11 shows a physical process model for convection through a pipe. To obtain an accurate model, the pipe has been divided into three segments (also called lumps). For each segment, the heat stored in the segment is modelled by a C element and the hydraulic resistance the water undergoes by an R element. When the water flows out of one segment into

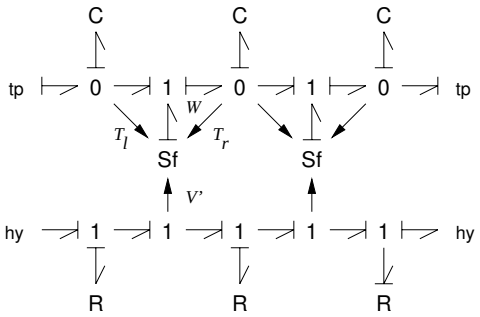


Figure 4.11: Pseudo bond graph for convection with three lumps.

the next, it will carry an amount of heat along with it. This is modelled by the convection Sf element. This element enforces a heat flow from a segment into the next. The magnitude of this heat flow W depends on the temperature T of the water flowing out the segment, the magnitude of the water volume flow V' and material properties of the water (ρ and c_v). The exact relation is $W = Tc_v\rho V'$, where $T = T_l$ when $V' > 0$ and $T = T_r$ when $V' < 0$ and c_v and ρ are the specific heat and specific mass of the fluid.

To support accurate modeling of components with convection, the library must support that models like the one in Figure 4.11 with arbitrary numbers of lumps can be constructed. This is because a model for convection with one lump assumes that at every point along the pipe, the water has the same temperature. With three lumps, like in Figure 4.11, the model is already more accurate, because each lump can have a different temperature. For longer pipes however, more segments are required for an accurate model. Therefore, the thermodynamic library must contain components, decompositions and bond graphs that can be put together to form models like in Figure 4.11 with arbitrary numbers of lumps.

4.2.2 Ways to Support Segmentation

There are a couple of alternative ways to support segmentation. The most straightforward way to do this is to make separate bond graphs like Figure 4.11 with $1, 2, \cdots, n$ lumps. This has the disadvantage that the number of models for convection in the library has to be very big.

A better option is to make bond graphs for pipe segments with one lump and provide decompositions of a pipe component into $1, 2, \cdots, n$ segments. This solution has the advantage that the segments are modeled explicitly at the component level, the level that should describe which tangible parts of the system are important for the model, but the problem of the large number of models has not disappeared. It has just been shifted from the process models to the decomposition structures.

Obviously, the solution is still not very practical and therefore, in the OLMECO library, a bond graph for a pipe component contains only one lump and a decomposition of a pipe component

in two others has been included. By recursive decomposition, pipes with arbitrary number of segments can be formed.

Now we only have to design an appropriate process model for a pipe segment. There are however some considerations that have to be kept in mind when these models are designed. They concern the way large models can be constructed out of instantiated library model fragments.

4.2.3 Design Considerations

The only constraint that OLMECO puts on the way model fragments are connected together to form a large model is the type of the terminals in the component model. For a hydro-thermal connection between two components this implies that the bond graphs of these components must be connected by a hydraulic and a thermal bond. No other constraints apply. As a consequence, when two models of convection (e.g. pipes) are connected together, it is *not* assumed that the amount of heat that flows through this connection is properly related to the water volume flow and the temperature of the water (according to $W = Tc_v\rho V'$). The process models of the segments must take care of this. This is called the principle of *independence*.

A second concern is that the models for passive components such as pipes should be *symmetrical*: because the interfaces of a process model for a pipe are identical, the orientation of the pipe in the larger model should not matter.

Next, we will discuss how given these properties usable process models for convection can be designed.

4.2.4 Dealing with Independence

Basically, when we consider the model in Figure 4.11, we see that the lumps consist of independent models for the thermal and the hydraulic domain that are connected by the convection Sf elements. To comply to the principle of independence, a process model should include two convection elements in the bond graph like in Figure 4.12*a* because only this will ensure that, no matter what is connected to either side, the thermal and hydraulic energy flows from and to the lumps will always be properly related to each other.

A problem is, that when two lumps (or two components with convection in general) are connected, it leads to two Sf elements that impose exactly the same constraint on the hydraulic and thermal energy flows between the segments. The only way that the modeling tool can conclude that one of them is redundant and can be removed, is to infer that both elements always impose the same heat flow. Unfortunately, the algebraic comparison of mathematical relations that this requires is too much asked for most of these tools.

As a result, when the set of mathematical relations are passed to the simulator, it detects that there are *two* relations that define *one* quantity. Because the simulator fails to detect that the values are always the same it will abort with an error message.

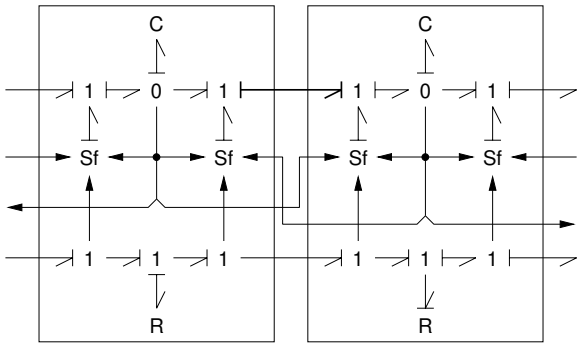


Figure 4.12: Symmetric bond graph for convection

Without computer algebra, the problem can only be solved by including just one Sf element in a lump model, like in figure 4.13. Unfortunately this results in asymmetrical models. When the right-hand side segment of Figure 4.13 is mirrored, the same problem as in Figure 4.12 occurs. We therefore have to find a way to deal with asymmetry.

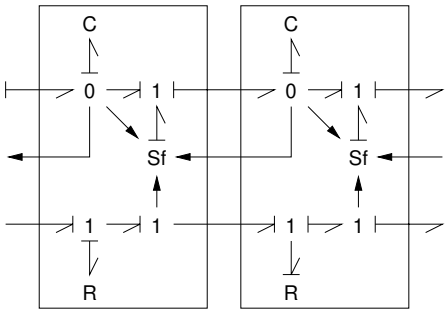


Figure 4.13: Asymmetric bond graph for convection.

4.2.5 Dealing with Asymmetry

Dealing with asymmetry requires an extra modeling assumption to assure that segments are combined as indicated in Figure 4.12*b*. There is no way to handle this assumption explicitly in the OLMECO library, so a trick has been used to handle it implicitly at the component level. Every hydro-thermal plug for which the hydraulic and thermal energy flows are related by $W = Tc_v\rho V'$ by an Sf element (the energy flows on the right hand sides of the lumps in Figure 4.12*b*) are labeled *out*. Plugs for which it is *assumed* that the energy flows are properly related (the left-hand side energy flows) are labeled *in*. By *requiring that in-plugs may only be connected to out-plugs*, it is ensured that models containing convection are constructed in the right way.

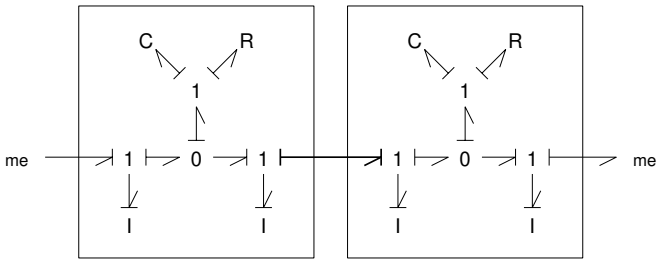


Figure 4.14: Bond graphs for two segments of a torsional shaft.

To show that the problems we have described above are not specific to the thermodynamic domain, we would like to mention the models for torsional shafts described by Wilson and Stein (1992). When a torsional shaft is modelled, segmentation has to be applied when multiple modes of vibration have to be taken into account. Independence and symmetry considerations lead to models with lumps like in Figure 4.14. These models bear the same problem of two relations prescribing the value of one quantity, just like the models for convection.

In the next section we will investigate the usability of the OLMECO library and the thermodynamic model fragments it contains.

4.3 A Modelling and Simulation Experiment

The modelling experiment for our domain, thermodynamic systems, consisted of the modelling and simulation of a large central heating system. This section describes this experiment. The experiment has been performed to find answers to the following two questions:

1. What is the practical usability of the library?
2. Does the thermodynamic library form a sufficient basis for the modelling of real thermodynamic systems from the point of view of reuse?

4.3.1 The Schieland Hospital Heating System

The subject of the experiment is the modelling and simulation of the existing heating system of the Schieland Hospital, a general hospital in Schiedam, The Netherlands. The schematic drawing of the system that has been modelled is given in Figure 4.15. Clearly, the system consists of two coupled subsystems: one subsystem around the heater (heater group, abbreviated hg) and one around the radiator (radiator group or rg).

Compared to simulation models used in the design of thermodynamic systems in engineering, the model designed here can be characterized as being large. The model contains a large

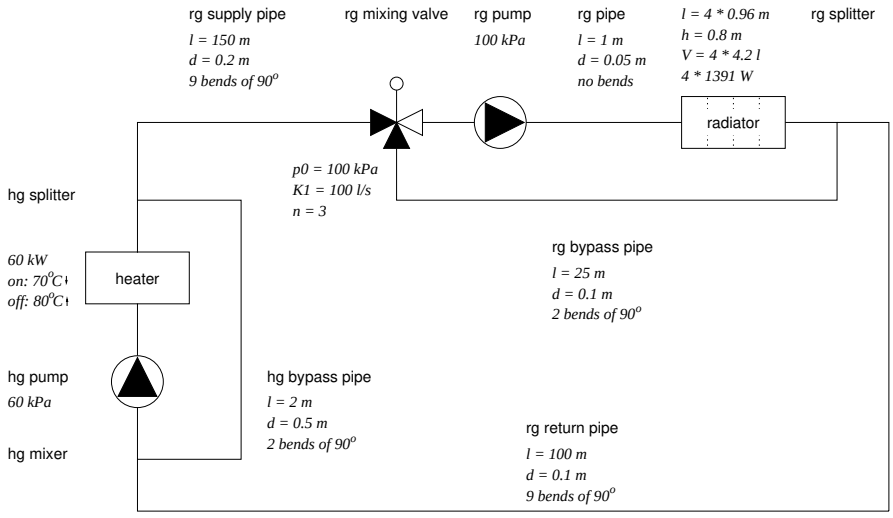


Figure 4.15: The simplified Schieland Hospital heating system.

number of components from the thermodynamic library and the nature of the system results in a complex mathematical model. The fact that both hydraulic and thermodynamic behaviour are modelled also contributes to the complexity. The model statistics in Table 4.3 give an impression of the complexity of the model.

statistics about the model	
19	model components
1	user defined component
18	components from the library
9	component classes from the library
4	decompositions from the library
26	coupled differential equations
$\pm$ 150	equations

Table 4.3: Some statistics on the model of the Schieland hospital heating system.

4.3.2 Modelling the System

The model of the system has been incrementally constructed in three stages. First the component model has been made, then the physical model and finally the mathematical model. For a detailed description of this we refer to Appendix B. In this section, only a global description will be given.

The Component Model The first step in the modelling of the system is to construct the component model using generic components from the thermodynamic library. This has resulted in the model shown in Figure 4.16. Names of the instantiated components are printed in bold face. All components, except for the controller, are instantiations of generic library components. The name of the library component a model component is instantiated from is printed in italics at the top left corner of a component.

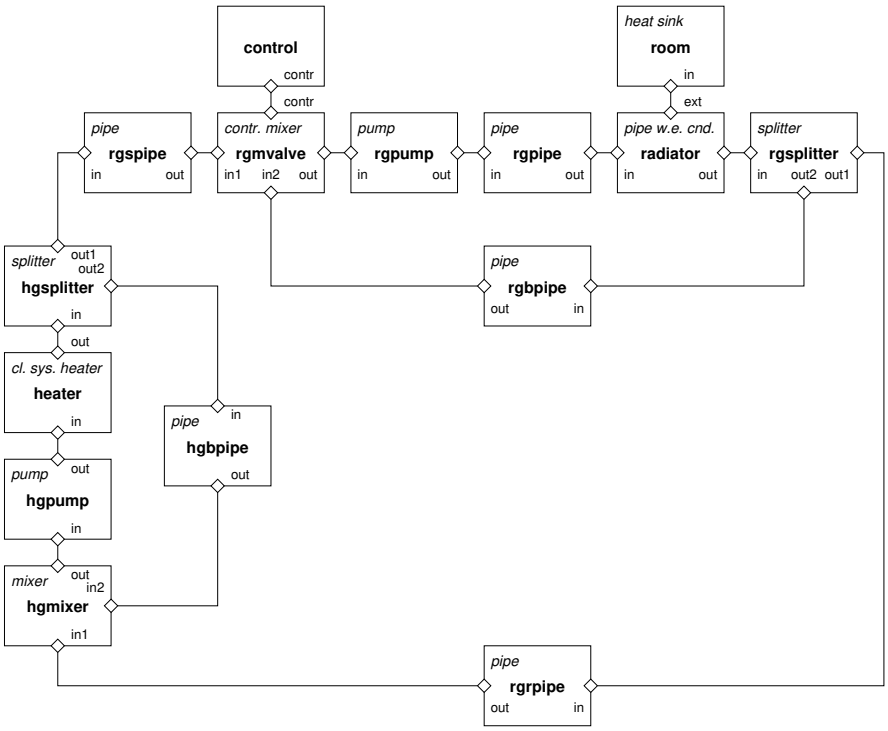


Figure 4.16: Component model of the Schieland Hospital heating system. All components but one are instantiated library components. The names of the generic library component a model component is based on is printed in italics. Note the close similarity to the schematic drawing of the system.

Three components need further explanation. Because the *heater group pump* is considered to be a top level component and not a subcomponent of the heater, the heater has a decomposition according to Figure 4.5 (a). The *controller* is a user defined component (i.e. not based on a library component) that supplies to the controlled mixing valve the information whether it is open, closed or partially open. The *radiator* has been modelled as a pipe with external conduction. In such a `pipe', the water that runs through it can lose its heat through the wall of the pipe to the environment, in this case the room. The abstraction from radiator to pipe with external conduction has been specified in the library taxonomy by means of the radiator keyword attached to the pipe with external conduction component (see Figure 4.4).

Furthermore, to increase the accuracy of the model (see Section 4.1.2), the radiator has been decomposed into four segments.

The Physical Process Model For the construction of the physical models, the physical processes that have to be modelled in order to obtain an accurate model must be chosen. Table 4.4 gives an overview.

component type	physical processes							
	heat convection	heat storage	hydraulic resistance	hydraulic inertia	thermal resistance	pressure source	heat source	temperature sink
pipe	√	√	√	√				
pipe w. ext. cond.	√	√	√	√	√			
splitter	√	√	√					
mixer	√	√	√					
controlled mixer	√	√	C					
pump	√	√				√		
heat source							C	
heat sink								√

Table 4.4: Table of modelled physical processes for each type of component in the model. Each row in this table is covered by a physical process description from the library that models the indicated processes. A *C* instead of a checkmark indicates a controlled process.

The importance of this table is that for each row in this table, there must be a process description in the library that models all the physical processes that are marked. For example, the process description used to model the pipes in the system can be found in Figure 4.7 (b). For all rows in the table the library contains model fragments that take into account the specified physical processes.

The Mathematical Model For most of the processes in the physical model there is only one mathematical relation applicable. Only for the hydraulic resistances in the pipes, splitters and mixers and the thermal resistance of the heater and radiator important choices had to be made. For the hydraulic resistances for instance, the relations for pipes with a rough surface and turbulent flow from Figure 4.8 were used. All relations required for the model were available from the library.

Determination of Model Parameters Before a model can be used for simulation, the values of the parameters in the model need to be determined. The way this has to be done can be found in engineering handbooks like the VDI Wärme Atlas (Verein Deutscher Ingenieure 1977), an atlas of relations for heating systems written by the society of German engineers. The relations this handbook gives for the determination of the model parameters depend on different types of data about the system. The required data can be classified as either *characteristic values of materials*, *geometric* or *measured* data.

Characteristic values of materials, like specific mass, specific heat capacity and heat transfer coefficients can be found in engineering handbooks on materials. Geometric data includes

values for volumes, areas, angles of incidence etc. For most components, these values are easy to calculate from lengths, thicknesses and radii. Some mathematical relations use measured values of properties of the modelled component, measured under standardized conditions. An example is the heat transferred by a radiator to air of $20°C$ when the incoming water has a temperature of $90°C$ and the outgoing water a temperature of $70°C$. Usually, the component manufacturer supplies these values. Some of the parameters required for the Schieland hospital model are shown in Figure 4.17. A full description of the process of determination of the parameters can be found in Appendix B.

Pipes, splitters, (controlled) mixers, radiator and pumps (16 in total): specific mass, specific heat capacity and viscosity of water; volume of the water and initially stored heat; length, diameter and water flow area of the component; roughness of the material the component is made of.

Pipes with bends (4 pipes, 22 bends): number and sharpness of the bends.

Radiator (4 segments): heat transfer at 90/70/20°C.

Controlled mixing valve (1): minimum and maximum volume flows at a pressure of $10^5 Pa$.

Splitters (2) and mixers (1): water flow areas and angles between in and out flows.

Pumps (2): supplied pressure.

Heat sources (2): supplied heat flow or temperature.

Figure 4.17: This figure gives an idea about the amount of information required to calculate the parameters in the model of the Schieland hospital heating system.

The relatively large amount of time it took to compute the parameters for the modelled system suggests that the next step to improve the support of engineers would be to help them with this process. This requires an extension of PHYSSYS and the OLMECO library to make it possible to specify the parameter relations from the atlas, characteristic values of materials, measurement data and geometry.

4.3.3 Simulation

Two simulations have been carried out, a simulation of the hydraulic behaviour of the system when the position of the valve changes and, secondly, a simulation of the thermodynamic behaviour when the heating system is switched on. The prediction of the thermodynamic behaviour can be found in Figure 4.18, that of the hydraulic behaviour in Appendix B. The most striking fact that can be observed is that it takes close to ten hours for the water in the system to heat up from room temperature up to the desired value of about 70°C. This behaviour is exactly what can be seen in reality with large heating systems like this. In the next paragraphs this behaviour will be qualitatively described.

Initially, when the heater is on, the heater temperature A will increase. At first it increases quickly because the water that flows into the heater is of almost the same temperature as

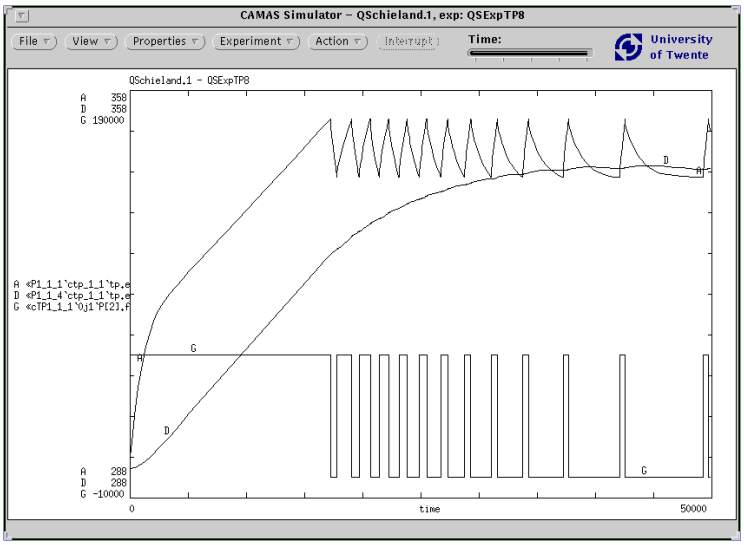


Figure 4.18: Predicted thermodynamic behaviour of the Schieland hospital heating system. The plotted values are A: the temperature of the heater, D: the temperature of the radiator pipe, both in K (Kelvin). G is the heat flow from the heater which has $0\ W$(att) as minimum, and $60\ kW$ as maximum value. The horizontal axis is the time scale in seconds.

the water that flows out of it. The net heat flow to the heater will then be approximately $60\ kW$. As the temperature of the heater increases, the amount of heat that flows out of the heater will become larger than the heat carried by the water that flows into it. The net heat flow to the heater will become smaller than $60\ KW$ and therefore the heater temperature will increase at a slower rate. When the radiator gets hot, the temperature of the water from the radiator group return pipe will increase and so will the temperature of the water that flows back into the heater. This will cause the heater temperature to increase at a constant rate until the maximum heater temperature is reached and the heater is switched off.

Next, the heater will be switched on and off repeatedly. The simulation shows that the periods that the heater is on become shorter and that the on-off interval becomes longer. This can be explained by the fact that the temperature of the water from the radiator group reaches a high value. Because of this, the net heat flow out of the heater will become smaller, so that it takes more time for the heater to cool down and less time to heat up again.

4.3.4 Findings from the Experiment

The first conclusion that can be drawn from this experiment is that the OLMECO library and the evolutionary modelling approach that are based on the conceptualization formalized in PHYSSYS, provide good assistance in the modelling process. This is reflected in amount of the time it took to construct the large and complex model of the heating system and the

quality of the result. Modelling the system took a small amount of time due to the fact that the library contained most of the required model fragments and to the fact that the model could be specified incrementally, starting with the component model which is very similar to the schematic drawing of the system. The other steps to processes and mathematics were guided very well by the suggestions the library contained for possible process descriptions and mathematical relations. The quality of the model fragments in the library contributes positively to the quality of the instantiated model.

The second conclusion is that the thermodynamic library is diverse enough to support compositional modelling of real world systems. The modelled system is considered to be large and contains a variety of components typical for the whole domain.

Furthermore, it would be nice to have a way to let the simulator check the validity domains of the submodels dynamically. The problem is that the validity of the model for hydraulic resistance for instance, depends on dynamic model variables like the volume flows. This makes it impossible to check the validity before simulation. In the present library, specification of the validity domain for models are pure textual annotations. To be able to store the more algorithmic checks, like the one for hydraulic resistance, these have to be formalized. So we need an active form of management of model assumptions.

The experiment carried out suggests an extension of PHYSSYS and the OLMECO library. This can be concluded from the time it took to determine the model parameters. Therefore we suggest an extension of the library in which it is possible to specify the way the parameters of a model component can be determined, like it is described in engineering handbooks. The parameter relations in the library could then be used for automatic parameter computation from geometric data supplied by the user. The present way to store parameter relations in the library is not sufficient because the parameter relations that have to be used can depend on geometric aspects of the component that is modelled. For instance, cylindrical and non-cylindrical pipes are modelled by the same component and the same equation for the hydraulic resistance, but the way to determine the parameters is different. This at least suggests a fourth view on the domain of physical modelling, that of geometry, and implies an additional ontology projection. The same holds for the material properties of components. In the next section, we will elaborate on this subject.

4.4 Parameters and Parameter Relations Determination

The OLMECO library gives a significant increase of modeling productivity and quality, at least concerning the dynamics of a system. However, as the example in Section 4.3 shows, in practice the parameters that determine dynamic behaviour are determined by other parameters that relate to geometrical, material and other data. For example, the constitutive relations for heat transfer through the wall of a pipe depend on geometrical (shape, length, diameter) and material properties (density, viscosity, surface roughness) of the components and fluids involved. Although the OLMECO library supports parameter relations in general, no structured framework for maintaining and retrieving these relations is defined yet. Consequently, the

bottleneck in the modeling process now is concentrated in finding and relating the proper parameters for each application. We claim that the introduction of the component level in the library facilitates a structured approach for storing parameter relations as well.

The first principle to be respected when setting up a more organized way to handle parameter relations is that *conceptual categories* must be respected. In other words, one should realize that *dynamic behaviour, geometry, material specification etc.* are different areas of information. Each of these areas specifies a certain view of the components in the system and requires its own tools. Different views should only be related through a well defined set of linking parameters. In the bond graph model the parameters for capacitance, resistance, inertia, transformation ratio, *etc.* are the linking parameters (interface) to geometrical and material data. Ideally, one should specify the geometrical parameters in a CAD-system and link them automatically to the simulator.

A second structuring principle is that parameters have different levels of generality. For example, we distinguish between constants and parameters. Constants are supposed to have a fixed value across many applications (*e.g.* the Stefan-Boltzmann constant), whereas parameters may differ from one simulation run to another. Note that there is another category of variables, which usually are referred to as *signals*, which do not have fixed values within a simulation run, but are not energetic variables (effort, flow, displacement etc.) either. In our view, signal variables should be considered as part of the dynamics domain, although they may depend on each other through relations from the geometrical or material domain (for example, the volume of a piston).

The component level of the OLMECO library provides a suitable framework for organizing parameter data and parameter relations. The idea is to consider different sets of variables and parameters as being part of different perspectives of a single component: energetic-dynamic, geometrical, material, etc. By disconnecting these perspectives, an independent selection can be made for each perspective. Consider for example the case of flow through a pipe. From the dynamics perspective the modeller must decide about which dynamic effects are relevant: friction, inertia. On the other hand, independently he or she must decide whether cylindrical or rectangular shapes are to be modeled. In particular for design problems it is useful to make these decisions separately and to be able to quickly change selections in one area (for example, from cylindrical to rectangular pipes) without affecting the other.

The component level in particular supports hierarchical structuring of parameters according to their degree of generality, or *scope*. Some parameters are common to all components in the system, others vary from one component to another. For example, in the heater example there is only one type of fluid, and this fluid is considered to be incompressible. Hence, the density ρ can be defined at the top level of the model. When setting up the actual simulation model (set of equations used by a solver), the system first checks for parameters at the lowest level (that of bond graph elements), than that of the components containing the bond graphs. This is repeated up to the highest level, until all parameters and parameter relations are known. Note that this can be considered as a form of *inheritance*, as defined in object oriented modeling (Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen 1991).

At the moment no commercial tools are available that support the component level properly. Hence, no support for maintaining parameters and parameter relations along the lines sketched above can begin yet. However, once such tools become available, the bottleneck of getting the proper parameter relations can significantly be reduced, in particular if an interface to CAD systems is realized.

4.5 Conclusions about OLMECO and Using PHYSSYS

In this chapter we have shown that it is relatively straightforward to derive a database specification for a library of model fragments that supports the construction of simulation models for engineering and design from the PHYSSYS ontology. We have demonstrated in a modelling experiment that the knowledge specified in PHYSSYS is adequate for simulation of real-life systems. Some minor changes and additions to the specifications in PHYSSYS were required because a model library should not only know which physical models are valid, but should also consider at what points the modeller makes decisions during the modelling process.

This has lead to a library structure with *model fragments* for generic components, decomposition structures, process descriptions and mathematical relations and *links* from components to alternative decompositions, from components to process descriptions and from mechanisms in these process descriptions to alternative mathematical relations. Furthermore, a component taxonomy has been introduced to support the retrieval of model fragments in an intuitive way.

A modelling and simulation experiment for a real life system showed that it is possible to construct valid simulation models. This proves the validity of the knowledge formalized in PHYSSYS.

It can also be concluded that the OLMECO library significantly increases modelling productivity an quality through reuse of validated model fragments. However, a bottleneck in the use of the OLMECO library has been the determination and specification of the model parameters for large models. To deal with this problem in the future we have shown that the component level is a good place for the specification of parameters on a global level, or on the level of subsystems. By means of parameter inheritance from systems and subsystems to components and process descriptions, the parameters are passed to the appropriate mathematical relations.

Due to constraints on the mathematical relations numerical simulation algorithms can work with, some restrictions on the use and design of the library model fragments were necessary. We have proposed two alternative ways to work around these problems: introduction of computer algebra and the introduction of constraints on the assembly of component models.

The work on the OLMECO library suggests several extensions of the PHYSSYS ontology. First of all, it could be extended with formalizations of the extensions that were required to design the OLMECO library: the idea of generic model fragments, the alternative modelling decisions and the component taxonomy. Furthermore, the proposed idea of parameter inheritance, the restrictions on the mathematical relations imposed by the simulation software, as well as the possible solutions to overcome these problems could be included in the ontology.

Chapter 5

Reusing and Extending an Ontology for Product Disassembly Analysis

In this Chapter we will investigate what happens when instead of an ontology for simulation of technical devices, an ontology for a totally different task in a different engineering domain is constructed. This new task is the ecological impact assessment of product disassembly. We will try to find an answer to the question whether it is possible to reuse parts of PHYSSYS*, or that we are forced to start from scratch.*

The organization of this chapter is as follows. Section 5.1 gives an introduction to ecological product disassembly analysis. In Section 5.2 we introduce the basic concepts needed to build product models for disassembly, and illustrate them by various examples. In order to test the validity of this theory and to provide a basis for a full ecological disassembly tool, the theory has been implemented in a knowledge based system called PROMOD. Section 5.3 describes the functionality and use of PROMOD. The way PROMOD has been implemented is described in Section 5.4. Section 5.5 gives some references to related research and discusses important issues regarding improvements of the theory and prototype. We conclude this chapter with a summary of the issues relevant for ontological engineering, and answer the question whether PHYSSYS could be reused.

5.1 Ecological Product Disassembly Analysis

In recent years, growing ecological concern has prompted for `design for environment' (Fiksel 1996; Umeda, Tomiyama, Kiriyama, and Baba 1995). One way to achieve this is to design products that are easy to disassemble, because this improves the ability to reuse or recycle

parts of a product. In analyzing these aspects, one needs to determine all feasible ways to disassemble a product. They can be jointly represented in an *AND/OR graph* (Fazio and Whitney 1987; Sturges Jr and Kilani 1992), with the fully assembled product as the root, the *and-nodes* indicating a disassembly operation splitting the product into subassemblies, and the *or-nodes* representing different applicable operations that give rise to alternative ways to break up a (sub)product. In such an AND/OR graph, each subtree that has and-nodes as its leaves, and contains exactly one out of the alternative branches at each or-node it contains, describes a distinct disassembly sequence.

Figure 5.1 gives an example of an AND/OR graph for a very simple product that is the root of the graph. Subassemblies form the and-nodes whereas black bullets mark the or-nodes. One of the alternative disassembly sequences has been marked in the figure using thick lines. Usually, identical subassemblies in an AND/OR graph are merged, thus forming a graph instead of a tree.

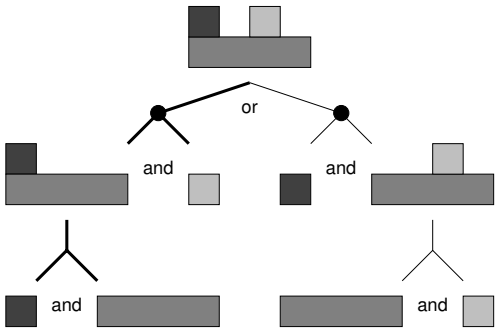


Figure 5.1: Disassembly sequences of a simple product visualized in an AND/OR graph.

In eco-design, an important goal is to determine the cost of disassembly as well as the environmental benefit of the parts that are separated. The energy required to perform the disassembly operations might be used as a measure for the disassembly cost. All parts that are separated in a disassembly process are candidates for recycling or reuse and have a positive impact on the benefits of disassembly. The degree to which the materials in a separated part of the product can be recycled is determined by the mix and the amounts of materials in that part. With such a cost-benefit analysis applied to the AND/OR graph, the disassembly sequence having the best cost-benefit ratio can be determined, as well as the impact of design decisions by comparing alternative product designs.

It is a common misunderstanding that the best *disassembly* sequence is simply the *assembly* sequence in reverse. It can be demonstrated that this is not the case by looking at Figure 5.1. If we assume that the assembly sequence of the product is indicated by the thick lines, and assume that the darkest block has to be separated for recycling, it becomes clear that the right branch in the tree is better than the left. To find the best disassembly sequence, we have to consider all possibilities. This becomes even more important when more than one part can be recycled or reused.

Therefore, in this chapter we will present a general ontology-based approach to two important tasks in product disassembly analysis:

1. automatically generating the AND/OR graph from a topological product model;
2. obtaining from the AND/OR graph the disassembly sequence having the best ecological cost-benefit ratio.

An ontological approach to this problem appears helpful, because it is evident in disassembly analysis that a notion of `connectedness' plays a crucial foundational role (Clarke 1981). As we will see, the standard topological relation, that expresses that two objects are connected or in contact, as incorporated in formal topological ontologies such as in 3, proves to be not adequate for this purpose. Nevertheless, we show that by extending a standard topological ontology with a small number of new ontological primitives regarding the *type* of connections, we can build product models that provide the knowledge to automatically carry out the mentioned tasks.

5.2 Theory of Product Models for Disassembly

In disassembly analysis, a product is conceived of as an *assembly* consisting of *objects* with mutual *connections*. In this section, the way product models for disassembly can be constructed with the basic model elements *objects* and *connections* is explained. We will see that other concepts like connection types and loops through connections must be introduced to be able to carry out a disassembly analysis. Section 5.2.5 discusses the way this analysis can be performed. In Section 5.2.6 more sophisticated model elements are introduced that allow complex product structures to be modelled.

5.2.1 Connected Objects

As has been mentioned before, a product is conceived of as an *assembly* consisting of *objects* with mutual *connections*. Objects are the smallest parts of a product that are relevant in a disassembly context. The most important attributes are the materials they are made of and the amount of these materials. With these attributes it is possible to determine to what degree the materials of a group of objects can be reused or recycled.

Connections specify the places of contact between objects. It is allowed that there are more than one connection between two objects. Connections have properties beyond standard topology that are important for disassembly analysis. When an object is situated between other objects, these objects may be in the way during disassembly. It can also be the case that the object in the middle can be removed by pulling it in another direction. But in order to do this, there may not be a rigid connection between the objects. Two properties of connections are therefore of importance: how *rigid* connections are (in the sense of physical forces) and how constrained (in a spatial or geometric sense) the *direction* of movement of objects is.

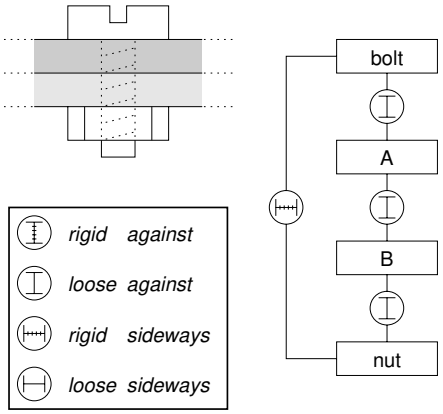


Figure 5.2: The four connection types and an associated disassembly model of a bolt-and-nut system.

Unfortunately, this implies that generally we have to deal with many, both geometric and chemical/physical concepts. This we want to avoid, practically because in the design stage of a product detailed models like 3D CAD drawings are often not yet available, and computationally because it involves strong and complex ontological commitments. However, it is possible in disassembly modeling and analysis to define task-oriented abstractions of geometric and physico-chemical connection properties that do the job. These abstractions are then brought into a topological ontology of disassembly models by means of different connection types.

Distinguishing four types of connections is already sufficient to be able to perform practically useful disassembly analyses. These types are based on a dichotomy within two important orthogonal dimensions. The first discriminating dimension is the mentioned *rigidness* of a connection. A useful dichotomy here is whether a connection is rigid or loose, and relates to the distinction of force-based versus shape-based connections:

rigid: A physical or chemical *force* keeps the objects together, so this force must be overcome first to break the connection. (Example: objects screwed or sticked together.)

loose: Objects are in contact with each other, but in a *purely spatial* sense without an additional binding force, so the connection disappears when the objects are moved apart. (Example: a glass standing on a table.)

The second distinguishing dimension is the *direction* in which a connection restricts the movement of the connected objects. The simplest possible conceptualization is to introduce the following dichotomy:

against: The connection restricts movement in the direction *perpendicular* to the surface of contact.

sideways: The connection restricts movement in a direction *within the plane* of the surface of contact.

This gives a two-by-two matrix, leading to four types of connections: *rigid-against, rigid-sideways, loose-against, loose-sideways*. Figure 5.2 depicts graphical representations of these types, and an example how these types are used in a product model for disassembly. When the rigid connection between the bolt and nut is broken (which requires applying a physical force), the loose-against connections in the model can be simple undone by moving the connected objects away from each other.

A straightforward way to give an ontological definition of connected objects is to construct it using the ontology of systems theory from PHYSSYS (Chapter 3). Objects are instances of individuals from systems theory and connections are typed topological connections. Because a model for disassembly is a *system* of disassembly objects, systems theory has been reused instead of topology. The ontology projection is of the include-and-specialize type. Figure 5.3 gives an idea of what the an ontology of disassembly models could look like.

```
1  define-theory disassembly-models

2  include-theory systems-theory

3  define-class d-object(x)
a    d-object(x) -> simple-m-individual(x)

4  define-class d-connection(c)
a    d-connection(c) ->
       connection(c) and
       connects(c,o1,o2) -> d-object(o1) and d-object(o2)
b    d-connection(c) ->
       d-con.type(c,a-loose) or d-con.type(c,a-rigid) or
       d-con.type(c,s-loose) or d-con.type(c,s-rigid)

5  define-class d-model(m)
a    d-model(m) -> system(m) and
                   in-system(m,o) -> d-object(o)
```

Figure 5.3: Proposal for an ontology for disassembly models (i). Disassembly objects and connections are defined in terms of system theoretic concepts.

Line 2 includes the ontology of systems theory which is projected onto disassembly models in Definition 3–4. Objects are defined as *atomic* mereological individuals because they are the smallest objects relevant for disassembly. Disassembly connections are topological connections between disassembly objects, and must have a connection type (Definition 4). Given these definitions, a disassembly model is a system of disassembly objects.

5.2.2 Force Loops

In the example in Figure 5.2, the two plates are in between the two bolt and nut. In this case, the direction of all connections are the same, but generally they are not. Objects can be restricted in different directions when they are in between several pairs of objects. We must therefore model which connections of an object are in the same direction, but do not want to do this using 3D vectors. Therefore, it is done using the topological concepts of *paths and loops* through connections. These paths encode the geometric information in an abstract

way. The paths can be constructed by linking the connections that are in one direction. In the model in Figure 5.2 we then see one loop running through all connections.

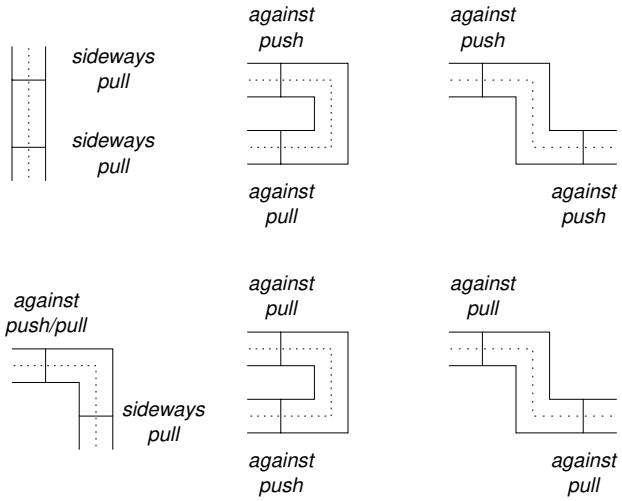


Figure 5.4: The possible transitions in force loops through connected objects in a disassembly model.

An easy way to determine the direction of connections and the paths that go through them is to imagine what happens when objects connected by a loose connection are pulled apart. For example, in Figure 5.2, when the connection between A and B is pulled, this implies that plate A is pushed against the bolt, the bolt is pulled away from the nut, the nut is pushed against plate B and finally (although this may sound as a contradiction) plate B is pushed against plate A. The outcome of this sequence of forces depends on the geometry of the product and, as we will see later, turns out to be exactly the information required for disassembly analysis.

In a rigidly connected product, i.e. a product of which all parts are firmly connected together, the geometric structure is such that the objects connected by loose-against connections are pushed against each other. This will result in paths through connections that make a full cycle, as is the case Figure 5.2. Therefore, we will speak of *force loops*. In cases where a path does not make a full cycle, we will say that the loop is broken or not *intact*.

Because the sequence of forces in a force loop depends on geometry, only certain force transitions are possible when a path is followed. This is shown in Figure 5.4[1]. These restrictions imply that the forces on connections in an intact force loop comply to the following rules:

- Loose-against connections are always pushed.
- Rigid-against connections are pushed or pulled.
- Sideways connections are always pulled.

[1] Note that torque has to be modelled as a combination of sideways and against connections.

When two objects connected by a loose connection are pushed together, we will say that the connection is *enforced*. This can only be the case when there is an intact loop going through the connection.

In more complex situations, there might be more than one connection in the same direction on one side of an object, like in the model in Figure 5.5. This can be modeled by two force loops going through one connection. In other situations, a group of connections in different directions collectively form a virtual connection in another direction. The way these situations can be dealt with will be explained in Section 5.2.6.

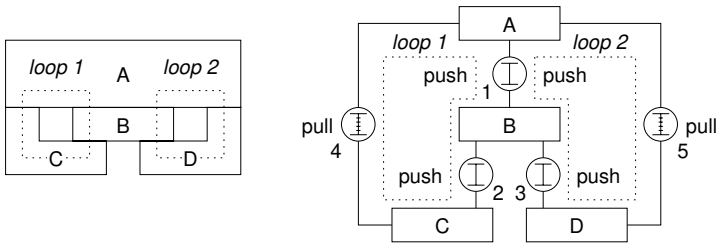


Figure 5.5: Each loop individually enforces the loose connection numbered 1.

For the definition of force loops it is convenient to look upon a disassembly model as being a graph (Skvarcius and Robinson 1986) in which the objects are the vertices and the connections the edges. In such a graph a force loop is a path or cycle. This approach is formalized in Figure 5.6.

```
5  include-theory graph-theory

6  define-relation model-graph(m,g)
a    d-model(m) -> exists g: graph(g) and model-graph(m,g)
b    model-graph(m,g) ->
       (graph.vertex(g,v) <-> in-system(v,m)) and
       (graph.edge(g,e) and edge.from-to(e,o1,o2) <->
        in-system(o1,m) and in-system(o2,m) and connects(e,o1,o2))

7  define-class force-loop(l)
a    force-loop(l) -> exists m,g: model-graph(m,g) and
                                  graph.path(g,l)

8  define-relation loop.conn.force(l,c,f)
a    force-loop(l) and loop.vertex(l,c)
       -> loop.conn.force(l,c,push) or
          loop.conn.force(l,c,pull)
```

Figure 5.6: Proposal for an ontology for disassembly (ii). Force loops can be looked upon as paths through a graph when connected disassembly objects are regarded as vertices connected by edges.

Line 5 includes an ontology of graph theory. This ontology defines a graph as a tuple of a set of vertices and a set of edges. Edges can be either directed or undirected, which leads to the concepts of undirected graphs (or simply *graphs*) and directed graphs (or *digraphs*).

Furthermore it defines *paths* and *cycles* through the edges of a graph. We also assume that it defines the classes of *connected graphs* in which there is a path between every two vertices in the graph and *unconnected graphs* where there this is not the case. A graph is a *subgraph* of another graph when the sets of vertices and edges of the first are subsets of the second.

In Definition 6 is stated that each disassembly model must be related to a graph where the vertices are the objects in the model and the edges the connections. An example of the `model-graph` relation, instantiated for a simple model, can be seen in Figure 5.7. A force loop can then be viewed upon as a path through the connections of a model-graph. The relation `loop.conn.force` specifies the force a loop exercises on each connection in the loop. The projection of graph theoretical paths to force loops makes sure that the connections in a force loop form one continuous chain. The dependency between the type of a connection and the allowed forces on it have not been formalized in Figure 5.6.

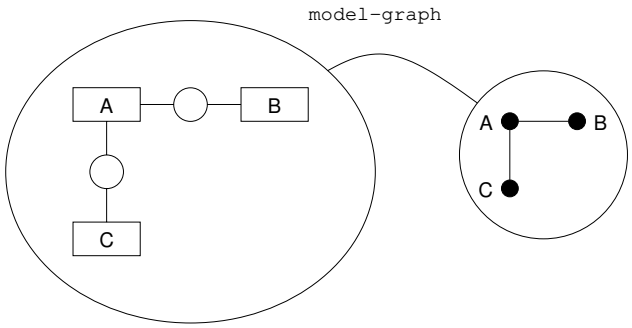


Figure 5.7: An example of the model-graph relation instantiated for a simple model.

The question may rise whether the system–subsystem relation is really different from the graph–subgraph relation. The answer lies in the semantics of being a graph or being a system. Because systems are *made out of* the entities they contain, systems are mereological individuals. This poses restrictions on the system–subsystem relation. For example, a system cannot be made out of a single subsystem. Furthermore, there are rules concerning the overlap of systems (see Section 3.1.1). For subgraphs there are no such restrictions. A graph does not *consist* of subgraphs, a subgraph is just a selection of vertices and edges in a graph.

The type of ontology projection performed in Figure 5.6 is a combination of specialization and mapping. Two *abstract viewpoints* on disassembly models (i.e. systems theory and graph theory) were included and specialized into product models for disassembly. Furthermore, the interdependencies were specified. The difference with domain viewpoints is that abstract viewpoints do not contain different categories of domain concepts, but represent knowledge in different forms. For the definition of some parts of the domain knowledge (`d-model` and connection types), system theoretical concepts are the most convenient starting point, and for other parts (force loops) concepts from graph theory.

5.2.3 Disassembly Operations

Disassembly operations performed on a product require changes in its disassembly model. Based on the different connection types, two kinds of modification operators are distinguished: *loosening* operations that change a connection type from rigid to loose, and *disconnecting* operations that delete loose connections. The first require applying a force to undo the rigidness, while the latter refer to changing the spatial location of objects by moving the objects apart.

Disassembly operations like cutting, sawing and unscrewing change the types of the connections involved from rigid to loose. In the bolt-nut example, loosening the bolt is modeled by replacing the rigid-sideways connection by a loose-sideways connection. Loosening operations can cause force loops to break. As a result, some loose connections will no longer be enforced and certain parts of the product (called subassemblies) can be separated from the rest of the product, by disconnecting operations. Again, the bolt-nut example in Figure 5.2 illustrates this. When the connection between the bolt and the nut has been loosened, the bolt (or the nut) can be removed from the product. This actually disconnects the bolt from the nut and the plate so the connections vanish.

Formalizing these changes in the product we encounter the problem of the extensionality of Mereology. We cannot reason about the modelled product after a disassembly operation because it will have the same parts as the product before and will therefore be considered to be the the same individual. A new concept has to be introduced to be able to capture these product changes. This is the *state* concept: a product after a disassembly operation is in a different state than before. States and events that cause state changes are usually represented as directed graphs in which the vertices represent the states and the edges the events. In these terms, a state space diagram is a directed graph (acyclic in the case of product disassembly).

We can imagine a general state space ontology that defines the concepts of states, events and state space in terms of graph theory. Because the meaning of being in a state depends on the nature of domain entities, the projection of the state space on domain concepts must include this knowledge.

This is what has been done in Figure 5.8. In Line 2 the ontology of product models is included. Line 3 includes the general state space ontology. The fact that a model has a state space is formalized in Line 4. Because a disassembly operation only affects connections, the connection types are a good description of a state. The relation `state.con.type` relates for each state a connection type to every product connection. The connection types that are distinguished are extended with the type *disconnected* to model disconnected connections. Line 5b states that for the initial state in the state space, the connection types correspond to the types specified with `pcon.type`. Axiom 5c is a very important axiom. It defines the *equal* relation for state descriptions. It is used in the state space ontology to ensure that there can be no two states in a state space having the same state description.

What remains to be formalized is the interpretation of the disassembly operations as events that cause state transitions in the state space. This will be presented in the following sections.

```
1  define-theory product-disassembly

2  include-theory disassembly-models

3  include-theory state-space

4  define-relation model-statespace(m,ss)
a    d-model(m) -> exists ss: model-statespace(m,ss) and
                              state-space(ss)

5  define-relation state.con.type(s,c,t)
a    model-statespace(m,ss) and graph.vertex(ss,s) and
       model-graph(m,g) and graph.edge(g,c) ->
         exists t: state.con.type(s,c,t)
b    model-statespace(m,ss) and statespace-istate(ss,s) and
       model-graph(m.g) and graph.edge(g,c) and pcon.type(c,t) ->
         state.con.type(s,c,t)
c    exists s1,s2: forall c: exists t:
       (state.con.type(s1,c,t) <-> state.con.type(s2,c,t)) ->
         equal-states(s1,s2)
```

Figure 5.8: Proposal for an ontology for product disassembly (i). The ontology includes the ontologies of disassembly models and state space and defines what it means for a product to be in a state of the state space.

5.2.4 Subassemblies

Loosening of rigid connections can make it possible that groups of objects can be removed from the product. If this is the case, there may be no enforced connections between such a group and the rest of the product. Furthermore it is required that the group does not contain smaller groups of objects that can be separated. In other words, internal connections in a group have to be enforced. In the ontology for disassembly, groups of components having these properties are called *subassemblies*.

Figure 5.9 gives an illustration of the concept of subassemblies. The model on the left-hand side consists of one subassembly because the enforced force loop holds all objects together. This changes when the sideways connection is loosened, as can be seen in the model on the right-hand side. Objects B and C still form a subassembly because there is a rigid connection between them. All other objects form individual subassemblies because they are loosely connected to other subassemblies.

For a formal definition of subassemblies, again it is convenient to switch to a graph theoretical view on a product model. When we consider the objects as vertices and *enforced* connections as the edges of a graph, the subassemblies of a product correspond to the so called connectivity components (the *maximal connected subgraphs*) of this graph. During product disassembly, the subassemblies change. Therefore they have to be defined for each product state. Figure 5.10 illustrates these definitions. For each product state an *assembly graph* is defined in which the maximal connected subgraphs specify the product subassemblies.

The ontology that is the result of this scheme can be found in Figure 5.11. The relation `state-asygraph` (Definition 6) relates every product state to a graph of the type described

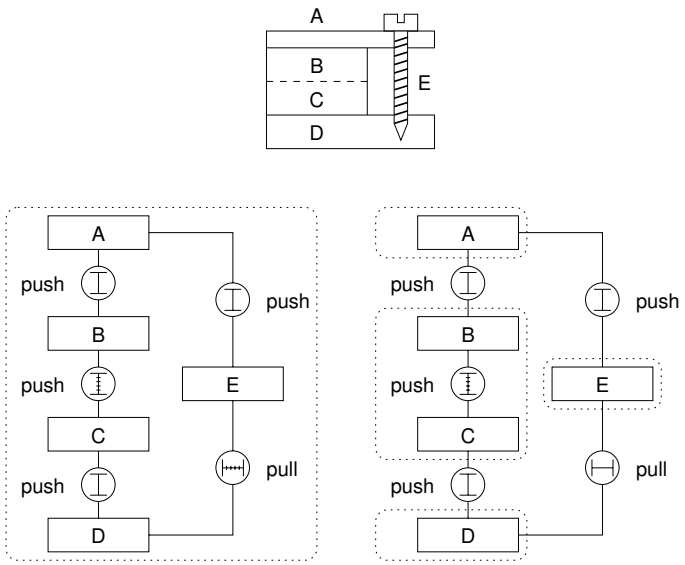


Figure 5.9: Subassemblies.

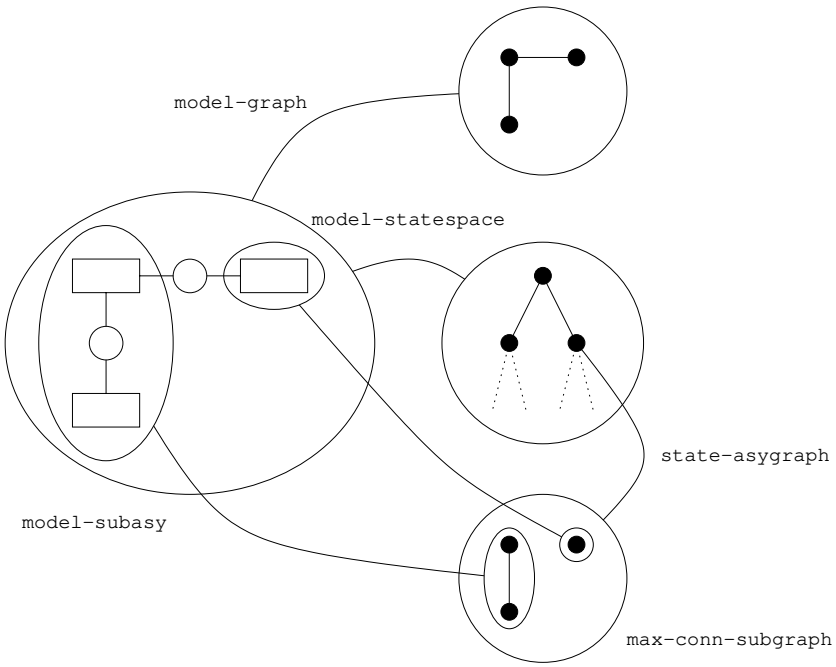


Figure 5.10: The relation between disassembly models, model state space, (sub)assembly graphs and model subassemblies.

```
6  define-relation state-asygraph(s,ag)
a    d-model(m) and model-statespace(m,ss) and
     graph.vertex(ss,s) and model-graph(m,g) ->
       exists ag: state-asygraph(s,ag) and
                  (graph.vertex(g,o) <-> graph.vertex(ag,o)) and
                  (graph.edge(g,c) and enforced(c) <-> graph.edge(ag,c))

7  define-relation state-subasy(s,sa)
a    d-model(m) and model-statespace(m,ss) and
     graph.vertex(ss,s) and state-asygraph(s,ag) and
     max-connected-subgraph(ag,sag) ->
       exists sa: state-subasy(s,sa) and subsystem-of(sa,m) and
                  (graph.vertex(sag,o) <-> in-system(o,sa))
```

Figure 5.11: Proposal for an ontology for product disassembly (ii). The ontology defines product subassemblies in terms of maximal connected subgraphs which are defined in the ontology of graph theory.

above. The subassemblies of a product in a certain state can then be defined (Definition 7) using concepts from the ontology of graph theory.

In the literature methods can be found to determine graph properties and paths and subgraphs having a certain property (Skvarcius and Robinson 1986). Examples are the computation of the number of maximal connected subgraphs in a graph, finding a (sub)optimal path covering all vertices of a graph (traveling salesman problem), finding a minimal spanning tree, finding a shortest path in a weighted digraph (Dijkstra's Algorithm) etc. A method to determine the maximal connected subgraphs (MCS) of a graph G is presented in Figure 5.12. In Section 5.4 it is used in the design of a prototype KBS that performs disassembly analysis.

```
MCS = {}
while there is a vertex v in graph G
      that is not in a graph in MCS
  create new graph S({v},{})
  while there is a vertex v' in G that is
        connected to a vertex in S
    add all edges between v' and vertices
        in S to the edges of S
    add v' to the vertices of S
  end
  add S to MCS
end
```

Figure 5.12: A method to compute the set of maximal connected subgraphs `MCS` of a graph `G`.

Each subassembly is a candidate for removal, but a direction has to be found in which the subassembly can be moved away from the rest of the product. The geometric information captured in the force loops can be reused to find the desired direction. Three cases can be distinguished, as follows.

1. When a subassembly has a loose external connection that is *not* linked to another external connection of the subassembly by a force loop, it means that no object blocks the subassembly in the direction of the connection. Because any other external connection

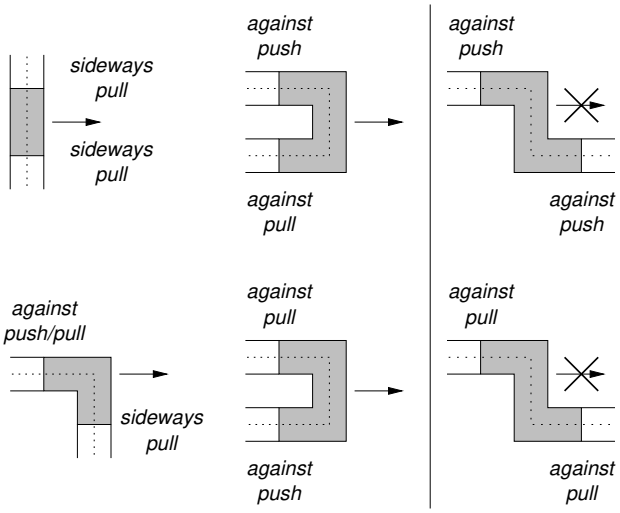


Figure 5.13: Situations where a subassembly (represented by the gray shapes) can be removed (left of the vertical line) or is blocked (right of the line).

of the subassembly has a different direction and is not enforced, the subassembly can be removed.

2. When the removal of the subassembly in the direction of a connection is blocked by external objects, a force loop goes through the connection and a second external connection of the subassembly. The subassembly is only blocked when this second connection is on the *opposite side* of the subassembly. The geometric information in the force loop can be used to see whether this is the case. This can be seen in Figure 5.13. Situations where a force loop leads to a connection on the opposite side so the subassembly is blocked appear on the right-hand side of the vertical line.

3. It may also be the case that there are *more than one loop* going through the external connection that lead to other external connections of the subassembly. Each external object connected by these other external connections may then be blocking the subassembly. Therefore, the subassembly can only be removed in the direction of the first external connection when *all* force loops through the connection match a situation depicted on the left hand side of the vertical line in Figure 5.13.

A subassembly is called a *free* subassembly when it can be removed. To see whether a subassembly is free, an external connection has to be found that matches one of the three cases described above. Figure 5.14 gives the formalization.

Note that this definition only takes into account the blocking of subassemblies caused by direct physical contact with other subassemblies. For situations where objects that are not in direct contact prevent subassemblies to be removed (or connections to be loosened), such

```
8  define-relation free(sa)
     free(sa) <->
       exists m,ss,s:
         model-statespace(m,ss) and graph.vertex(ss,s) and
         state-subassembly(s,sa) and
         exists c1:
           in-boundary(c1,sa) and
           not(state.con.type(s,c1,disconnected)) and
           forall fl,c2:
             path.edge(fl,c1) and path.edge(fl,c2) and
             in-boundary(c2,sa) and c1 != c2 ->
               state.con.type(s,c2,disconnected) or
               state.con.type(s,c1,s-loose) or
               state.con.type(s,c2,s-loose) or
               (state.con.type(s,c1,a-loose) and
                state.con.type(s,c2,a-loose) and
                not exists f: loop.con.force(fl,c1,f) and
                              loop.con.force(fl,c2,f))
```

Figure 5.14: Proposal for an ontology for product disassembly (iii). The relation `free` specifies which subassemblies can be removed from the rest of the product.

as closed covers or lids, the *disassembly state* concept (not to be confused with a state in state space) has been introduced. This makes it possible to specify for instance that the lid has to be opened or that the cover has to be removed before subassemblies can be removed from the product.

5.2.5 Disassembly Analysis

Given a product model at any point in the disassembly process, physical disassembly operations effectuate one of two kinds of changes in the model:

- connections can be loosened, or
- free subassemblies can be removed, disconnecting the external connections.

For a realistic environmental analysis of product disassembly, the edges of the state space should correspond to physical disassembly operations like sawing, unscrewing, breaking etc. However, for simplicity we will abstract from physical operations and relate loosening and removing operations to the edges in state space. This has been formalized by the relation `edge-op` in Figure 5.15.

The ternary relation `edge-op` relates the operation type (`t:loosen/remove`) and the object being handled (`a`: a connection or a subassembly) to an edge. Axiom 9a states that every edge in a model's state space must have a disassembly operation associated to it. The dependency between the type of disassembly operation and the type of the object being handled is formalized by Axiom 9b. Axiom 9c models the fact that an edge in the state space graph can be related to only one disassembly operation. Axiom 9d and 9e ensure that for each product state in which a connection can be loosened or a subassembly can be removed, an edge exists

```
9  define-relation edge-op(e,t,a)
a    model-statespace(m,ss) and graph.edge(ss,e) ->
       exists t,a: edge-op(e,t,a)
b    edge-op(e,t,a) ->
       (t = loosen <-> d-connection(a)) and
       (t = remove <-> exists s: state.subasy(s,a))
c    edge-op(e,t1,a1) and edge-op(e,t2,a2) ->
       t1 = t2 and a1 = a2
d    model-statespace(m,ss) and graph.vertex(ss,s1) and
     (state.con.type(s1,c,a-rigid) or
      state.con.type(s1,c,s-rigid)) ->
       exists e,s2:
         edge-op(e,loosen,c) and graph.edge(ss,e) and
         graph.edge(ss,s2) and edge.from-to(e,s1,s2)
e    model-statespace(m,ss) and graph.vertex(ss,s1) and
     state-subasy(s1,sa) and free(sa) ->
       exists e,s2:
         edge-op(e,remove,sa) and graph.edge(ss,e) and
         graph.edge(ss,s2) and edge.from-to(e,s1,s2)
f    model-statespace(m,ss) and graph.edge(ss,e) and
     edge.from-to(e,s1,s2) and edge-op(e,loosen,c) ->
       forall ci:
         (ci = c and state.con.type(s1,ci,a-rigid) ->
                     state.con.type(s2,ci,a-loose)) and
         (ci = c and state.con.type(s1,ci,s-rigid) ->
                     state.con.type(s2,ci,s-loose)) and
         (ci != c and state.con.type(s1,ci,t) ->
                      state.con.type(s2,ci,t))
g    model-statespace(m,ss) and graph.edge(ss,e) and
     edge.from-to(e,s1,s2) and edge-op(e,remove,sa) ->
       forall ci:
         (in-boundary(ci,sa) ->
            state.con.type(s2,ci,disconnected)) and
         (not in-boundary(ci,sa) ->
           (state.con.type(s1,ci,t) <-> state.con.type(s2,c,t)))
```

Figure 5.15: Proposal for an ontology for product disassembly (iv). The relation `edge-op` relates edges in the state space to operations that change disassembly models.

in the state space. Axioms 9f and 9g define the relationship between edges in state space and the transformation of the state description of the states that are linked. Note that the result of the projection of the state space onto the process models for disassembly is similar to situation calculus (Davis 1990).

The benefit of the definition of the state space is that there are various general search algorithms defined in terms of state space graphs (Bolc and Cytowski 1992). Some of these have the precondition that the entire state space graph must be in memory beforehand, others can *build* the state space graph themselves using functions that generate successor states.

The algorithms can also be different in the stop criterium they use. For some tasks it is sufficient to find a state that matches a certain criterium. As soon as one such state is found, the search can be stopped. The most straightforward approach to find an optimal state in the state space is to examine every state in it. Smarter methods use knowledge to cancel the examination of states in a branch of the tree when it can be deduced that every state in this branch is suboptimal.

In heuristic approaches, the search is guided by an estimation of the effectiveness of the edges leading out of the state being examined. Edges with a high estimated effectiveness can be search first, edges with a lower estimation later and edges with a very low estimation can even be disregarded.

One can also differentiate between finding the best state disregarding the cost to reach that state, or finding the best state with minimal cost. In the first case, a state-benefit function has to be specified whereas in the second case, not only the benefit of a state is important, but also the cost of reaching that state from the initial state.

These methods can be used to accomplish certain tasks (such as finding a good disassembly sequence). Each method has a certain *competence* in accomplishing the task and requirements on the knowledge it uses. These aspects can be formally specified in so called method ontologies. For a more detailed explanation of method ontologies, see (Gennari, Tu, Rothenfluh, and Musen 1994; Tu, Erikson, Genari, Shahar, and Musen 1995; Fensel, Schönegge, Groenboom, and Wielinga 1996).

For disassembly analysis, a search method can used that searches the entire state space (all disassembly sequences) to find the optimal state with the lowest cost. Figure 5.16 shows such a method in pseudo-code. We can see that tree types of domain knowledge are required by the method. This knowledge is expressed in terms of concepts from the state space ontology.

- the initial state
- a method to determine the edges out of a given state
- a method to compare paths starting from the initial state to any other state in the state space

The ontology of product disassembly we have constructed so far contains the first two types of required domain knowledge. We only have to add a relation to compare disassembly sequences, i.e. acyclic paths through the state space.

```
statespace = {initial-state}
best-path = {}
find-optimal-path(initial-state,{})

find-optimal-path(s,p)
  if better-path(p,best-path)
    best-path = p
  endif
  foreach state s' that can be reached from s following edge e
    add state s' to statespace
    add edge e from s to s' to statespace
    add e to p
    find-optimal-path(s',p)
    remove e from p
  end
end
```

Figure 5.16: Exhaustive, depth-first state space search method to find an optimal path.

Our formal definition of the relation `better-path` (which defines a total order over state space paths) is very simple and unfortunately not realistic for practical disassembly analysis. Nevertheless we have chosen it because it allows to build a prototype to assess the validity and usability of the theory presented in this chapter. Figure 5.17 gives a (partial) specification of the relation.

```
10 include-theory engineering-mathematics

11 define-relation model-clump(m,c)
a    model-clump(m,c) ->
       d-model(m) and subsystem-of(c,m)

12 define-relation better-path(p1,p2)
a    better-path(p1,p2) <-> rating(p1) < rating(p2)
```

Figure 5.17: Proposal for an ontology for product disassembly (v). The relation `better-path` specifies a total order over paths in the state space. It is required by the method in Figure 5.16 to find the best disassembly sequence.

The comparison of two disassembly sequences is based on a specification of objects that have to be removed for recycling or reuse. Such a group of objects is called a *clump* and a relation that associates a clump to a model can be found in Definition 11a. Each loosening or removal operation applied to the model accounts for some given ecological cost (e.g. 5 units for loosening a connection, 1 unit for removing a subassembly). The ecological benefits depend on the degree to which the clump objects have been separated from the product. For simplicity, the cost-benefit analysis does not calculate an ecological benefit, but instead a penalty (e.g. 15 units) for each unwanted (non-clump) object that is still attached the clump. The total rating of a situation in the disassembly process is then defined as $1/(cost + penalty)$. Figure 5.17 does not show the function that defines this rating. We suffice to say that it is possible using the EngMath ontology of engineering mathematics and previously introduced concepts and relations. What the figure does show is the way the rating function is used to define the `better-path` relation required by the search method.

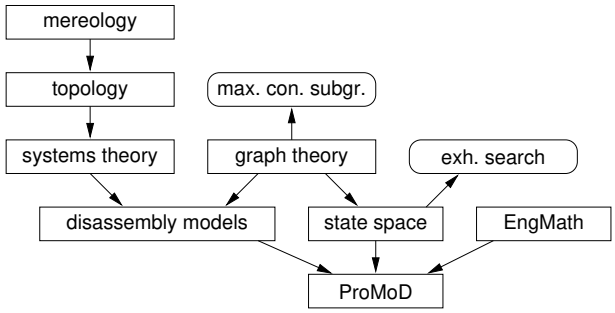


Figure 5.18: Inclusion lattice of the application ontology of PROMOD. Boxes represent ontologies and arrows indicate ontology inclusion. Rounded boxes are method ontologies.

Together with the definitions in Figure 5.8, 5.11,5.14 and 5.15, Figure 5.17 gives a proposal for an application ontology for product disassembly analysis. Figure 5.18 shows its inclusion lattice. Boxes with rounded corners are method ontologies defining the competence and knowledge requirements for the problem solving methods used.

5.2.6 Model Extensions

Based on the application ontology defined in the previous section it is possible to implement a prototype KBS that performs disassembly analysis on a product model. Before we will describe this prototype in Section 5.3, some extensions will be introduced that allow complex product structures to be modelled. We will not give ontological definitions of these extensions, so readers who are mainly interested in ontology construction can proceed with Section 5.3 from this point.

Three new model elements will be described in this section. First, the *disassembly state* concept will be introduced. With disassembly states, preconditions for loosening connections can be defined. After that, concepts to handle dependencies between connections will be described. An example of dependent connections are rigid sideways connections based on friction that are formed by clamping an object between two other objects. Finally, we will explain how situations where connections in different directions form a `virtual' connection can be modelled.

Conditional Disassembly

Sometimes, breakable connections can be loosened and disconnected only when some parts of the product have been removed. For many products the case has to be opened before connections inside the case can be loosened. Therefore, connections that can be loosened only under a certain condition have been introduced. This condition can best be looked upon as a *state* the (partially disassembled) product has to be in for further disassembly. Note that

this notion of product state is different from the states introduced in Section 5.2.5. We will therefore call the states that are conditions for the loosening and disconnection of connections *disassembly states*.

A disassembly state can be defined as a logical expression of other disassembly states (with the logical operators *and*, *or* and *not*) and the predicates *disconnected* and *separated*. The predicate *disconnected* is true for all connections that are disconnected in a situation. The predicate *separated* is true for tuples of objects of which one object has been removed from the other, i.e. when they are part of separated subassemblies.

In situation where, for instance, a front panel has to be removed, *separated(front,case)* could be the condition for breakable connections inside the case. In the situation where there is a lid connected to the case by a hinge and a fastening, there is no need that the lid is completely separated from the case, so the condition can be weaker. Here, the condition *disconnected(fastening)* is sufficient as the hinge may still be intact.

Dependent Connections

Up to now, there has not been any consideration about the fact that a sideways connection in one direction can form an against connection in a perpendicular direction. This has not been necessary because usually connections are independent. However, there are two common situations where there is such a dependency: against connections that form sideways connections based on friction and sideways connections that form against connections. Figure 5.19 gives two examples.

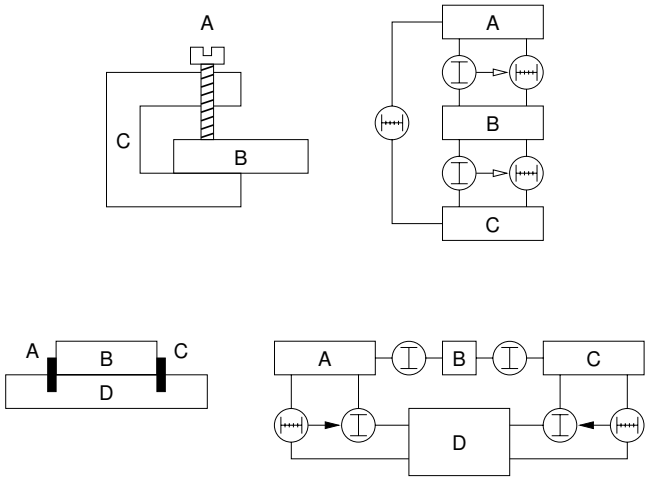


Figure 5.19: Two different dependencies between connections.

In this figure, the top model shows a construction where an object B is connected in the horizontal direction as long it is pressed by object A in the vertical direction. The open arrows

therefore have to be interpreted as: the connection pointed to is rigid as long as the connection that is pointed from is enforced by a force loop. The two rigid sideways connections become loose sideways connections when the screw is unscrewed. Object B then becomes a free subassembly and can be removed.

The bottom part of the figure shows a construction where an object B is clamped between two pins (objects A and C) that are tight-fitted in object D. When the rigid sideways connection between a pin and object D in the vertical direction is disconnected, it means that the pin is pulled out of object D. As a result, the loose against connections in the horizontal direction are also disconnected. This is modelled with the solid arrows: the connection the arrow points to is disconnected when the connection that it points from is disconnected.

Dependent Force Loops

In the models encountered so far, all directions of the connections were independent or perpendicular to one another. Now, situations are considered where there is a dependency between the direction of connections. An example of such a situation can be found in Figure 5.20 where a disc (A) is fixed by three objects (B, C and D) that are attached to a plate (object E). In this situation, it is easy to find the loops that exist by imagining what forces must apply to the connections to cause the fact that loose against connections are pushed. The only difference with previous situations is that here, in order to be enforced, *all* loops going through the loose against connections have to be intact instead of just one.

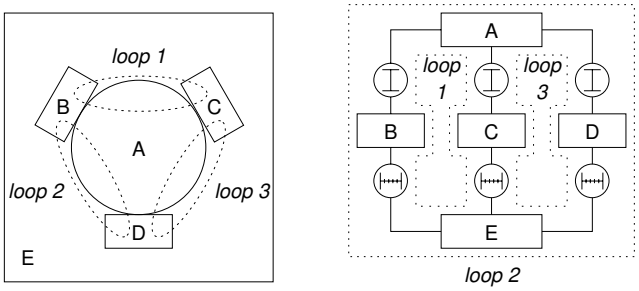


Figure 5.20: Two loops collectively enforce a loose connection.

A model extension that allows these situations to be handled is the introduction of so called *groups of dependent force loops*, or just *groups*. In the example of Figure 5.20, the three loops form one group and only if all loops in the group are intact, the loose connections in the loops are enforced. The introduction of groups requires some changes in the theory that has been presented.

The amount of changes to the theory can be kept to a minimum when it is required that all loops belong to at least one group. This means that in all previous examples, each loop forms a group that contains only that loop. This also means that in the new situation a connection is enforced iff it is a rigid connection or a loose against connection that has a loop going through it that belongs to an intact *group*. A group is intact when all of its loop are intact.

5.2.7 Related Work in Mechanical Engineering

The problem to obtain the AND/OR graph of a product has also been considered in computational (mechanical) engineering, and several approaches are reported in the literature.

In the *reversed fishbone* approach (Ishii and Lee 1996), the designer has to come up with the preferred way to disassemble the product him/herself. This is graphically specified in a reversed fishbone diagram which can be considered as a sub-graph of the AND/OR graph. The disadvantage of this approach is that when the design of the product changes, possibly large parts of the diagram have to be changed. This makes this method less suited for comparison of alternative designs.

A method to obtain the AND/OR graph automatically is to generate it from a geometric model of the product. This is done in the *degrees of freedom* (Dof) approach (Khosla and Mattikali 1989) where a 3D CAD drawing of the product is utilized. After a design modification the AND/OR graph can automatically be recomputed, but the drawback is that it needs extensive input information, which is not always available in the early stages of product design.

The third approach uses a topological *liaison diagram* (Fazio and Whitney 1987; Sturges Jr and Kilani 1992) in which nodes denote objects and edges physical connections. The AND/OR graph is generated from this diagram plus additional relations specifying a partial ordering over the breaking of connections. A tool based on this approach is LASER (Ishii, Eubanks, and Marks 1993; Ishii, Lee, and Eubanks 1995).

Our theory is related to the last approach. Its novel and distinguishing aspect is that by a clever choice of connection types, it can capture geometric information otherwise only available in 3D geometric models. The use of these new models is what makes the PROMOD system that will be described in the next section different from a tool like LASER.

5.3 The PROMOD System for Disassembly Analysis

PROMOD is a prototype KBS that implements the product models for disassembly as well as the AND/OR tree generation algorithm described in Section 5.2. In addition, it contains a simple form of ecological cost-benefit analysis. This section describes the functionality of PROMOD. Most importantly, the syntax of the input file that specifies a disassembly model will be defined. An example session will be given to demonstrate PROMOD's usage.

5.3.1 Functionality of PROMOD

PROMOD first reads in a product model and then generates the AND/OR tree. The ecological cost-benefit analysis is then performed on each situation in the disassembly process defined by the AND/OR tree. It uses a list of objects (called the *clump*) that have to be removed from the product for recycling or reuse. Each loosening or removal operation applied to the model

accounts for some given ecological cost (e.g. 5 units for loosening a connection, 1 unit for removing a subassembly). The ecological benefits depend on the degree to which the clump objects have been separated from the product. For simplicity, the cost-benefit analysis does not calculate an ecological benefit, but instead a penalty (e.g. 15 units) for each unwanted object that is still attached to one of the clump objects. The total evaluation score of a situation in the analysis process is then defined as $1/(cost + penalty)$.

5.3.2 Syntax of PROMOD Models

We will describe the syntax of the PROMOD model files using the BNF technique (Naur 1963) for syntax definitions. A short overview of the notation can be found in Figure 5.21. The definition of the syntax for PROMOD models can be found in Figure 5.22. The top-level production rule is *model*.

model : rule	production rule for non-terminal *model*
`product`	the terminal `product` appears literally
{ rule }	group the definitions between curly brackets
[rule]	the rule appears optionally
`push` | `pull`	either `push` or `pull` appears
object...	one or more appearances of *object*
{ rule }...	one or more appearances of the rule
[rule]...	zero or more appearances of the rule
↩	a newline character
EOF	the end-of-file character
id	an identifier
string	a character string

Figure 5.21: Notational conventions of BNF syntax definitions.

A product line specifies the name of the modelled product. The name is a character string that follows the `product` keyword and is ended by the end of the input line.

An object definition is an identifier defining its name followed by identifiers specifying additional information about the object. An identifier is a string of characters without whitespace that is a name of a model element. All model elements must have a unique name. For the moment, PROMOD disregards the extra information of objects.

Because the prototype has no knowledge of physical disassembly operations like unscrewing, cutting, sawing etc. it cannot determine which rigid connections in the product model can be loosened. Therefore, the fact that a (rigid) connection can be loosened and ultimately be disconnected has to be specified explicitly with the optional `breakable` keyword in the connection definition. The disassembly algorithm will only loosen rigid connections that were marked as breakable. When a name of a disassembly state is specified in a connection definition it means that the connection can only be loosened when the product has certain properties (see below).

model : `product` *name* ↩
{`object` *object whatever* `end`}...
[`connection` *connection type object object* [`breakable` [*state*]] `end`]...
[`loop` *loop node node*... `end`]...
[`group` *group loop*... `end`]...
[`state` *state* {*separate* | *disconnected* | *and* | *or* | *not*} `end`]...
`clump` *object*... `end`
EOF

name : *string*
object : *id*
whatever : [*id*]...
connection : *id*
type : `a-loose` | `a-rigid` | `s-loose` | `s-rigid`
loop : *id*
node : *connection force*
force : `push` | `pull`
group : *id*
state : *id*
separate : `separate` *object object*
disconnected : `disconnected` *connection*
and : `and` *state state*...
or : `or` *state state*...
not : `not` *state*

Figure 5.22: BNF syntax definition of PROMOD models.

A force loop is formed by a collection of nodes. A node is a combination of a connection and the force that is applied to the connection. In order to make it easy to check whether a loop is properly defined, the connections in the nodes of a loop must be specified in the order in which they are visited when the path is followed from object to object.

A disassembly state definition specifies states that parts of a model can have during the disassembly process. There are five different kinds of states: *separate*, *disconnected*, *and*, *or* and *not* states. A *separate* state holds iff the objects are part of subassemblies that have been separated. A *disconnected* state is true iff the specified connection has been disconnected. The *and*, *or* and *not* states are true when the logical operators they represent are true when applied to the specified states.

5.3.3 An Example Session

We will now discuss an example to give an indication of the usability and reasoning power of the PROMOD system. Figure 5.23 shows a model of a coffee machine that has been analyzed by the system. Connection `c8` can only be loosened when the base has been separated from the case. The object `block` has been marked as the clump.

Figure 5.24 shows part of the results of the disassembly analysis by PROMOD. For each sequence of disassembly operations information is given on the subassemblies, the state of the subassemblies, the groups of objects that are separated and on the ecological rating of the

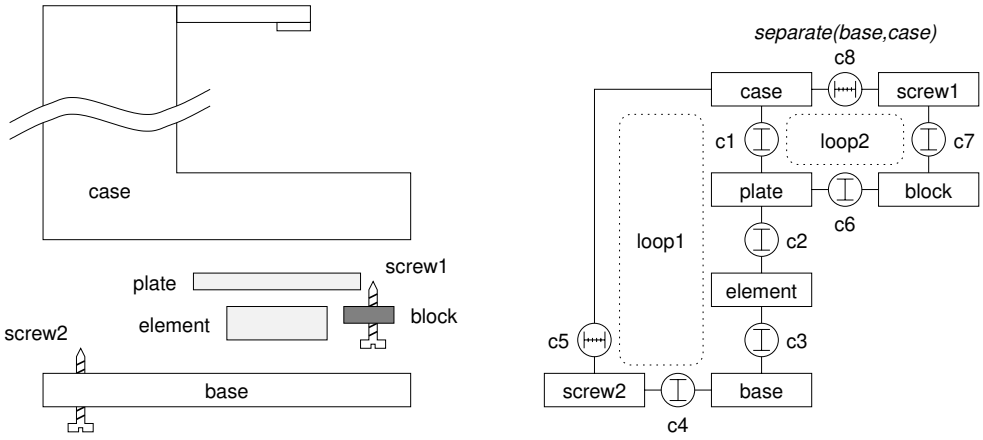


Figure 5.23: A disassembly model of a coffee machine.

situation. The rating consists of three numbers: the ecological cost of disassembly, the total object penalty and the sum of these numbers. The ratings can also be visualized in a diagram like in Figure 5.25.

5.4 Implementation of PROMOD

The PROMOD prototype has been implemented in the programming language C (Kernighan and Ritchie 1988). To cater for easy prototyping, Lisp-like data structures and functions that operate on lists have been used (Steele Jr. 1990). This section describes the most important aspects of the implementation and the way the application ontology presented in Section 5.2 could be used as a specification of the prototype.

In Section 5.4.1, the data structures in PROMOD will be introduced using the OMT modelling technique (Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen 1991). The actual implementation in terms of lists will be specified with BNF definitions (Naur 1963). Most of the concepts, attributes and methods are implementations of what has been described in Section 5.2. However, some new concepts were introduced to overcome combinatorial explosion that lead to a poor performance of the prototype. Other extensions concern the extended model entities for product states and force loop groups.

Section 5.4.2 describes the most important functions and procedures in PROMOD. A mixture of pseudo-code and real function and procedure calls is used in these descriptions.

```
Situations considered for `Coffee Machine'

sequence: ((loosen c5) (remove screw2) (remove base)
           (loosen c8) (remove screw1) (remove block))
asies:    ((free case) (blocked plate) (separate screw1)
           (free element) (separate base) (separate block)
           (separate screw2))
clumps:   ((case plate element) (screw1) (base) (block)
           (screw2))
rating:      14    0   14
sequence: ((loosen c5) (remove screw2) (remove base)
           (loosen c8) (remove screw1))
asies:    ((free case) (blocked plate) (separate screw1)
           (free element) (separate base) (free block)
           (separate screw2))
clumps:   ((case plate element block) (screw1) (base)
           (screw2))
rating:      13   75   88

[many sequences removed]

sequence: ((loosen c5))
asies:    ((free case plate screw1 block) (blocked element)
           (blocked base) (free screw2))
clumps:   ((case plate screw1 element base block screw2))
rating:       5  150  155
sequence: NIL
asies:    ((separate case plate screw1 element base block
            screw2))
clumps:   ((case plate screw1 element base block screw2))
rating:       0  150  150
```

Figure 5.24: Disassembly analysis result from the PROMOD system for the coffee machine.

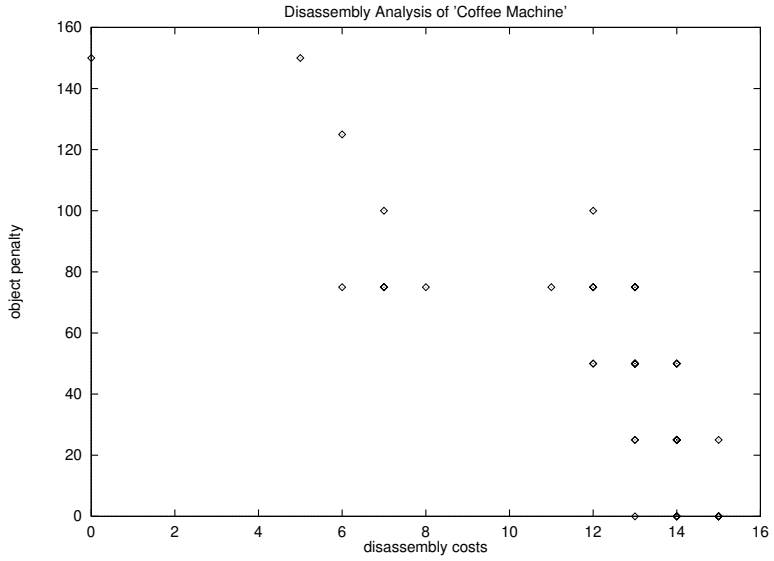


Figure 5.25: Cost-benefit analysis of the coffee machine. Diamonds represent disassembly sequences from Figure 5.24 having the same disassembly costs and object penalty

5.4.1 Data Structures

An OMT conceptual scheme of the data structures of PROMOD can be found in Figure 5.26. A short overview of the OMT notation is given in Chapter 2 in Figure 2.4.

The data structures have been designed based on the definitions in the application ontology in Section 5.2. For efficiency reasons, the two abstract viewpoints on the product models (system theory and graph theory) have been combined to form compact data structures. Constraints on the models, as expressed by the axioms in the ontologies were used in the design of the syntax of the input files and data consistency checks performed by the input parser. The functions and procedures that modify the data structures have been designed in such a way that model consistency is retained throughout the analysis process.

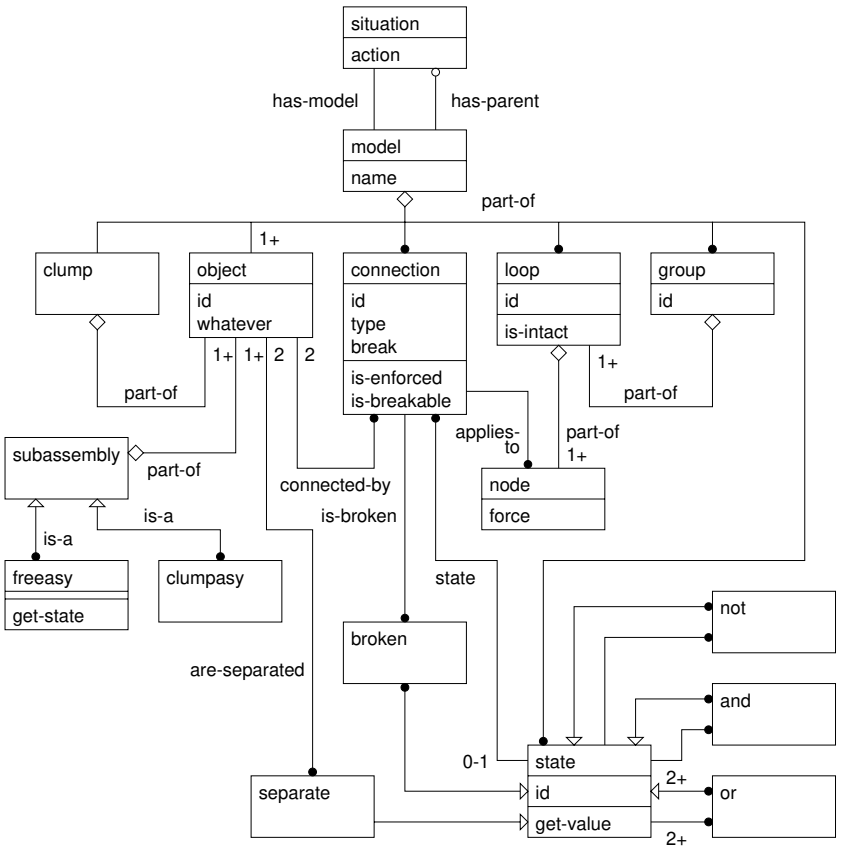


Figure 5.26: Conceptual schema of the data structures in PROMOD.

In the following sections, the data structures will be described as well as the way they have been implemented using lists.

Models

In Figure 5.27 the BNF definitions of the Lisp data structures that implement the conceptual scheme in Figure 5.26 are given.

model : (*connections loops objects groups states clump name*)
objects : ((*object whatever*) . . .)
connections : ((*connection type object object break* [*state*]) . . .)
loops : ([(*loop intact node*. . .)]. . .)
groups : ([(*group loop*. . .)]. . .)
states : ([(*state* {*separate* | *disconnected* | *and* | *or* | *not*})]. . .)
clump : (*object*. . .)

name : *string*
object : *id*
whatever : [*id* | *string* | *integer* | *float*]. . .
connection : *id*
type : `a-loose` | `a-rigid` | `s-loose` | `s-rigid`
break : `breakable` | `unbreakable` | `disconnected`
loop : *id*
intact : `intact` | `broken`
node : (*connection force*)
force : `push` | `pull`
group : *id*
state : *id*
separate : `separate` *object object*
disconnected : `disconnected` *connection*
and : `and` *state state*. . .
or : `or` *state state*. . .
not : `not` *state*

Figure 5.27: BNF definition of PROMOD's model data structures.

The disassembly model, the objects, connections, loops, groups and loosening states all have unique names as identifiers. The parser will generate an error message when model elements with non-unique names are defined or a definition refers to an undefined model element.

The model definition consists of a list containing the definition of the objects, connections, loops, groups, disassembly states and the clump of the modelled product. To avoid confusion between product states in the state space and disassembly states defined for conditional loosening, the product states in state space will be referred to as *situations* from now on and disassembly states as *states*.

The *break* attribute of a connection is initialized with `breakable` or `unbreakable` depending on the occurrence of the `breakable` keyword in the definition of the connection in the input file. During the disassembly analysis it can be changed to `disconnected` when a subassembly is removed from the product.

The attribute `intact` of a loop indicates whether a loop is intact or broken. It is not a real attribute because its value can always be determined by inspecting the connections that form the loop. In PROMOD it is used to speed up the disassembly analysis. Its initial value is determined as soon as the model has been read in. During disassembly analysis, its value is

updated every time a connection in the loop is loosened or disconnected. Together with break attributes and the *type* attribute of a connection, that can be changed from rigid to loose, *intact* attributes are the only thing in which the models in two situations can be different.

The methods *is-enforced* (of connections) and *is-intact* (of loops) are the implementations of the definitions of the *enforced* and *intact* predicates in Section 5.2.2.

The optional *state* relation specifies the disassembly state in which a product must be before a connection can be loosened. The method *get-value* determines the truth value of a state.

The method *is-breakable* returns true for connections that are in a state in which they can be loosened. This includes checking whether the connection has *breakable* as break attribute and whether the optional state related to the connection is true. Furthermore, the method takes into account the fact that when an enforced against connection is loosened, it is still not possible to move the connected objects away. The method *is-breakable* therefore only returns *true* when the connection is not enforced. This will reduce the number of equivalent disassembly sequences generated.

Subassemblies

A subassembly is a list of the identifiers of objects in a group of objects. Therefore they are more like subgraphs then subassemblies. However, the extra constraints the ontology of systems theory puts on subassemblies are respected due to the way the subassemblies are determined. This is done by a procedure that is an implementation of the maximum connected subgraph method.

subassembly : (*object*...)

The object-classes *freeasy* and *clumpasy* are subassemblies with restrictions on the internal and external connections of the objects in it.

A *freeasy* is a subassembly of the type defined in Section 5.2.2: internally connected by enforced connections and externally by loose connections that are not enforced. A freeasy is therefore a rigid group of objects that is connected loosely to other subassemblies in the product.

A *clumpasy* is somewhat different: the internal connections may be of any kind as long as they are not disconnected. The external connections however *must* be disconnected. The objects in a clumpasy therefore are spatially (rigidly or loosely) connected, but as a group they are disconnected from the rest of the product.

The reason for the introduction of clumpasies is that they can be used to determine the truth values of the *separate* states. Two objects are separated iff they are part of different clumpasies. Furthermore, clumpasies are used to determine the object penalty. If a clumpasy contains a clump object, every non-clump object in the clumpasy contributes to the object penalty.

Freeasies can have two states. They can be *free* according to the definition in Section 5.2.4, so they can be removed, or they are *separate*, which means that they have been removed already. When a freeasy also happens to be a clumpasy, its state is defined as *separate*. In all other cases it is *blocked*. The method *get-state* determines the state of a freeasy.

Situations

The product disassembly state space is represented by a list of situations. Each situation is a list containing a model definition, a pointer to the predecessor (parent) situation and a specification of the disassembly operation (action) that formed the situation from its parent.

situation : (*model parent action*)
parent : *model*
action : (`loosen` *connection*) | (`remove` *subassembly*)
situations : (*situation*...)

Because the models in the situation list only differ in the definition of the connections and force loops, these definitions are placed at the head of a model definition list. This speeds up the check whether a disassembly operation leads to a situation that is already in the situation list. Also, because the other definitions in the model of a situation are identical to those of the initial model, they are implemented as pointers to the definitions in the initial model, rather than copies.

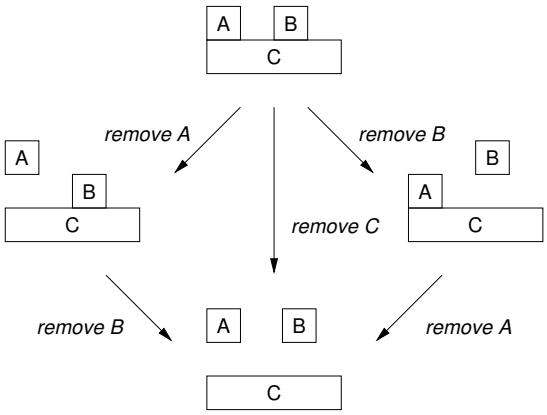


Figure 5.28: Three paths lead to the fully disassembled situation.

The implication of the data structures chosen here is that it is impossible to store the information that a situation can be reached from two alternative parent situations. A model in which this occurs is shown in Figure 5.28 where two objects A and B are loosely connected to a third object C. There are three paths to reach the fully disassembled situation, but a situation can have only one parent situation, so it is impossible to represent the whole figure in the data

structures. Fortunately this is not a problem because when a second path to the final situation is found, the cost of path can be calculated and only the best path can be stored. When a better path to the final situation is found, the action and parent situation of the final situation gets updated.

5.4.2 Functions and Procedures

This section describes the main functions and procedures in PROMOD. We will describe the main procedure, the implementation of the disassembly algorithm based on the state space search method, the determination of the subassemblies based on the method to determine the maximal connected subgraphs, the determination of the freeasy states based on the ontological definition in Section 5.2 and the determination of the truth value of product states. For each function and procedure a detailed pseudo code specification will be given that is explained further in the text.

Main

The main procedure (Figure 5.29) is straightforward. Disassembly analysis is started by introducing the initial situation which consists of the model that is read from file, an empty parent situation specification and an empty disassembly operation (action). The determination of the best disassembly sequence is delayed until all situations have been considered. The reason for this will become clear when the disassembly procedures are described.

```
procedure main()
  initialize cost/penalty constants
  open input and output files
  m = fread_model(fin)         (read model)
  write model to .out file
  new_situation(m, NIL, NIL)   (start disassembly analysis)
  dump_situations(...)         (determine best sequence and write all
                                results to output files)
  dump_gnuplot(...)            (write gnuplot file)
  close files
  remove data structures from memory
end
```

Figure 5.29: The main procedure of PROMOD.

Disassembly Algorithm

The procedures that perform the disassembly analysis have been adapted from the exhaustive, depth first state space search algorithm to find an optimal path in Figure 5.16. The method has been implemented with two procedures which build the search state space dynamically by generation of successor situations which are added to the situation list.

It is possible that a situation can be reached by application of alternative disassembly operation sequences, of which only one can be stored in the data structures. Therefore, it must be checked whether a generated situation already exists in the situation list.

The procedure `new_situation` in Figure 5.30 does this. It checks whether the new situation has been analyzed before (`get_situation`). If this is the case, the disassembly costs of the two alternatives (paths through state space) are compared. When the new situation was reached with lower costs, it replaces the old situation. Note that the disassembly cost of situations which are successors of the old situation are also lowered by this replacement. For this reason, the final cost-benefit analysis can only be performed when all situations have been generated.

```
procedure new_situation(model, parent, action)
  s = get_situation(model)  (check whether situation already analyzed)
  if s found
    if disassembly cost of new situation < disassembly cost of s
      replace the action and the parent situation of s by
        action and parent
    endif
  else
    ns = list(model, parent, action)  (create new situation)
    add_situation(ns)                 (add ns to situation)
    disassemble_situation(ns)
  endif
end
```

Figure 5.30: Procedure to handle a new situation.

New situations that are not already in the situation list are added to it and disassembled further by the procedure `disassemble_situation`. This procedure can be found in figure 5.31. It generates the successor situations and calls `new_situation` for each of them.

```
procedure disassemble_situation(sit)
  m = model of sit
  clumpasies = get_assemblies(any_connection, m)     (determine clumpasies)
  freeasies = get_assemblies(rigid_or_enforced, m)   (determine freeasies)
  freeasies = add_asy_state(freeasies, m)            (add asy state to freeasies)
  sts = eval_states(clumpasies, m)                   (determine disassembly states)
  foreach connection c
    if is-breakable(c)
      action = loosen c
      m' = loosen_connection(c, m)       (m' = copy of m with c loosened)
      new_situation(m', m, action)
    endif
  end
  foreach freeasy a
    if state of a is free
      action = remove a
      m' = remove_asy(a, m)              (m' = copy of m with a removed)
      new_situation(m', m, action)
    endif
  end
end
```

Figure 5.31: Procedure to disassemble a situation.

The function `get_assemblies` is an implementation of the method to determine the maximal connected subgraphs. It will be described in detail below. One of its arguments is a function that must return *true* for connection that must be regarded as internal connections of the subgraph. With the function `any_connection`, which returns *true* for every connection that is not disconnected, `get_assemblies` returns a list of all clumpasies in the model. With the function `rigid_or_enforced`, which returns true for rigid connections and enforced loose connections, a list of all freeasies is returned.

The function `add_asy_status` prepends all subassemblies in `freeasies` with their state, according to the definition in Figure 5.14. This function will be described below.

The functions `loosen_connection(c,m)` and `remove_asy(a,m)` both make a copy of the model `m` and modify the connections in it. The former will change the connection type of the connection to loose and the latter will change the `break` attribute of all external connections of subassembly `a` to `disconnected`, as has been formalized with the relation `state-op` in the ontology for disassembly. Both functions will update the *intact* attributes of the loops in the model.

Determination of Subassemblies

The procedure that determines the subassemblies in a model can be found in Figure 5.32. Clearly, it is an implementation of the method to determine the maximal connected subgraphs in Figure 5.12. One difference is that the graph theoretical expression in the algorithm have been implemented by functions that inspect PROMOD's data structures. This caters for the knowledge-representation integration. A second change is that the procedure takes a function as argument that defines which connections have to be regarded as internal connections of the subassemblies. This way, the algorithm can be used to determine freeasies (with property function `any_connection`) or subasies (with `rigid_or_enforced`). Furthermore, the iterative constructs `while there is` in the method in Figure 5.32 have been worked out using a nested `while` and `foreach` loop.

Determination of Freeasy States

In order to find out which freeasies can be removed, it must be checked which of them satisfy the `free` predicate defined in Figure 5.14. Such an expression with logical operators and universal and existential quantors can be implemented as a search over the quantified variables. This is what is done in the algorithm in Figure 5.33. It is used by the function `add_asy_state` to prepend the states to the freeasies in the freeasy list. A small extension has made it possible to distinguish between subassemblies that are free to be removed, subassemblies that cannot be removed because they are blocked by others, and subassemblies that already have been removed (are separated from the rest of the product).

First it is checked whether there are external connections (that have not been disconnected) at all. If not, the subassembly is *separated* from the rest of the product. Otherwise, it is assumed

```
algorithm get_subassemblies(property p)
  subassemblies = {}
  while exists object o that does not belong to a subassembly do
    newassembly = {o}
    changed = true
    while changed do
      changed = false
      foreach object p that does not belong to a subassembly do
        if p is connected to an object in newassembly
           by connection c and c has property p
          add p to newassembly
          changed = true
        endif
      end
    end
    add newassembly to subassemblies
  end
end

function any_connection(c)
  return true iff c is not disconnected
end

function rigid_or_enforced(c)
  return true iff connection c is rigid or c is enforced
                  and not disconnected
end
```

Figure 5.32: Algorithm to determine the subassemblies of a model.

that the subassembly is blocked. As soon a an external connection can be found for which all loops going through it lead to disconnected connections or to connections on the same side of the subassembly, the assumption is wrong and the subassembly is free. To determine this, first it is assumed that there is no loop that leads to a blocking external object. All loops are considered and when one loop is found that lead to an object that does block, the assumption is withdrawn. When an external connection is found that has no loops leading to blocking objects, the subassembly status is changed to *free*.

In the prototype, the force loop groups described in Section 5.2 have been implemented. With that extension, groups have to play the role of the loops in the algorithm in Figure 5.33. A new foreach-loop has to be inserted to check whether a group contains loops such that the subassembly is blocked. This is done by first assuming that all loops of a group lead to blocking objects. As soon as one loop is found for which this is not the case, the assumption is withdrawn. Also, some speed up has been achieved by exiting from a loop as soon as an assumption is withdrawn.

Determination of Disassembly State Values

The function `eval_states(clumpasies,m)` in Figure 5.34 determines the truth values of the states of model `m`. First, it computes the values of the *separate* and *disconnected* states. Then, it tries to derive other state values. This will be repeated until no new state value can be determined. When there are some states left for which the value could not be determined an error message will be generated.

```
algorithm to determine the state of subassembly s
  ecs = all external connections of s
  if ecs is empty or all connections in ecs are disconnected
    state = separate
  else
    state = blocked                  (assume all connections are blocked)
    foreach connection c1 in ecs that is not disconnected
      no-loop-blocks = true          (assume all loops are free)
      foreach loop l that goes through c1
        foreach connection c2 in ecs
          if c1 <> c2 and c2 in l and c2 is not disconnected
               ((c1 is against push and c2 is against push) or
                (c1 is against pull and c2 is against pull))
            no-loop-blocks = false   (a loop is blocking)
          endif
        end
      end
      if no-loop-blocks = true       (all loops are free)
        state = free                 (a connection is free)
      endif
    end
  endif
end
```

Figure 5.33: An algorithm to determine the state of a subassembly.

```
function eval_states(clumpasies, m)
  sts = ()
  foreach disassembly state definition s in m
    if s is a disconnected or separate state
      determine the value of s and add
        (s true) or (s false) to sts
  end
  changed = true
  while changed = true
    changed = false
    foreach state definition s in m
      if the value of s has not been determined yet and
         all argument states of s are known
        determine the value of s and add
          (s true) or (s false) to sts
        changed = true
      endif
    end
  end
  if the value of a state could not be determined
    generate error message
  endif
  return sts
end
```

Figure 5.34: Function to determine the truth value of product states.

5.5 Discussion and Related Research

This section discusses three important aspects of the modelling technique and the implementation of the prototype described in this chapter. In Section 5.5.1 we will discuss which extensions of the modelling technique and prototype are required before it can be used to perform realistic disassembly analyses. Section 5.5.2 discusses how our viewpoint on method ontologies relates to the literature on this subject. Finally, in Section 5.5.3 the way the geometric abstractions in PROMOD models can be defined in terms of full geometry is discussed.

5.5.1 Disassembly Analysis

A more realistic cost-benefit analysis requires that *physical* disassembly operations are considered. In the present prototype, only the model changes that are caused by physical disassembly operations are considered. The ecological cost of a physical disassembly operation depends on the type of the physical connection (weld, screw, glue, etc.), the size of the connection, its orientation, the materials of the connected objects, the size and mass of the objects, the previous operations performed and so on. This has to be studied further. Good starting points might be (Ishii 1996; Ishii, Lee, and Eubanks 1995; Ishii, Eubanks, and Marks 1993; Sturges Jr and Kilani 1992). Disassembly analysis based on physical disassembly operations requires extension of the product models and changes in the procedure for disassembly analysis in Figure 5.31.

To be able to deal not only with recycling, but also with reuse, the concept of components must be added. A component is a group of objects that can perform a certain function, like the components defined in the OLMECO library. For recycling of *clumps* we are only interested in the materials of the objects in the clump. For reuse of *components*, not only the objects in it are of importance, but also the connections that hold these objects together. Unlike clumps, components must be removed from the product intact, without breaking their internal connections.

Instead of the object penalty that is used in the prototype to determine the (reciprocal of the) benefits of disassembly, a true benefit analysis needs to be performed. This can be done by assessing the ecological benefits for recycling or reuse with an ecological life cycle analysis (LCA). Consequently, a realistic cost-benefit analysis can only be achieved by integration of the prototype in LCA software.

The need to incorporate LCA analyses in the disassembly analysis process prompts us to have a better look at the combinatoric complexity of the prototype. It is very likely that a search strategy more intelligent compared to the brute force method used now has to be developed. This method will have to estimate the effect of disassembly operations for the removal of the clump, will try the most promising operations first, postpone the less promising and could even disregard the least promising. Here, a balance has to be found between always finding the best solution taking much time and finding a good solution in limited time.

Another useful extension of the prototype would be the development of a library of commonly used partial product structures, such as complex connections based on connector-objects (bolts an nuts) and case structures (for instance for consumer electronics). A library like OLMECO seems to be adequate for this purpose. Because product models can become large, a kind of blackboxing could be used to create hierarchical product models where detailed product structures are hidden from the top-level. This resembles OLMECO's decomposition structures.

5.5.2 Method Ontologies

Our view on method ontologies closely follows the literature on problem solving methods and method ontologies (Angele, Decker, Perkuhn, and Struder 1996; Fensel, Schönegge, Groenboom, and Wielinga 1996; Gennari, Tu, Rothenfluh, and Musen 1994). There, method ontologies are described that provide the terminology for expressing the competence and knowledge requirements of problem solving methods. In our opinion, the part of these method ontologies that specifies the terminology for a class of methods has to be separated from the parts that concern the competence and knowledge requirements for individual methods. The principles of ontology construction can be used for this modularization. With this modularization, the boundary between abstract ontologies, domain, method and application ontologies becomes vague because the the context in which an abstract ontology is used determines its characterization.

The terminological part of a method ontology is nothing more than an abstract ontology (for instance the state space ontology) which is projected onto ontologies specifying the competence and knowledge requirements of the (state space search) methods. Together they form a method ontology. But the same abstract ontology can be specialized to form a domain ontology. Then it becomes part of this domain ontology. An application ontology consists of a domain ontology, which is projected onto, or constructed using an abstract ontology that gives the terminological specification for the methods used for the application. Figure 5.35 gives an overview of these scenarios.

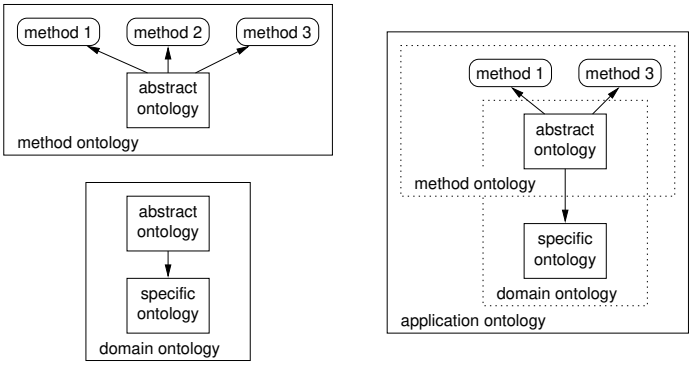


Figure 5.35: The role of abstract ontologies in domain, method and application ontologies.

5.5.3 Geometrical Ontologies

PROMOD's connection types and force loops are abstractions of physical and geometrical properties of connections and product structures. Although they form a sufficient basis for disassembly analysis, it would be interesting to study how they are related to ontologies of geometric and spatial reasoning. This would give us more information about the competence of the modelling technique and provides the knowledge of how to generate PROMOD models from CAD drawings in cases where these are available.

At least two approaches to formalize geometry can be found in the literature. One describes geometry in a mathematical way: mathematical equations define shapes, surfaces, lines and points in space. A system that uses a polygon representation of products to reason about mechanical assembly is described in (Wilson and Latombe 1994). The second approach extends Clarke's mereo-topology with relations to express congruency of objects (being aligned), being enclosed by an object, overlapping the convex hull of an object and so on (Borgo, Guarino, and Masolo 1996a; Randell and Cohn 1992). Because these approaches avoid the introduction of points in space, the theories are called `pointless' theories. A good example of the second approach to geometrical reasoning can be found in (Randell, Cohn, and Cui 1992).

For PROMOD's geometric abstractions, rigidness and congruency are the key aspects. Congruency is one of the basic relations in pointless approaches, so these theories are good candidates to specify PROMOD's abstractions. It is also possible to define congruence in a mathematical way. For congruent objects it is possible to find a straight line that crosses all objects. The rigidness of connections can also be expressed in both approaches. In pointless theories they coincide with so called strong connections. In mathematical approaches, connections can be formalized with mathematical relations about the position of shapes, surfaces, lines and points. The distinction between rigid and loose connections gives rise to two interpretations of these relations. For rigid connections the relation has to be interpreted as a constraint that *must* hold, and for loose connections it is just an observation about the position of the connected objects.

Both approaches seem to be suited for formalizing the physical and geometrical abstractions used in PROMOD. Pointless approaches are well suited to gain insight on the actual meaning of the abstractions whereas mathematical approaches are closer to solid model specifications and therefore better for linking PROMOD models to CAD drawings.

5.6 Conclusions about PROMOD and Reusing PHYSSYS

Product disassembly is not simply a matter of carrying out the assembly process in reverse order (see Section 5.1). Therefore, for disassembly analysis, specialized methods are required. In this chapter a new computational theory for ecological product disassembly analysis has been presented. Its most important feature is that it introduces unconventional geometrical abstractions in a topological context. Thus, advantages of several existing techniques found in the literature are combined.

We have demonstrated how this theory can be formalized in an ontology using the design principles described in Chapter 3. We were able to reuse the EngMath ontology as well as the ontologies of mereology, topology and systems theory from PHYSSYS. Furthermore, two new reusable abstract ontologies were introduced: that of graph theory and the state space ontology based on graph theory.

Three views on product disassembly were distinguished: (i) a static view on product models for disassembly, (ii) the way these models can be modified by disassembly operations and form a disassembly state space and (iii) the ecological view on product disassembly. The ecological cost of the disassembly has to be compared to the ecological benefits of recycling or reuse of the parts removed from the product. At the moment, this cost-benefit relation demonstrates the principles of ontology projection nicely, but is too limited for practical purposes (see also Section 5.5).

The ontology for product models for disassembly has been constructed from ontologies of systems theory and graph theory, enriched with physico-chemical connection properties and force loops. The system theoretical point of view is convenient for expressing the model–subassemblies–connected-objects hierarchy in the product models, which must comply to the mereo-topological axioms. Graph theory on the other hand is better suited for expressing the force loop structures at the connected objects level and for expressing the dependency between subassemblies and properties of the internal and external connections of these subassemblies. Graph theory and systems theory are different abstract views to look upon different parts of the knowledge captured in product models for disassembly.

We have also seen that in the literature, both in AI and general computer science, general methods can be found that compute instances and properties of concepts defined in abstract ontologies. The idea is that these methods specify how to perform certain tasks in a *domain independent* way. Examples used in PROMOD are the computation of the maximal connected subgraphs and the exhaustive state space search, but many others can be found. We have not talked in detail about the exact specification of the competence and knowledge requirements of these methods. More on this subject can be found in (Fensel, Schönegge, Groenboom, and Wielinga 1996). The methods that were mentioned in this chapter were monolithic. In a more complex view on methods, they can be decomposed into submethods (Puerta, Egar, Tu, and Musen 1992; Fensel, Schönegge, Groenboom, and Wielinga 1996).

The most important question regarding the the use and reuse of general problem solving methods is how to find a proper method for a *domain specific* task. In the case of PROMOD we have seen that some of them are found in a natural way because the abstract ontologies for which the methods are defined appear also in the formalization of the domain knowledge. This has been the case with the state space search methods. For the method used to find the subassemblies something different has been the case. In order to find the method that solves this task we had to discover that graph theory is a valid way to look upon a part of the knowledge captured in the product models. In other words, the problem in the domain of product disassembly had to be transformed into an equivalent problem in the abstract domain of graph theory. Finding a method for a domain problem therefore requires four steps:

1. Construct an initial domain ontology.
2. See whether the domain knowledge can be projected onto an abstract ontology which has general problem solving methods defined for it.
3. Translate the domain independent task performed by the methods to domain concepts and see whether this results in the domain problem to be solved.
4. Select a method with adequate competence for which the knowledge requirements can be fulfilled.

Once the application ontology has been constructed, it can be used as an information-system specification. First, data structures have to be designed that can store the knowledge defined in the application ontology. Usually, this means that abstract viewpoints on the domain knowledge will be combined in one representation. Although this knowledge structure is then no longer visible, it can be used to describe the implementation in the design documentation.

The selected methods in the application ontology are a great help for the implementation of the functions and procedures of the application. Because the abstract viewpoints for which the methods are defined are combined into one knowledge representation, the methods have to be adapted to make inferences on this new representation. Another possibility is to write functions that translate information from the data structures (e.g. in terms of lists) to the abstract representation (e.g. vertices and edges). This way, less changes to the methods are required.

During the design of the prototype, we have seen that the axioms that define what valid models look like provide information for the design of the input parser, user interface, the functions and the procedures. The parser and the user interface of the application are designed in such a way that they only accept and store valid models. The design of the procedures and functions that manipulate the data ensure that this initial consistency is preserved.

Chapter 6

Conclusions

In this chapter we present the conclusions of our research. First we will draw the conclusions about the usability and reusability of ontologies for large applications. In the next parts we will present the advances of our work for ontological engineering, compare it to the object patterns approach and describe the role ontologies may play in the future.

In this research we have investigated the usability and reusability of ontologies for a large and complex domain. In Chapter 3 we showed that a complex ontology for the modelling of physical systems called PHYSSYS can be constructed out of smaller, reusable ontologies. In Chapter 4 we demonstrated that PHYSSYS can be *used* as a specification of a library of model fragments that supports the construction of simulation models in an industrial setting. A modelling and simulation experiment of a large heating system proved the usability and validity of PHYSSYS. In Chapter 5, parts of PHYSSYS were *reused* in the construction of an ontology for an application in a different engineering domain, that of ecological product disassembly analysis.

6.1 Advances of the Research for Ontological Engineering

In this thesis we have shown that ontologies can be a useful instrument for knowledge sharing and reuse. We have investigated the nature, construction and practical role of ontologies as mechanisms for knowledge sharing and reuse for some real-life industrial applications. For each of these three aspects we will summarize our main results and insights below.

6.1.1 Modularization of Ontologies

Concerning the nature of ontologies, we have discussed the development of an ontology collection called PHYSSYS (Chapter 3) that covers a wide, multidisciplinary range of physical systems and their engineering. This collection contains different types similar to the distinctions proposed in (van Heijst, Schreiber, and Wielinga 1997): highly generic ontologies (mereology, topology, systems theory), base ontologies valid for a whole field (e.g. technical components, physical processes, representing natural categories or viewpoints within a broad field) and domain ontologies (specializations of base ontologies to a specific domain, e.g. thermodynamics). We have indicated how we can extend this to method ontologies (Chapter 5) and how we can exploit the whole collection as the basis for application ontologies (Chapter 4 and 5). Employing the distinctions between the mentioned types of ontologies is a natural and operational way of organizing a library of ontologies in a modular fashion.

6.1.2 Ontology Construction

Concerning the construction of ontologies (Chapter 3 and 5) our main conclusions are:

1. *Use and reuse of `super' theories.* We have shown in detail that there are highly general `super' theories which can be employed to gradually develop large domain ontologies in a structured fashion. In our case, we have used and reused generic ontologies concerning mereology, topology and general systems theory, but it is not difficult to imagine other useful supertheories. This approach enhances both the modularity and the reusability of ontologies.

2. *Distinguishing natural `viewpoints' or base categories.* In knowledge acquisition one finds that it is often possible to distinguish broad natural `viewpoints' or base categories within a field. These broad conceptual distinctions can then be exploited to develop separate base ontologies which are valid and reusable across many subdomains and tasks. In our application, these distinctions refer to groups of properties that are seen as naturally belonging together. For example, we can view an engineering system as a device configured out of known `hardware' components, or as a collection of physical processes determining its dynamic behaviour, as a thing possessing a certain three-dimensional shape, or as being composed out of different materials. Distinguishing and separating such basic viewpoints appears to be an important structuring principle in ontology building: giving rise to strong internal coherence and weak coupling.

3. *Ontology projections.* We have introduced `ontology projections' as a flexible mechanism to link and configure ontologies into larger ones. There are different types of ontology projections. First, we have a technique called include-and-extend, whereby several theories are included and extended with axioms at the same level of abstraction (example: the specification of topology and general systems theory). A second technique is include-and-specialize, whereby several ontological theories are included and subsequently are specialized to a domain by instantiation, term and concept mappings

and additional specific axioms (example: the process ontology). By taking different abstract viewpoints on a domain (systems theory and graph theory viewpoints on disassembly models), not only convenient ways to specify domain knowledge were found, it also led to the method to find maximal connected subgraphs which could be used in the application ontology of PROMOD. Finally, a third, new type is what we have called `include-and-map'. Here, the connection between two ontologies itself assumes the form of a full blown ontological theory. An example here is the connection between the PHYSSYS process ontology and the EngMath ontology. This example is moreover interesting because it exemplifies the reuse of an outside ontology developed by another research group in a different context.

4. *Piecemeal ontological commitment.* Together, the above mechanisms provide a pragmatic approach for handling piecemeal ontological commitments in ontology development. In applications it is not so much strictly minimal ontological commitment that we want, but achieving the *right* commitment. This, however, needs to be built up starting from the minimal side, and step-by-step extending this by adding small additional commitments.

We would like to underline the important role of abstract ontologies in the construction process. Not only are they useful for the specification of domain knowledge, they are also the `glue' between domain and method ontologies. It has been encouraging to see that useful methods could be found in general computer science literature.

6.1.3 Practical Role of Ontologies

Concerning the practical role of ontologies, we believe that a key aspect is their capability to explicate in detail tacit background knowledge required for real-life tasks. Acquiring and analyzing this background knowledge is hard, because it is often seen as `self-evident' by domain experts and practitioners and much of it is implicitly shared by the associated community — this is precisely why it is tacit knowledge. Bringing out this tacit knowledge is important for two reasons: (i) to find out what is really shared by the community in order to enhance reuse *within* this community; (ii) to develop more knowledgeable information systems that provide intelligent support for end users that are less experienced, or are from a related but different community, thus facilitating knowledge transfer *between* communities. In this thesis, we have given extensive and real-life illustrations of this for the domain of engineering modelling, simulation and systems design (Chapter 4), and design for the environment (Chapter 5). We have seen that from the PHYSSYS ontology the database schema of OLMECO could be determined in a quite natural way, with minor changes and extensions. In Chapter 5 we have seen that the ontological definitions served as a specification for the data structures, input parser and other procedures of the KBS prototype. The most important procedures could be implemented using to off the shelf methods. A final note of interest is that our applications has been implemented in various kinds of conventional information systems. Thus, the scope and usefulness of knowledge engineering is much wider than knowledge-based systems alone.

6.2 Ontologies and Object Patterns

Both ontologies and object analysis/design patterns are instruments for knowledge sharing and reuse. Ontologies have a number of properties that make them, in our opinion, preferable to object patterns:

- Ontologies cater for a much more detailed specification of the properties of objects and relations between them. Conceptual schemata can be used to give overviews of the contents of an ontology that match the clarity of object patterns.
- Unlike patterns, ontologies can exploit the internal structure of reusable knowledge by distinguishing multiple levels of abstraction. This makes it possible that knowledge of a higher degree of reusability can be specified.
- Ontologies can be constructed using ontology projections. For patterns, no such construction operators have been defined. Ontology projections are important because they allow complex bodies of reusable knowledge to be constructed out of smaller modules.
- Ontologies are not limited to object oriented knowledge representations and methods.

This shows that ontologies are a step forward for knowledge sharing and reuse. It would be fruitful to combine the expressiveness and high reusability of ontologies with the suitability for application implementation of object patterns.

6.3 The Role of Ontologies in the Future

We believe that ontologies can play an important role in future developments of information technology. They can play a useful role when knowledge intensive applications are being developed or have to be integrated. Because ontologies are specifications at the knowledge level, their applicability is not limited to information systems and knowledge bases alone, but includes knowledge intensive and possibly distributed databases, multi-agent systems, knowledge exchange over the Internet, multimedia systems etcetera. In the following list of future roles of ontologies we are referring to applications in this broader sense.

Application Integration: Because domain ontologies make explicit the similarities and differences between the knowledge used by applications, it will be easier to design interfaces and translators that allow knowledge exchange between different applications.

Better Applications: A library of well thought of and validated ontologies will not only speed up application development, but will also improve the quality of the applications. Furthermore, the reuse of ontologies will lead to uniformity across applications in a domain and ensure that the knowledge of applications in related domains share a large common base. In both situations, application integration will become easier.

Open Applications: Ontology libraries will show application developers which applications in other domains can benefit from an application's knowledge. They can improve the usability of their application by enabling it to cooperate with these other applications. For instance, only minor additions to CAD drawing representations are required to include product data for life cycle analysis, disassembly analysis, computer aided manufacturing etc.

Innovative Applications: A library with abstract and reusable ontologies will be an excellent basis for the development of ontologies for new domains. This will enable the development of innovative applications. New applications can be expected in fields that can benefit from a specification of the domain knowledge, such as knowledge extraction from natural language, data mining, visualisation of numerical data and search engines for on-line libraries and the Internet.

Application Generation: The development of method ontologies and better ontology specification systems may result in tools that can automatically generate (the framework of) applications from application ontologies.

Before these developments can take place, ontology specification systems need to be improved in the following ways:

Support for Ontology Construction: Ontology specification systems have to be improved to better support ontology construction. The different types of ontologies and the types of ontology projections used for their construction have to be made explicit.

Better Libraries: A better way to access ontologies in an ontology library has to be developed. The indexing mechanism to accomplish this has to take into account the knowledge-content and applications of the ontologies. Furthermore, it must give an indication of the similarities and differences between the ontologies in the library.

More Reusable Ontologies: More ontologies have to be incorporated in ontology libraries because they are the building blocks for ontologies for new domains.

Support for Application Generation: Mechanisms for (semi-)automatic generation of efficient applications from application ontologies have to be developed.

Appendix A

The PHYSSYS Library of Ontologies

This appendix gives the Ontolingua 4.0 specification of the PHYSSYS *library of ontologies for physical systems as described in Chapter 3.*

The organization of this appendix is as follows.

A.1: Ontolingua implementation of Classical Extensional Mereology.

A.2: Topological extension to Mereology.

A.3: System theoretical concepts and relations on top of topology.

A.4: The technical component viewpoint of PHYSSYS.

A.5: The physical process viewpoint of PHYSSYS

A.6: The top-level ontology of PHYSSYS. This ontology includes the component, process and mathematical viewpoints and defines the mapping relations that combines the three viewpoint.

For a description of the EngMath ontology for engineering mathematics we refer to (Gruber 1994).

A.1 Ontology of Mereology

```
; ------------------------------------------------------ ;
; Classical Extensional Mereology                        ;
;                                                        ;
; System SC of Classical Mereology:                      ;
;   no sets; proper-part-of primitive; equal assumed     ;
; Part of the PhysSys ontology                           ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

(define-theory classical-extensional-mereology
               (frame-ontology kif-relations))

(in-theory 'classical-extensional-mereology)

(define-relation equal (?x ?y)
  :axiom-def
    (binary-relation equal))

(define-class m-individual (?i)
  :iff-def
    (equal ?i ?i))

(define-class m-individual-class (?C)
  :iff-def
    (and (class ?C)
         (=> (instance-of ?i ?C) (m-individual ?i))))

(define-relation proper-part-of (?x ?y)
  "SA1:  proper-part-of is asymmetric.
   SA2:  proper-part-of is transitive.
   SA3:  a whole has at least two disjoint proper-parts.
   SA24: general sum principle."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :axiom-def
    ((=> (proper-part-of ?x ?z) (not (proper-part-of ?z ?x)))  ; SA1
     (=> (and (proper-part-of ?x ?y) (proper-part-of ?y ?z))   ; SA2
         (proper-part-of ?x ?z))
     (=> (proper-part-of ?x ?y)                                ; SA3
         (exists ?z (and (proper-part-of ?z ?y)
                         (disjoint ?z ?x))))
     (=> (and (m-individual ?x) (instance-of ?x ?C))           ; SA24
         (exists ?x
           (forall ?y (<=> (overlap ?y ?x)
                           (exists ?z (and (instance-of ?z ?C)
                                           (overlap ?y ?z)))))))))


; ---- Definitions ----

(define-relation part-of (?x ?y)
  "SD1: ?x is a proper-part-of ?y or equal to ?y."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (or (proper-part-of ?x ?y)
        (equal ?x ?y)))

(define-relation overlap (?x ?y)
  "SD2: overlapping m-individuals share a part."
```

```
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (exists ?z (and (part-of ?z ?x)
                    (part-of ?z ?y))))

(define-relation disjoint (?x ?y)
  "SD3: disjoint m-individuals do not have a common part."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (not (overlap ?x ?y)))

(define-function general-sum (?C) :-> ?x
  "SD9: the general sum of the m-individuals in class ?C is that m-individual
        which something overlaps iff it overlaps at least one m-individual
        of ?C"
  :when
    (m-individual-class ?C)
  :iff-def
    (forall ?y (<=> (overlap ?x ?y)
                    (exists ?z (and (instance-of ?z ?C)
                                    (overlap ?z ?y))))))

(define-function binary-product (?x ?y) :-> ?z
  "SD4: the binary product of ?x and ?y is that m-individual which is part
        of both and which is such that any common part of both ?x and ?y
        is part of it."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (= ?z (general-sum (kappa (?v) (and (part-of ?v ?x)
                                        (part-of ?v ?y))))))

(define-function binary-sum (?x ?y) :-> ?z
  "SD7: the binary sum of ?x and ?y is that m-individual which something
        overlaps iff it overlaps at least one of ?x and ?y."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (= ?z (general-sum (kappa (?v) (or (part-of ?v ?x)
                                       (part-of ?v ?y))))))

(define-function general-product (?C) :-> ?x
  "SD10: the general product of the m-individuals in class ?C is that
         m-individual which is part of all m-individuals in ?C and which is
         such that any common part of all m-individuals in ?C is part of it."
  :when
    (m-individual-class ?C)
  :iff-def
    (= ?x (general-sum (kappa (?v) (forall ?y (=> (instance-of ?y ?C)
                                                  (part-of ?v ?y)))))))

(define-function difference (?x ?y) :-> ?z
  "SD11: the difference of ?x and ?y is the largest m-individual contained
         in ?x which has no part in common with ?y."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (= ?x (general-sum (kappa (?v) (and (part-of ?v ?x)
                                        (disjoint ?v ?y))))))

(define-class m-universe (?U)
  "SD12: the m-universe is a unique m-individual of which
         all m-individuals are part."
  :when
    (m-individual ?U)
```

```
  :iff-def
    (= ?U (general-sum (kappa (?v) (m-individual ?v)))))

(define-function complement (?x) :-> ?c
  "SD13: the complement of ?x is the unique m-individual comprising the rest
         of the m-universe."
  :when
    (m-individual ?x)
  :iff-def
    (exists ?U (and (m-universe ?U)
                    (= ?c (difference ?U ?x)))))

; additional definitions

(define-relation direct-part-of (?x ?y)
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (and (proper-part-of ?x ?y)
         (not (exists ?z (and (proper-part-of ?z ?y)
                              (proper-part-of ?x ?z))))))

(define-relation proper-overlap (?x ?y)
  "Two m-individuals properly overlap eachother when they overlap and neither
   is part of the other."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (and (overlap ?x ?y)
         (not (part-of ?x ?y))
         (not (part-of ?y ?x))))

(define-relation upper-bound (?C ?z)
  "SD5: ?z is an upper bound for m-individuals in ?C if
        they are all part of ?z."
  :when
    (and (m-individual-class ?C) (m-individual ?z))
  :iff-def
    (and (exists ?x (instance-of ?x ?C))
         (forall ?x (=> (instance-of ?x ?C)
                        (part-of ?x ?z)))))

(define-function general-least-upper-bound (?C) :-> ?x
  :when
    (m-individual-class ?C)
  :iff-def
    (forall ?y (<=> (part-of ?x ?y)
                    (forall ?z (=> (instance-of ?z ?C)
                                   (part-of ?z ?y))))))

(define-function least-upper-bound (?x ?y) :-> ?z
  "SD6: the least upper bound of ?x and ?y is that m-individual which is part
        is part of something iff both ?x and ?y are also."
  :when
    (and (m-individual ?x) (m-individual ?y))
  :iff-def
    (forall ?w (<=> (and (part-of ?x ?w) (part-of ?y ?w))
                    (part-of ?z ?w))))

(define-relation atom (?x)
  "SD14: an atom is an m-individual with no proper parts."
  :when
    (m-individual ?x)
  :iff-def
    (not (exists ?z (proper-part-of ?z ?x))))

(define-class simple-m-individual (?i)
```

```
  :iff-def
    (and (m-individual ?i)
         (atom ?i)))
```

A.2 Ontology of Topology

```
; ------------------------------------------------------ ;
; Topology                                               ;
;                                                        ;
; Topological relations on top of mereology              ;
; Part of the PhysSys ontology                           ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

(onto-load "mereology.lisp")

(define-theory topology (frame-ontology kif-relations
                         classical-extensional-mereology))

(in-theory 'topology)

(define-class t-connection (?c)
  :iff-def
    (and
      (not (m-individual ?c))
      (exists (?x ?y) (connects ?c ?x ?y))))

(define-relation connected (?x ?y)
  :iff-def
    (exists ?c (connects ?c ?x ?y)))

(define-relation unconnected (?x ?y)
  :iff-def
    (not (connected ?x ?y)))

(define-relation connects (?c ?x ?y)
  :def
    (and (t-connection ?c)
         (m-individual ?x)
         (m-individual ?y))
  :axiom-def
    ((=> (connects ?c ?x ?y)            ; Symmetry
         (connects ?c ?y ?x))
     (=> (connects ?c ?x ?y)            ; Parts and wholes not connected
         (not (or (part-of ?x ?y)       ; Includes irreflexivity
                  (part-of ?y ?x))))
     (=> (and (connects ?c ?x ?y)       ; 'transivity'
              (part-of ?x ?z)
              (disjoint ?z ?y))
         (connects ?c ?z ?y))
     (=> (and (connects ?c ?x ?y)       ; connection to whole also to parts
              (not (simple-m-individual ?y)))
         (exists ?z (and (part-of ?z ?y)
                         (connects ?c ?x ?z))))
     (forall ?y1
       (=> (and (connects ?c ?x1 ?y1)   ; connection is _one_ line
                (connects ?c ?x2 ?y2))  ; and it doesn't fork
           (not (and (disjoint ?x1 ?x2)
                     (disjoint ?x1 ?y2)))))))
```

A.3 Ontology of Systems Theory

```
; ------------------------------------------------------ ;
; System Theory                                          ;
;                                                        ;
; System theoretic relations on top of topology          ;
; Part of the PhysSys ontology                           ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

(onto-load "topology.lisp")

(define-theory system-theory (frame-ontology kif-relations
                              topology classical-extensional-mereology))

(in-theory 'system-theory)

(define-class system (?s)
  :def
    (m-individual ?s))

(define-relation in-system (?x ?s)
  :iff-def
    (and (proper-part-of ?x ?s)
         (system ?s)
         (simple-m-individual ?x)))

(define-relation in-boundary (?c ?s)
  :iff-def
    (and (t-connection ?c)
         (system ?s)
         (exists (?x ?y)
           (and (connects ?c ?x ?y)
                (part-of ?x ?s)
                (not (part-of ?y ?s))))))

(define-relation subsystem-of (?sub ?sup)
  :iff-def
    (and (system ?sub)
         (system ?sup)
         (proper-part-of ?sub ?sup)))

(define-class open-system (?s)
  :iff-def
    (and (system ?s)
         (exists (?c) (in-boundary ?c ?s))))

(define-class closed-system (?s)
  :iff-def
    (and (system ?s)
         (not (open-system ?s))))
```

A.4 Component Ontology

```
; ------------------------------------------------------ ;
; Component View                                         ;
;                                                        ;
; Technical viewpoint on a physical system               ;
; Viewpoint of the PhysSys ontology                      ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

(onto-load "systheory.lisp")

(define-theory
  component-view
  (frame-ontology kif-relations
   classical-extensional-mereology topology system-theory))

(in-theory 'component-view)

(define-class component (?c)
  "components are individuals, components are atomic or decomposed in more
   than one subcomponent, components do not overlap (except for part-wholes)"
  :def
    (and
      (value-type ?c comp.term terminal)
      (>= (value-cardinality ?c comp.term) 1)
      (value-type ?c comp.subcomp component)
      (/= (value-cardinality ?c comp.subcomp) 1)
      (m-individual ?c)
      (not (exists ?c1 (proper-overlap ?c ?c1)))))

(define-relation comp.subcomp (?c ?s)
  :iff-def
    (and
      (component ?c)
      (component ?s)
      (direct-part-of ?s ?c)))

(define-relation comp.term (?c ?t)
  :axiom-def
    (and (domain comp.term component)
         (range comp.term terminal)
         (function comp.term)))

(define-class terminal (?t)
  :def
    (and
      (value-type ?t term.term-type terminal-type)
      (value-cardinality ?t term.term-type 1)))

(define-relation term.term-type (?t ?tt)
  :axiom-def
    ((domain term.term-type terminal)
     (range term.term-type terminal-type)
     (function term.term-type)))

(define-class terminal-type (?tt))

; some examples of terminal types
(define-instance mech-trans (terminal-type))
(define-instance mech-rot (terminal-type))
(define-instance electric (terminal-type))
```

```
(define-instance pneumatic (terminal-type))
(define-instance hydraulic (terminal-type))
(define-instance thermal (terminal-type))
(define-instance hydro-thermal (terminal-type))

(define-class connection (?c)
  "a connection is a topological connection connecting components
   a connection connects two terminals of the same type
   a connection establishes a topological connection between the components
   a topolological connection between components needs a connection"
  :def
    (and
      (t-connection ?c)
      (=> (connects ?c ?c1 ?c2) (and (component ?c1) (component ?c2)))
      (value-type ?c con.term terminal)
      (value-cardinality ?c con.term 2)
      (=> (and (con.term ?c ?t1)
               (con.term ?c ?t2))
          (exists ?tt (and (term.term-type ?t1 ?tt)
                           (term.term-type ?t2 ?tt)))))
  :axiom-def
    (and
      (=> (and (connection ?c) (con.term ?c ?t) (comp.term ?c1 ?t))
          (exists ?c2 (connects ?c ?c1 ?c2)))
      (forall ?c2
        (=> (and (component ?c1) (connection ?c)
                 (connects ?c ?c1 ?c2))
            (exists (?t)
              (and (terminal ?t)
                   (con.term ?c ?t)
                   (comp.term ?c1 ?t)))))))

(define-relation con.term (?c ?t)
  :axiom-def
    ((domain con.term connection)
     (range con.term terminal)
     (function (inverse con.term))))

(define-relation con.type (?c ?t)
  :iff-def
    (exists (?t1 ?t2)
      (and (con.term ?c ?t1)
           (con.term ?c ?t2)
           (/= ?t1 ?t2)
           (term.term-type ?t1 ?t)
           (term.term-type ?t2 ?t))))

(define-relation comp.con (?c ?con)
  :iff-def
    (exists ?t (and (comp.term ?c ?t)
                    (con.term ?con ?t))))

(define-class phys-system (?s)
  :iff-def
    (and (system ?s)
         (=> (in-system ?c ?s) (component ?c))))
```

A.5 Physical Process Ontology

```
; ------------------------------------------------------ ;
; Process View                                           ;
;                                                        ;
; Physical process viewpoint on technical systems        ;
; Viewpoint of the PhysSys ontology                      ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

; (onto-load "systheory.lisp")

(define-theory process-view
              (frame-ontology kif-relations
               classical-extensional-mereology topology system-theory))

(in-theory 'process-view)


; -- Stuff, effort and flow -- ;

(define-class stuff (?s)
  "The class of different types of stuff.
   Stuff are the things that can be stored.
   Dynamic behaviour is change of stuff.")

(define-instance displacement (stuff))
(define-instance angle (stuff))
(define-instance volume (stuff))
(define-instance charge (stuff))
(define-instance entropy (stuff))

(define-class effort (?e)
  "The class of different types of effort.
   Effort is needed to bring about a change of stuff.")

(define-instance force (effort))
(define-instance torque (effort))
(define-instance pressure (effort))
(define-instance voltage (effort))
(define-instance temperature (effort))

(define-class flow (?f)
  "The class of different types of flow.
   Flow is change of stuff.")

(define-instance velocity (flow))
(define-instance ang-velocity (flow))
(define-instance volume-flow (flow))
(define-instance current (flow))
(define-instance entropy-flow (flow))


; -- Physical Domains -- ;

(define-class physical-domain (?d)
  "The class of physical domains. Each domain must have one stuff type
   associated to it with the relation phys-dom.stuff, one effort type
   with phys-dom.effort and one flow type with phys-dom.flow."
  :def
    (and
      (value-type ?d phys-dom.stuff stuff)
```

```
      (value-cardinality ?d phys-dom.stuff 1)
      (value-type ?d phys-dom.stuff effort)
      (value-cardinality ?d phys-dom.effort 1)
      (value-type ?d phys-dom.flow flow)
      (value-cardinality ?d phys-dom.flow 1)))

(define-relation phys-dom.stuff (?d ?s)
  "Relates a kind of stuff to a domain."
  :axiom-def
    ((domain phys-dom.stuff physical-domain)
     (range phys-dom.stuff stuff)
     (function phys-dom.stuff)))

(define-relation phys-dom.effort (?d ?e)
  "Relates a kind of effort to a domain."
  :axiom-def
    ((domain phys-dom.effort physical-domain)
     (range phys-dom.effort effort)
     (function phys-dom.effort)))

(define-relation phys-dom.flow (?d ?f)
  "Relates a kind of flow to a domain."
  :axiom-def
    ((domain phys-dom.flow physical-domain)
     (range phys-dom.flow flow)
     (function phys-dom.flow)))

; Now some physical domains are defined.
; This set is not exhaustive.

(define-instance mech-trans (physical-domain)
  "Translational mechanics"
  :axiom-def
    ((phys-dom.stuff mech-trans displacement)
     (phys-dom.effort mech-trans force)
     (phys-dom.flow mech-trans velocity)))

(define-instance mech-rot (physical-domain)
  "Rotational mechanics"
  :axiom-def
    ((phys-dom.stuff mech-rot angle)
     (phys-dom.effort mech-rot torque)
     (phys-dom.flow mech-rot ang-velocity)))

(define-instance electric (physical-domain)
  "Electric"
  :axiom-def
    ((phys-dom.stuff electric charge)
     (phys-dom.effort electric voltage)
     (phys-dom.flow electric flow)))

(define-instance pneumatic (physical-domain)
  "Pneumatic"
  :axiom-def
    ((phys-dom.stuff pneumatic volume)
     (phys-dom.effort pneumatic pressure)
     (phys-dom.flow pneumatic volume-flow)))

(define-instance hydraulic (physical-domain)
  "Hydraulic"
  :axiom-def
    ((phys-dom.stuff hydraulic volume)
     (phys-dom.effort hydraulic pressure)
     (phys-dom.flow hydraulic volume-flow)))


(define-instance thermal (physical-domain)
```

```
  "Thermal"
  :axiom-def
    ((phys-dom.stuff thermal entropy)
     (phys-dom.effort thermal temperature)
     (phys-dom.flow thermal entropy-flow)))


; -- Mechanisms -- ;

(define-class mechanism (?m)
  "A mechanism is a simple mereological individual"
  :def
    (simple-m-individual ?m))

(define-relation mech.ef (?m ?ef)
  "This relation relates a mechanism to the energy flows
   described by it"
  :iff-def
    (and (mechanism ?m)
         (energy-flow ?ef)
         (exists ?m2 (and (mechanism ?m2) (connects ?ef ?m ?m2)))))

(define-class one-port (?m)
  "A one port mechanism describes one energy flow"
  :iff-def
    (and
      (mechanism ?m)
      (value-cardinality ?m mech.ef 1)))

(define-class two-port (?m)
  "A two port mechanism describes two energy flows"
  :iff-def
    (and
      (mechanism ?m)
      (value-cardinality ?m mech.ef 2)))

(define-class multiport (?m)
  "A multi-port mechanism describes two or more energy flows"
  :iff-def
    (and
      (mechanism ?m)
      (>= (value-cardinality ?m mech.ef) 2)))


; -- Energy flows -- ;

(define-class energy-flow (?ef)
  "an energy flow is a topological connection between mechanisms
   an energy flow is of a certain domain"
  :def
    (and (t-connection ?ef)
         (=> (connects ?ef ?m1 ?m2)
             (and (mechanism ?m1) (mechanism ?m2)))
         (value-type ?ef ef.phys-dom physical-domain)
         (value-cardinality ?ef ef.phys-dom 1)))

(define-relation ef.from-to (?ef ?f ?t)
  "This relation defines energy flows. Topology is projected onto it
   to incorporate the necessary ontological commitments."
  :iff-def
    (and (energy-flow ?ef)
         (mechanism ?f)
         (mechanism ?t)
         (connects ?ef ?f ?t)))

(define-relation ef.phys-dom (?ef ?d)
  "Relates physical domains to energy flows."
```

```
  :axiom-def
    ((domain ef.phys-dom energy-flow)
     (range ef.phys-dom physical-domain)
     (function ef.phys-dom)))


; -- Mechanism types -- ;

(define-class source (?s)
  "A source defines the effort or flow of one energy flow.
   When the energy flow is directed toward the mechanism it is a sink."
  :def (one-port ?s)
  :axiom-def
    (exhaustive-subclass-partition
      source
      (setof effort-source flow-source)))

(define-class store (?s)
  "A store stores the stuff or action (generalized momentum)
   of one energy flow.  The energy flow must be directed toward
   the mechanism."
  :def
    (and (one-port ?s)
         (exists (?ef ?f)
           (and (energy-flow ?ef)
                (mechanism ?f)
                (ef.from-to ?ef ?f ?s))))
  :axiom-def
    (exhaustive-subclass-partition
      store
      (setof stuff-store action-store)))

(define-class dissipator (?d)
  "A dissipator dissipates the energy of one energy flow that
   must be directed toward the mechanism (otherwise it would be
   a source)."
  :def
    (and (one-port ?d)
         (exists (?ef ?f)
           (and (energy-flow ?ef)
                (mechanism ?f)
                (ef.from-to ?ef ?f ?d)))))


(define-class convertor (?c)
  "A convertor converts one kind of energy flow into another."
  :def
    (and (two-port ?c)
         (exists (?efin ?efout ?f ?t)
           (and (energy-flow ?efin)
                (energy-flow ?efout)
                (mechanism ?f)
                (mechanism ?t)
                (ef.from-to ?efin ?f ?c)
                (ef.from-to ?efout ?c ?t))))
  :axiom-def
    (exhaustive-subclass-partition
      convertor
      (setof transformer gyrator)))

(define-class distributor (?d)
  "A distributor distributes energy over energy flows
   of the dame domain."
  :def
    (and (multiport ?d)
         (forall (?ef)
           (exists (?dom)
```

```
            (forall ?m
              (=> (connects ?ef ?d ?m)
                  (ef.phys-dom ?ef ?dom))))))
  :axiom-def
    (exhaustive-subclass-partition
      distributor
      (setof effort-distributor flow-distributor)))


; -- Mechanisms -- ;

(define-class effort-source (?se)
  :def (source ?se))

(define-class flow-source (?sf)
  :def (source ?sf))

(define-class stuff-store (?c)
  :def (store ?c))

(define-class action-store (?i)
  :def (store ?i))

(define-class transformer (?tf)
  :def (convertor ?tf))

(define-class gyrator (?gy)
  :def (convertor ?gy))

(define-class effort-distributor (?je)
  :def (distributor ?je))

(define-class flow-distributor (?jf)
  :def (distributor ?jf))


; -- Processes -- ;

(define-class process (?p)
  "A process is a system of mechanisms."
  :iff-def (and (system ?p)
                (=> (in-system ?m ?p) (mechanism ?m))))
```

A.6 Physical Systems Ontology

```
; ------------------------------------------------------ ;
; PhysSys Ontology                                       ;
;                                                        ;
; Ontology for the modelling of physical systems         ;
;                                                        ;
; Author: Pim Borst                                      ;
; Date:   24-07-1997                                     ;
; ------------------------------------------------------ ;

(in-package :ol-user)

(onto-load "compview.lisp")
(onto-load "procview.lisp")

(onto-load "engineering-math/abstract-algebra.lisp")
(onto-load "engineering-math/physical-quantities.lisp")
(onto-load "engineering-math/scalar-quantities.lisp")
(onto-load "engineering-math/standard-dimensions.lisp")
```

```
(onto-load "engineering-math/unary-scalar-functions.lisp")

(define-theory PhysSys
              (frame-ontology kif-relations kif-numbers
               classical-extensional-mereology topology system-theory
               component-view process-view
               abstract-algebra physical-quantities scalar-quantities
               standard-dimensions unary-scalar-functions))

(in-theory 'PhysSys)

; -- Mapping of Components to Processes -- ;

(define-relation comp.proc (?c ?p)
  "Maps every simple component to one process description.
   Every mechanism must be in a process description of a component.
   The process descriptions of components may not overlap."
  :axiom-def
    (and
      (domain comp.proc component)
      (range comp.proc process)
      (one-to-one-relation comp.proc)
      (=> (and (component ?c)
               (simple-m-individual ?c))
          (exists (?p) (comp.proc ?c ?p)))
      (=> (mechanism ?m)
          (exists (?c ?p)
            (and (process ?p)
                 (in-system ?m ?p)
                 (comp.proc ?c ?p))))
      (=> (and (comp.proc ?c1 ?p1)
               (comp.proc ?c2 ?p2)
               (/= ?c1 ?c2))
          (disjoint ?p1 ?p2))))

; -- Mapping of connections to energy flows -- ;

(define-relation term-type.phys-dom (?t ?d)
  "Relates the terminal types to the domains of the energy flows
   going through a terminal of that type."
  :axiom-def
    (and (domain term-type.phys-dom terminal-type)
         (range term-type.phys-dom physical-domain)
         ; some examples
         (term-type.phys-dom mech-trans mech-trans)
         (term-type.phys-dom mech-rot mech-rot)
         (term-type.phys-dom electric electric)
         (term-type.phys-dom pneumatic pneumatic)
         (term-type.phys-dom hydraulic hydraulic)
         (term-type.phys-dom thermal thermal)
         (term-type.phys-dom hydro-thermal hydraulic)
         (term-type.phys-dom hydro-thermal thermal)))

(define-relation conn.ef (?c ?ef)
  "Maps every connection to one or more energy flows.
   Each related energy flow belongs to exectly one connection.
   Every connection is mapped to (an) energy flow(s).
   Each energy flow between process descriptions is mapped to a connection."
  :axiom-def
    (and
      (domain conn.ef connection)
      (range conn.ef energy-flow)
      (one-to-many-relation conn.ef)
      (=> (and (con.term ?c ?t1) (con.term ?c ?t2) (/= ?t1 ?t2)
               (comp.term ?c1 ?t1) (comp.term ?c2 ?t2)
               (comp.proc ?c1 ?p1) (comp.proc ?c2 ?p2)
               (con.type ?c ?t) (term-type.phys-dom ?t ?d))
```

```
          (exists (?ef)
            (and (conn.ef ?c ?ef) (ef.phys-dom ?ef ?d)
                 (in-boundary ?ef ?p1) (in-boundary ?ef ?p2))))
      (=> (and (comp.proc ?c1 ?p1) (comp.proc ?c2 ?p2) (/= ?c1 ?c2)
               (energy-flow ?ef) (in-boundary ?ef ?p1) (in-boundary ?ef ?p2)
               (ef.phys-dom ?ef ?d))
          (exists (?c ?t)
            (and (conn.ef ?c ?ef)
                 (con.term ?c ?t1) (con.term ?c ?t2) (/= ?t1 ?t2)
                 (comp.term ?c1 ?t1) (comp.term ?c2 ?t2)
                 (con.type ?c ?t) (term-type.phys-dom ?t ?d))))))

; -- Mapping of Processes to Mathematics -- ;

(define-relation effort.phys-dim (?e ?d)
  "Relates effort types to physical dimensions"
  :axiom-def
    (and
      (domain effort.phys-dim effort)
      (range effort.phys-dim physical-dimension)
      (function effort.phys-dim)
      ; some examples
      (effort.phys-dim force force-dimension)
      (effort.phys-dim torque (* force-dimension length-dimension))
      (effort.phys-dim pressure (* force-dimension (expt length-dimension -2)))
      (effort.phys-dim voltage
                       (* energy-dimension
                          (expt (* electrical-current-dimension time-dimension) -1)))
      (effort.phys-dim temperature thermodynamic-temperature-dimension)))

(define-relation stuff.phys-dim (?s ?d)
  "Relates stuff types to physical dimensions"
  :axiom-def
    (and
      (domain stuff.phys-dim stuff)
      (range stuff.phys-dim physical-dimension)
      (function stuff.phys-dim)
      ; some examples
      (stuff.phys-dim displacement length-dimension)
      (stuff.phys-dim angle identity-dimension)
      (stuff.phys-dim volume (expt length-dimension 3))
      (stuff.phys-dim charge (* electrical-current-dimension time-dimension))
      (stuff.phys-dim entropy
                      (* energy-dimension
                         (expt thermodynamic-temperature-dimension -1)))))

(define-relation flow.phys-dim (?f ?d)
  "Relates flow types to physical dimensions"
  :axiom-def
    (and
      (domain flow.phys-dim flow)
      (range flow.phys-dim physical-dimension)
      (function flow.phys-dim)
      ; some examples
      (flow.phys-dim velocity (* length-dimension (expt time-dimension -1)))
      (flow.phys-dim ang-velocity (* identity-dimension (expt time-dimension -1)))
      (flow.phys-dim volume-flow (* (expt length-dimension 3) (expt time-dimension -1)))
      (flow.phys-dim current electrical-current-dimension)
      (flow.phys-dim entropy-flow
                     (* energy-dimension
                        (expt (* thermodynamic-temperature-dimension
                                 time-dimension)
                              -1)))))

; -- Mapping of energy flows to physical quantities -- ;

(define-relation ef.effortq (?ef ?q)
```

```
  "Relates an effort quantity to an energy flow"
  :axiom-def
    (and (domain ef.effortq energy-flow)
         (range ef.effortq physical-quantity)
         (one-to-one-relation ef.effortq)
         (=> (and (energy-flow ?ef)
                  (ef.phys-dom ?ef ?dom)
                  (phys-dom.effort ?dom ?e)
                  (effort.phys-dim ?e ?dim))
             (exists (?q)
               (and (physical-quantity ?q)
                    (ef.effortq ?ef ?q)
                    (quantity.dimension ?q ?dim))))))

(define-relation ef.flowq (?ef ?q)
  "Relates an flow quantity to an energy flow"
  :axiom-def
    (and (domain ef.flowq energy-flow)
         (range ef.flowq physical-quantity)
         (one-to-one-relation ef.flowq)
         (=> (and (energy-flow ?ef)
                  (ef.phys-dom ?ef ?dom)
                  (phys-dom.flow ?dom ?f)
                  (flow.phys-dim ?f ?dim))
             (exists (?q)
               (and (physical-quantity ?q)
                    (ef.flowq ?ef ?q)
                    (quantity.dimension ?q ?dim))))))

; -- Index Functions -- ;

(define-relation mech.ef-in (?m ?ef)
  :iff-def (exists (?f) (ef.from-to ?ef ?f ?m)))

(define-relation mech.ef-out (?m ?ef)
  :iff-def (exists (?t) (ef.from-to ?ef ?m ?t)))

;(define-relation mech.ef (?m ?ef)
;  :iff-def (or (mech.ef-in ?m ?ef) (mech.ef-out ?m ?ef)))

(define-class index-function (?ixf)
  "An index function maps integers to physical quantities."
  :def
    (and
      (binary-relation ?ixf)
      (function ?ixf)
      (domain ?ixf integer)
      (range ?ixf physical-quantity)
      (value-type ?ixf min-index integer)
      (value-cardinality ?ixf min-index 1)
      (value-type ?ixf max-index integer)
      (value-cardinality ?ixf max-index 1)
      (=> (and (min-index ?ixf ?min)
               (max-index ?ixf ?max))
          (and (=> (and (integer ?idx) (< ?idx ?min))
                   (not (exists (?q) (holds ?ixf ?idx ?q))))
               (=> (and (integer ?idx) (>= ?idx ?min) (=< ?idx ?max))
                   (exists (?q) (holds ?ixf ?idx ?q)))
               (=> (and (integer ?idx) (> ?idx ?max))
                   (not (exists (?q) (holds ?ixf ?idx ?q))))))))

(define-relation min-index (?ixf ?min)
  "Relates the minimal index to an index function"
  :axiom-def
    (and (domain min-index index-function)
         (range min-index integer)
         (function min-index)))
```

```
(define-relation max-index (?ixf ?max)
  "Relates the maximum index to an index function"
  :axiom-def
    (and (domain max-index index-function)
         (range max-index integer)
         (function max-index)))

(define-relation mech.ixf-e-in (?m ?ixf)
  "Relates every mechanism to an indexfunction indexing the
   effort quantities of the energy flows toward the mechanism."
  :axiom-def
    (and
      (domain mech.ixf-e-in mechanism)
      (range mech.ixf-e-in index-function)
      (one-to-one-relation mech.ixf-e-in)
      (=> (mechanism ?m)
          (exists (?ixf)
            (and (mech.ixf-e-in ?m ?ixf)
                 (forall (?ef ?q)
                   (=> (and (mech.ef-in ?m ?ef)
                            (ef.effortq ?ef ?q))
                       (exists ?idx (holds ?ixf ?idx ?q)))))))))

(define-relation mech.ixf-f-in (?m ?ixf)
  "Relates every mechanism to an indexfunction indexing the
   flow quantities of the energy flows toward the mechanism."
  :axiom-def
    (and
      (domain mech.ixf-f-in mechanism)
      (range mech.ixf-f-in index-function)
      (one-to-one-relation mech.ixf-f-in)
      (=> (mechanism ?m)
          (exists (?ixf)
            (and (mech.ixf-f-in ?m ?ixf)
                 (forall (?ef ?q ?f)
                   (=> (and (mech.ef-in ?m ?ef)
                            (ef.flowq ?ef ?q))
                       (exists ?i (holds ?ixf ?i ?q)))))))))

(define-relation mech.ixf-e-out (?m ?ixf)
  "Relates every mechanism to an indexfunction indexing the
   effort quantities of the energy flows from the mechanism."
  :axiom-def
    (and
      (domain mech.ixf-e-out mechanism)
      (range mech.ixf-e-out index-function)
      (one-to-one-relation mech.ixf-e-out)
      (=> (mechanism ?m)
          (exists (?ixf)
            (and (mech.ixf-e-out ?m ?ixf)
                 (forall (?ef ?q)
                   (=> (and (mech.ef-out ?m ?ef)
                            (ef.effortq ?ef ?q))
                       (exists ?i (holds ?ixf ?i ?q)))))))))

(define-relation mech.ixf-f-out (?m ?ixf)
  "Relates every mechanism to an indexfunction indexing the
   flow quantities of the energy flows from the mechanism."
  :axiom-def
    (and
      (domain mech.ixf-f-out mechanism)
      (range mech.ixf-f-out index-function)
      (one-to-one-relation mech.ixf-f-out)
      (=> (mechanism ?m)
          (exists (?ixf)
            (and (mech.ixf-f-out ?m ?ixf)
```

```
                (forall (?ef ?q)
                  (=> (and (mech.ef-out ?m ?ef)
                           (ef.flowq ?ef ?q))
                      (exists ?i (holds ?ixf ?i ?q)))))))))

; -- The mapping of Mechanisms to Relations: The Real Stuff -- ;

(define-class ascending (?r)
  :iff-def
    (=> (and (holds ?r ?x1 ?y1)
             (holds ?r ?x2 ?y2)
             (> ?x2 ?x1))
        (> ?y2 ?y1)))

(define-relation mech.quant (?m ?q)
  :axiom-def
    (and
      (domain mech.quant mechanism)
      (range mech.quant physical-quantity)
      (one-to-one-relation mech.quant)

      (=> (and (effort-source ?m)
               (mech.ef ?m ?ef)
               (ef.effortq ?ef ?e)
               (ef.flowq ?ef ?f))
          (exists (?q)
            (and (mech.quant ?m ?q)
                 (= ?e ?q))))                          ; e = q

      (=> (and (flow-source ?m)
               (mech.ef ?m ?ef)
               (ef.effortq ?ef ?e)
               (ef.flowq ?ef ?f))
          (exists (?q)
            (and (mech.quant ?m ?q)
                 (= ?f ?q))))                          ; f = q

      (=> (and (transformer ?m)
               (mech.ef-in ?m ?in)
               (mech.ef-out ?m ?out)
               (ef.effortq ?in ?ein)
               (ef.flowq ?in ?fin)
               (ef.effortq ?out ?eout)
               (ef.flowq ?out ?fout))
          (exists (?r ?q)
            (and (mech.quant ?m ?q)
                 (= ?ein (* ?q ?eout))                 ; e1 = q e2
                 (= ?fout (* ?q ?fin)))))              ; f2 = q f1

      (=> (and (gyrator ?m)
               (mech.ef-in ?m ?in)
               (mech.ef-out ?m ?out)
               (ef.effortq ?in ?ein)
               (ef.flowq ?in ?fin)
               (ef.effortq ?out ?eout)
               (ef.flowq ?out ?fout))
          (exists (?r ?q)
            (and (mech.quant ?m ?q)
                 (= ?ein (* ?q ?fout))                 ; e1 = q f2
                 (= ?eout (* ?q ?fin)))))              ; e2 = q f1

        (=> (and (effort-distributor ?m)
                 (mech.ixf-e-in ?m ?ein)
                 (min-index ?ein ?minin)
                 (max-index ?ein ?maxin)
                 (mech.ixf-e-out ?m ?eout)
                 (min-index ?eout ?minout)
```

```
                (max-index ?eout ?maxout))
           (and (= (summation ?ein ?minin ?maxin)
                   (summation ?eout ?minout ?maxout))
                (exists (?q)
                  (and (mech.quant ?m ?q)
                       (=> (and (mech.ef ?m ?ef)
                                (ef.flowq ?ef ?f))
                           (= ?f ?q))))))

       (=> (and (flow-distributor ?m)
                (mech.ixf-f-in ?m ?fin)
                (min-index ?fin ?minin)
                (max-index ?fin ?maxin)
                (mech.ixf-f-out ?m ?fout)
                (min-index ?fout ?minout)
                (max-index ?fout ?maxout))
           (and (= (summation ?fin ?minin ?maxin)
                   (summation ?fout ?minout ?maxout))
                (exists (?q)
                  (and (mech.quant ?m ?q)
                       (=> (and (mech.ef ?m ?ef)
                                (ef.effortq ?ef ?e))
                           (= ?e ?q))))) ))

(define-relation mech.mathrel (?m ?r)
  :axiom-def
    (and
      (domain mech.mathrel mechanism)
      (range mech.mathrel relation)
      (one-to-one-relation mech.mathrel)

      (=> (and (dissipator ?m)
               (mech.ef-in ?m ?ef)
               (ef.effortq ?ef ?e)
               (ef.flowq ?ef ?f))
          (exists (?r)
            (and (mech.mathrel ?m ?r)
                 (holds ?r ?e ?f)                              ; r(e,f)
                 (=> (holds ?r ?x ?y)
                     (and (>= (* ?x ?y) (* 0 energy-dimension))
                          (<=> (zero-quantity ?x) (zero-quantity ?y))))))))

      (=> (and (stuff-store ?m)
               (mech.ef-in ?m ?ef)
               (ef.effortq ?ef ?e)
               (ef.flowq ?ef ?f))
          (exists (?r ?q)
            (and (mech.mathrel ?m ?r)
                 (mech.quant ?m ?q)
                 (holds ?r ?q ?e)                              ; r(q,e)
                 (= (D/DT ?q) ?f)                              ; dq/dt = f
                 (ascending ?r))))

      (=> (and (action-store ?m)
               (mech.ef-in ?m ?ef)
               (ef.effortq ?ef ?e)
               (ef.flowq ?ef ?f))
          (exists (?r ?q)
            (and (mech.mathrel ?m ?r)
                 (mech.quant ?m ?q)
                 (holds ?r ?q ?f)                              ; r(q,f)
                 (= (D/DT ?q) ?e)                              ; dq/dt = e
                 (ascending ?r))) ))
```

Appendix B

The Thermodynamic Models in the OLMECO Library

This appendix describes the thermodynamic model fragments developed for the OLMECO *design library (see Section 4). Its function is to support engineers in the construction of large simulation models from library model fragments. This appendix is a revised version of the* OLMECO *deliverable (Top, Borst, and Akkermans 1995) and describes work that is part of the ECN contribution to the* OLMECO *project. The focus is on heat generation, exchange and transport through thermal fluids (liquid or gas), where both convection and conduction of thermal energy can play a role. The usability and reusability of the model library is demonstrated by a large-scale evaluation experiment, modelling and simulating the heating installation of a hospital in The Netherlands. It is concluded that the* OLMECO *library approach leads to higher quality models, with less effort.*

The organization of this appendix is as follows, Section B.1 gives an overview of the thermodynamic models in the library and the modelling assumptions that apply to them. In Section B.2 the general physical process model for convective heat transport is described. Many process models in the library are based on this general model. Section B.3–B.5 the thermodynamic models themselves are described. Section B.6 gives a more detailed description of the modelling and simulation experiment compared to the description in Chapter 4. Section B.6.6 gives detailed conclusions about the experiment.

B.1 Introduction

This appendix describes the generic model components in the thermodynamic domain for the OLMECO library (see also Chapter 4). Figure B.1 gives an overview of the thermal components, for which reusable library models for simulation and design will be presented in this

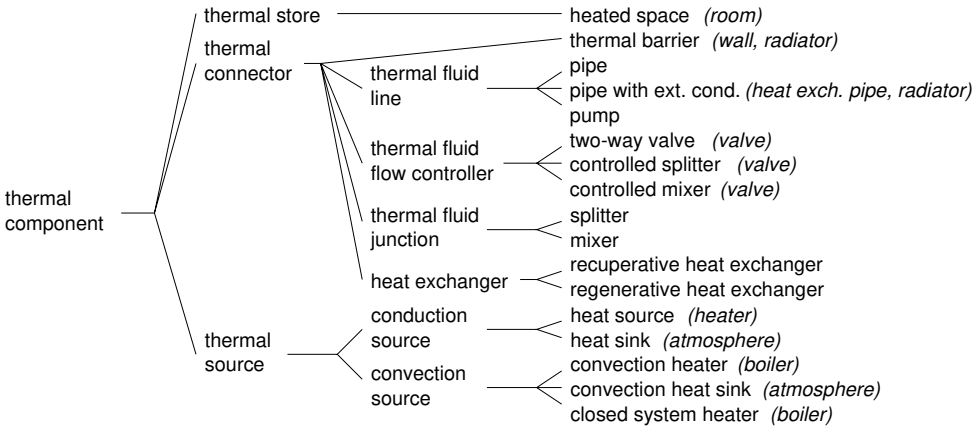


Figure B.1: Taxonomy of thermal components.

report. The usability and reusability of the model library is demonstrated by a large-scale evaluation experiment, modelling and simulating the heating installation of the Schieland hospital in Schiedam, The Netherlands (see Section B.6 and also Chapter 4).

Our focus is on heat transport through thermal fluids (liquid or gas), where both convection and conduction of thermal energy can play a role. At the physical process level, both pseudo-bondgraphs and entropy-based bondgraphs will be considered. If the assumptions underlying the pseudo-bondgraph approach are properly acknowledged, this representation has the advantage that it is closer to standard engineering practice, in particular in the building services industry (den Ouden, Top, and Akkermans 1994). The basic consequences implied by a pseudo-bondgraph representation for the underlying system dynamics are:

- Rather than entropy flow, the energy (enthalpy) flow is taken as the conjugate of temperature. Note that this may introduce redundancy in the model since this energy term also includes non-thermal contributions (*e.g.*, convected kinetic energy).
- Since the entropy is not modelled explicitly, dissipation is assumed to be sufficiently small such that it can be neglected.
- Since the half arrows do not represent power, it is not possible to connect them directly to real power bonds. Hence, care must be taken when interaction with other domains is involved.
- Often *ad hoc* constructions in terms of signals (active bonds) will have to be used, in particular when linking to other domains.
- There is no way to find or check constitutive relations using energy functions: there is no reciprocity condition for storage elements (Karnopp 1979).
- In general, the physical rationale behind the model is more implicit.

If these consequences cannot be accepted, the entropy flows have to be modelled explicitly; they are the conjugated variables of temperature. The explicit entropy approach is generally preferable from a theoretical point of view and provides more insight than the pseudo-bondgraph approach.

In Section B.3–B.5 a number of library components for the heating systems domain will be described. Each chapter covers a number of basic system components, physical process descriptions in terms of bond graphs, and constitutive mathematical relations for design and simulation (Top and Akkermans 1994; Top, Breunese, van Dijk, Broenink, and Akkermans 1994; Akkermans, Borst, Pos, and Top 1995; Top, Breunese, Broenink, and Akkermans 1995).

Note that at the component level only *plugs* are specified, which can be filled with any number of energybonds or signals. This allows one to describe the components independently from the process level; hence, both pseudo- and entropy-bondgraphs can be used at the process level. Further we note that decompositions may be applied recursively. For example, when fluid flow through a pipe has to be modelled by multiple segments this can be done by repeated decomposition.

At the level of physical processes some general assumptions will be made:

- Since our approach is based on *reticulation* (network modelling), we assume that fluids mix instantaneously within each lump.
- When considering fluid flow we assume a global Eulerian (fixed volume) frame of reference, with only local Lagrangian frames (e.g. a fixed amount of fluid passing through a pipe).
- Except when compressibility is explicitly mentioned, fluids are assumed to be incompressible.
- Substances (fluids) are assumed to consist of one chemical component only.

In the following chapters we use the following convention to indicate which domain is associated with a bond:

symbol	**domain**	**effort**	**flow**
th	thermodynamic	temperature (K)	entropy flow (W/K)
tp	thermal, pseudo	temperature (K)	energy (enthalpy) flow (W)
hy	hydraulic	pressure (N/m^2)	volume flow (m^3/sec)
ma	material	(total) mat. potential (J)	material flow ($moles/sec$)

In Section B.3–B.5, models belonging to one of the three-level `ontology' approach (Chapter 4) is are presented: (i) technological level: system components and their decompositions; (ii) physical level: process descriptions of physical behaviour; and (iii) mathematical level: associated mathematical equations. The links between these library entries are indicated in their descriptions. Subsequently, the use and reuse of the library is demonstrated by presenting a large-scale evaluation experiment, whereby the thermal as well as hydraulic behaviour

of the heating installation of a hospital in The Netherlands is modelled and simulated. But first, Section B.2 presents the background of the models for convective transport of thermal energy used in the library. See also Section 4.2 for design considerations about the models in the library.

B.2 Convective transport of thermal energy

Convection of internal energy occurs when matter flows from one place in a system to another. For the representation of convection in terms of temperature and energy as pseudo-conjugated variables, we build on the proposal made by Karnopp (Karnopp 1978). Here, a two-port R-element determines the amount of convected energy (enthalpy), modulated by the temperature of the upstream convection element and the fluid flow velocity, as shown in Fig. B.2a. However, from the definition of the element and its fixed causality it appears that actually a source element is used to `inject' an amount of heat. Therefore, we use the bond graph description shown in Fig. B.2b. Also for the flow splitter and mixer alternative models are proposed. The advantage of our proposal is that part of the underlying model can now be obtained from the junction element, whereas in (Karnopp 1978) these relations have to be given explicitly for the three-port R-element.

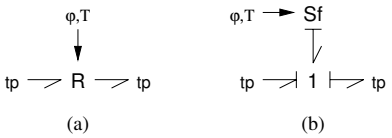


Figure B.2: The convection process as suggested by Karnopp (Karnopp 1978): *(a)* original representation; *(b)* our proposal.

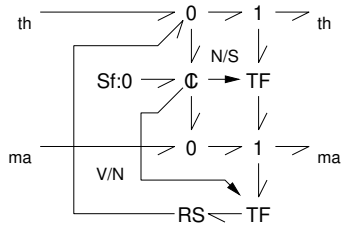


Figure B.3: The convection process in full bond graph form (Breedveld 1984), simplified for the case of a single component material flow.

The theoretical basis for description of convection in terms of thermodynamic bondgraphs has been developed in (Breedveld 1984). Here we restrict ourselves to flows with a single chemical component and only convection of entropy is considered. This means that the general convection bond graph proposed in (Breedveld 1984) is simplified to the one shown in Fig. B.3. The two 1-junctions and the modulated TF together represent the (discretized)

Gibbs-Duhem relation $\Delta\mu = v\Delta p - s\Delta T$, where $\Delta\mu$ is the difference in material potential between this convective element and the next, v and s are specific volume and entropy, p is pressure and T is temperature.

B.3 Components and Decompositions

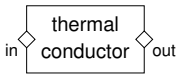
B.3.1 Thermal Conductor

Function: To provide thermal conduction or insulation between two systems.

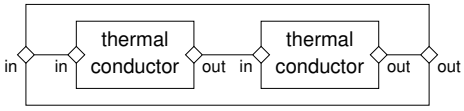
Examples: wall, wire, tube wall

Process Models: heat conduction (B.4.1), heat radiation (B.4.2), free convection (B.4.3)

Component Model:



Possible Decompositions: thermal conductor (B.3.1)



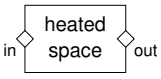
B.3.2 Heated Space

Function: To store heat, to maintain a given temperature.

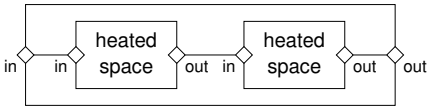
Examples: room, thick wall, dewar, tube wall

Process Models: heat storage (B.4.4)

Component Model:



Possible Decompositions: heated space (B.3.2)

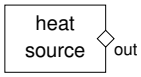


B.3.3 Heat Source

Function: To provide thermal energy to a system without being affected by it.

Process Models: temperature source (B.4.5), heat/entropy source (B.4.7)

Component Model:

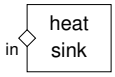


B.3.4 Heat Sink

Function: To provide or consume thermal energy to a system without being affected by it.

Process Models: temperature sink (B.4.6), heat/entropy sink (B.4.8)

Component Model:

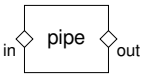


B.3.5 Pipe

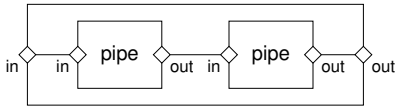
Function: To provide transport of thermal energy by means of a fluid from one side of the pipe to the other.

Process Models: convection (B.4.9)

Component Model:



Possible Decomposition: pipe (B.3.5)

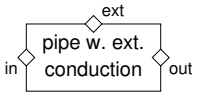


B.3.6 Pipe with External Conduction

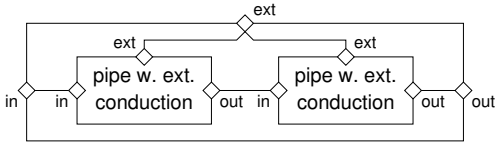
Function: To provide transport of thermal energy through a fluid from one side of the pipe to the other and from the fluid to the system enclosing the pipe.

Process Models: convection with external conduction (B.4.10)

Component Model:



Possible Decomposition: pipe with external conduction (B.3.6)

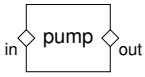


B.3.7 Pump

Function: To provide transport of thermal energy by delivering a hydraulic pressure for thermal fluid flow.

Process Models: convection through hydraulic source (B.4.11)

Component Model:

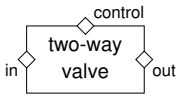


B.3.8 Two-Way Valve

Function: To control the amount of fluid transporting thermal energy.

Process Models: controlled convection (B.4.12)

Component Model:

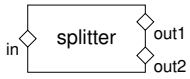


B.3.9 Splitter

Function: To split the transport of thermal energy and continue in two separate pipes.

Process Models: convection splitting (B.4.13)

Component Model:

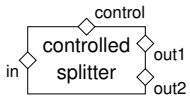


B.3.10 Controlled Splitter

Function: To control the splitting of a fluid that proceeds in two separate pipes.

Process Models: controlled convection splitting (B.4.14)

Component Model:

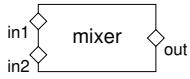


B.3.11 Mixer

Function: To merge two convective paths, proceeding as one.

Process Models: convection mixing (B.4.15)

Component Model:

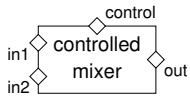


B.3.12 Controlled Mixer

Function: To control the mixing of two fluids that flow in a single pipe.

Process Models: controlled convection mixing (B.4.16)

Component Model:

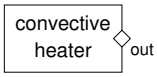


B.3.13 Convective Heater

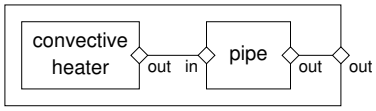
Function: To provide thermal energy to a system by convection of-, or conduction through fluid, without being affected by it.

Process Models: convective temperature source (B.4.17)

Component Model:



Possible Decompositions: convective heater (B.3.13), pipe (B.3.5)

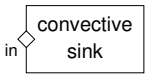


B.3.14 Convective Heat Sink

Function: To provide or consume thermal energy by convection of, or conduction through, fluid, without being affected by it.

Process Models: convective temperature sink (B.4.18)

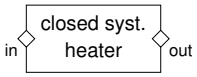
Component Model:



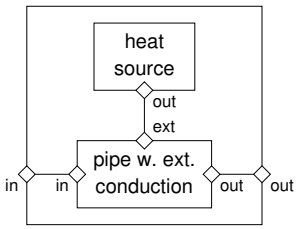
B.3.15 Closed System Heater

Function: To provide thermal energy to a closed system by convection of, or conduction through, fluid.

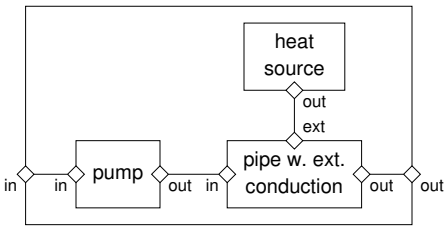
Component Model:



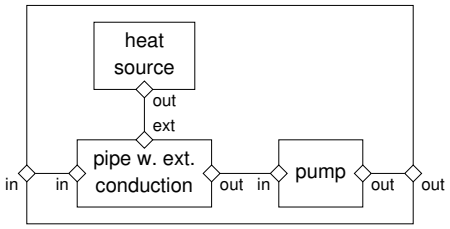
Possible Decompositions: (a) heat source (B.3.3), pipe with external conduction (B.3.6), (b) pump (B.3.7), heat source (B.3.3), pipe with external conduction (B.3.6), (c) heat source (B.3.3), pipe with external conduction (B.3.6), pump (B.3.7)



(a)



(b)

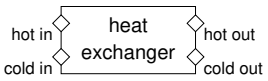


(c)

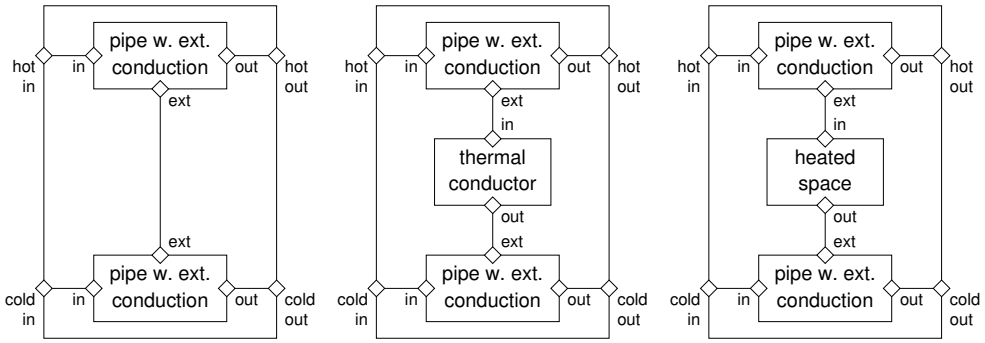
B.3.16 Basic Recuperative Heat Exchanger

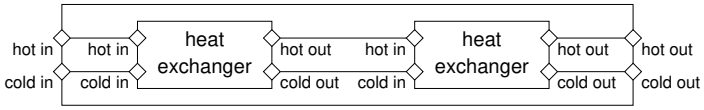
Function: To transfer thermal energy from one flow to another *via* an intermediate path. Sometimes heat exchangers also serve as pressure vessels. Flows are parallel or opposite. Note that improved heat transfer implies increased pressure drop, thus requiring more mechanical energy.

Component Model:

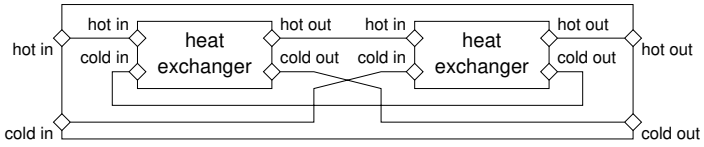


Possible Decompositions: *basic decompositions (a–c):* pipe with external conduction (B.3.6), thermal conductor (B.3.1), heated space (B.3.2); *parallel flow (d):* heat exchanger (B.3.16); *counter flow (e):* heat exchanger (B.3.16)





(d)



(e)

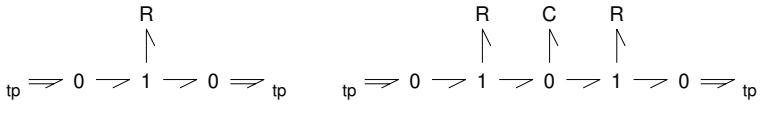
B.4 Physical Processes

B.4.1 Heat Conduction

Component Models: thermal conductor (B.3.1)

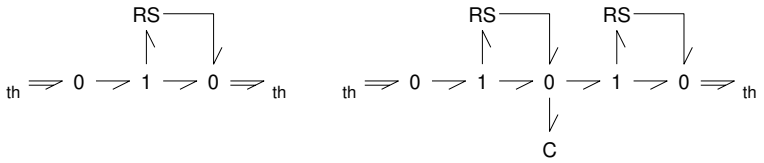
Mathematical Relations: heat conduction resistor (B.5.1), heat capacitor (B.5.4)

Pseudo Bondgraph:



(a) (b) (c) (d)

Entropy Bondgraph:



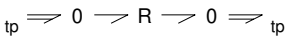
(a) (b) (c) (c)

B.4.2 Heat Radiation

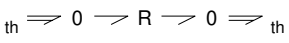
Component Models: thermal conductor (B.3.1)

Mathematical Relations: heat radiation resistor (B.5.2)

Pseudo Bondgraph:



Entropy Bondgraph:

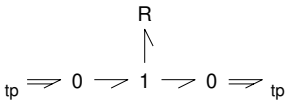


B.4.3 Free Convection

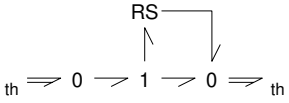
Component Models: thermal conductor (B.3.1)

Mathematical Relations: free convection resistor (B.5.3)

Pseudo Bondgraph:



Entropy Bondgraph:

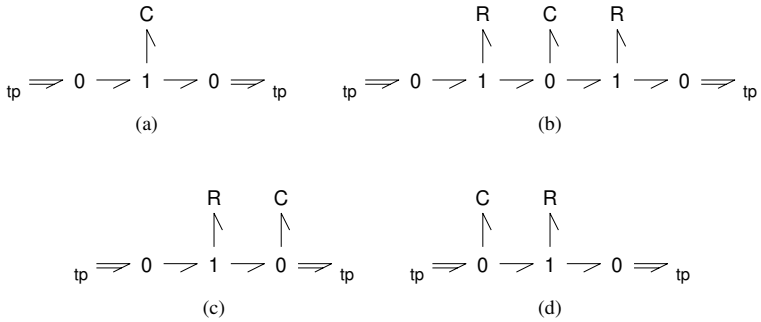


B.4.4 Heat Storage

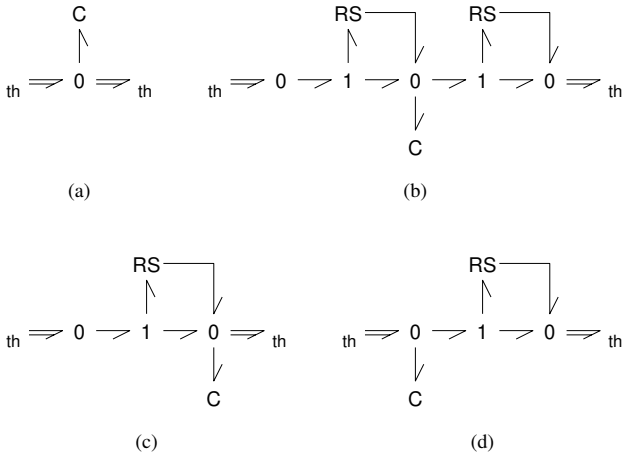
Component Models: heated space (B.3.2)

Mathematical Relations: heat capacitor (B.5.4), heat conduction resistor (B.5.1)

Pseudo Bondgraph:



Entropy Bondgraph:

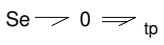


B.4.5 Temperature Source

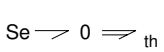
Component Models: heat source (B.3.3)

Mathematical Relations: temperature source (B.5.5)

Pseudo Bondgraph:



Entropy Bondgraph:

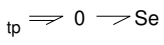


B.4.6 Temperature Sink

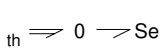
Component Models: heat sink (B.3.4)

Mathematical Relations: temperature sink (B.5.6)

Pseudo Bondgraph:



Entropy Bondgraph:

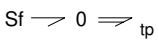


B.4.7 Heat/Entropy Source

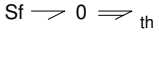
Component Models: heat source (B.3.3)

Mathematical Relations: heat/entropy source (B.5.7)

Pseudo Bondgraph:



Entropy Bondgraph:

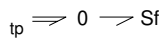


B.4.8 Heat/Entropy Sink

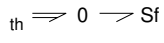
Component Models: heat sink (B.3.4)

Mathematical Relations: heat/entropy source (B.5.8)

Pseudo Bondgraph:



Entropy Bondgraph:

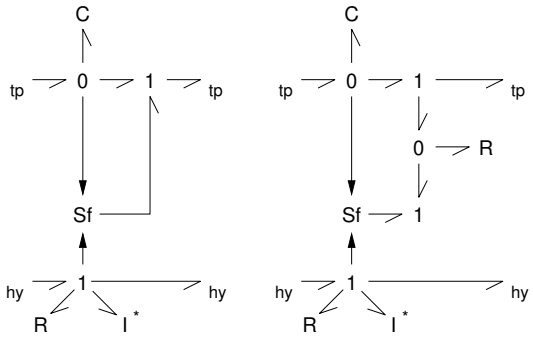


B.4.9 Convection

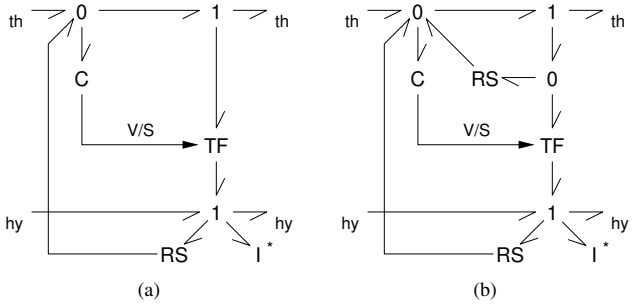
Component Models: pipe (B.3.5)

Mathematical Relations: hydraulic resistor (B.5.12), hydraulic inertance (B.5.9), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic inertance (I*)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic inertance (I*)

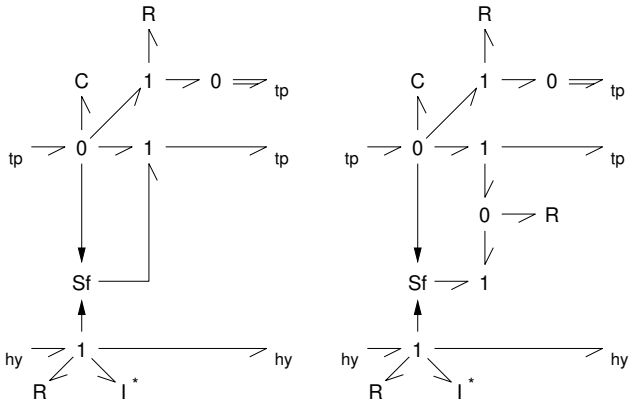


B.4.10 Convection with External Conduction

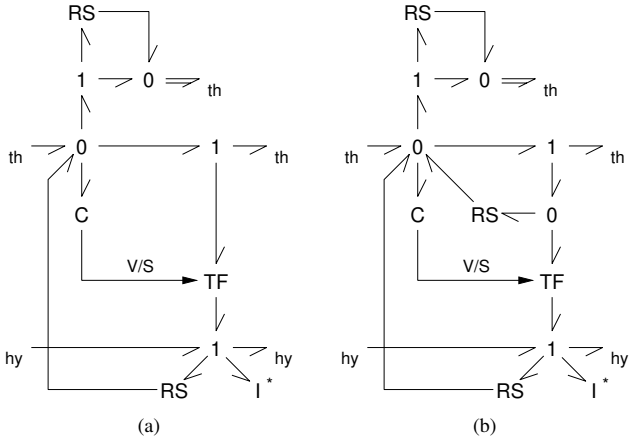
Component Models: pipe with external conduction (B.3.6)

Mathematical Relations: hydraulic resistor (B.5.12), hydraulic inertance (B.5.9), heat capacitor (B.5.4), heat conduction resistor (B.5.1), radiator resistor (B.5.14), external conduction resistor (B.5.18), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic inertance (I*), with/without external thermal resistance (R*)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic inertance (I*), with/without external thermal resistance (RS*)

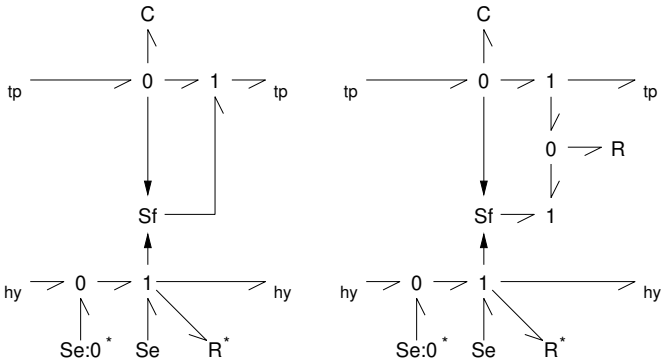


B.4.11 Convection through Hydraulic Source

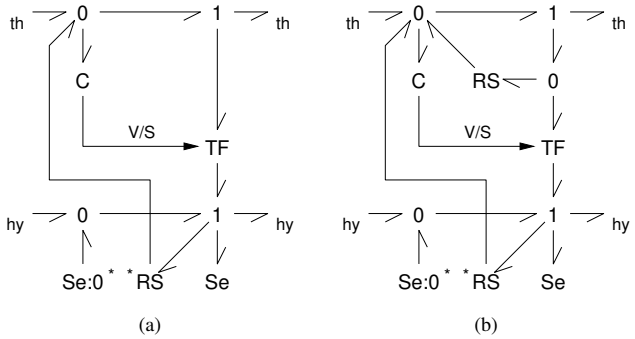
Component Models: pump (B.3.7)

Mathematical Relations: hydraulic resistor (B.5.12), pressure source (B.5.15), pressure reference source (B.5.17), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic resistance (R^*), with/without reference pressure source ($Se{:}0^*$)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b), with/without hydraulic resistance (RS^*), with/without reference pressure source ($Se{:}0^*$)

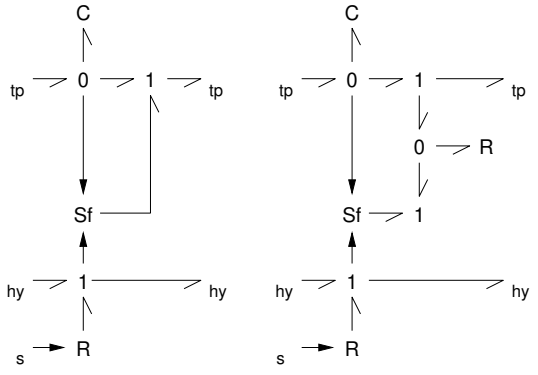


B.4.12 Controlled Convection

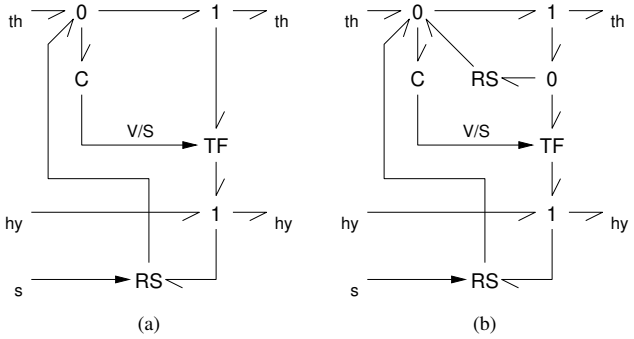
Component Models: two-way valve (B.3.8)

Mathematical Relations: controlled hydraulic resistor (B.5.13), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b)

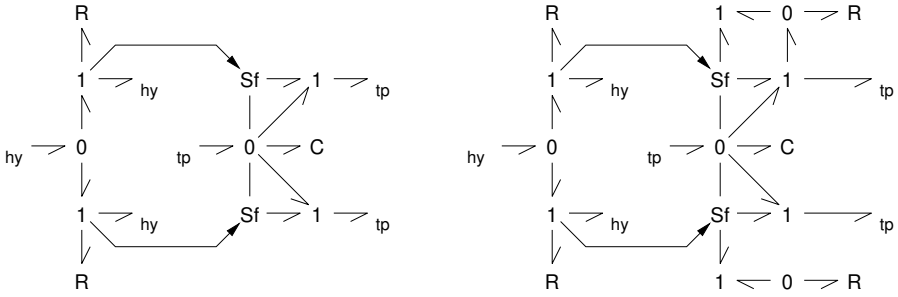


B.4.13 Convection Splitting

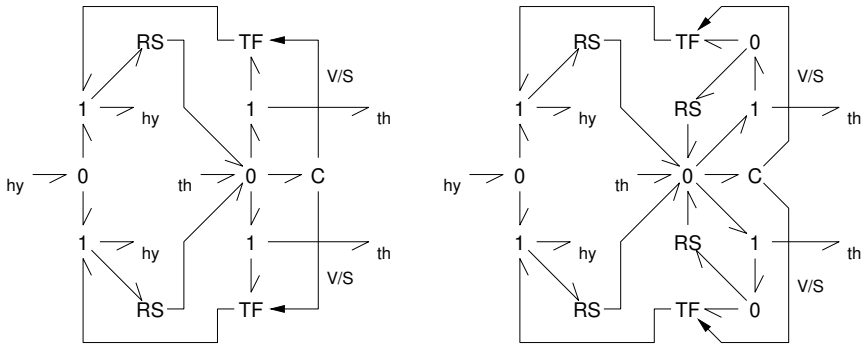
Component Models: splitter (B.3.9)

Mathematical Relations: hydraulic resistor (B.5.12), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b)

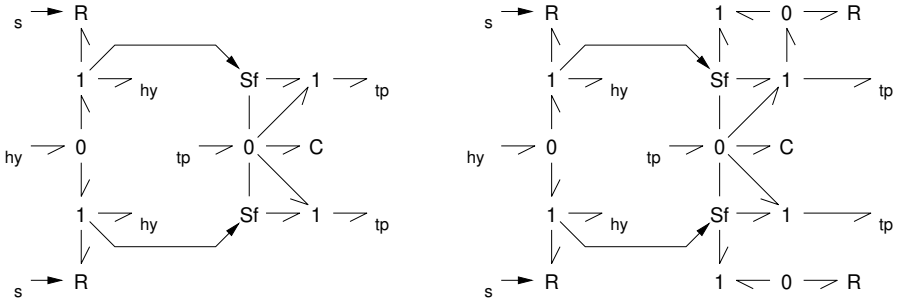


B.4.14 Controlled Convection Splitting

Component Models: controlled splitter (B.3.10)

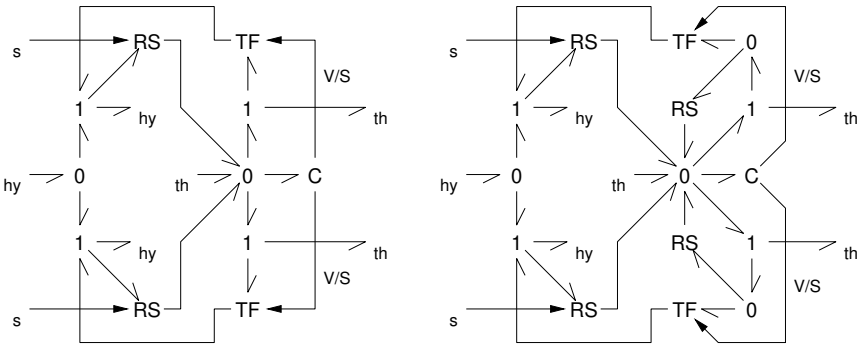
Mathematical Relations: controlled hydraulic resistor (B.5.13), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



(a) (b)

Entropy Bondgraph: without internal conduction (a), with internal conduction (b)



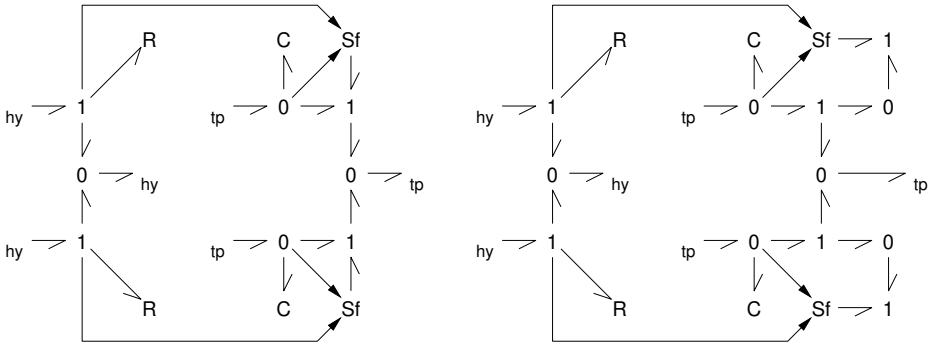
(a) (b)

B.4.15 Convection Mixing

Component Models: mixer (B.3.11)

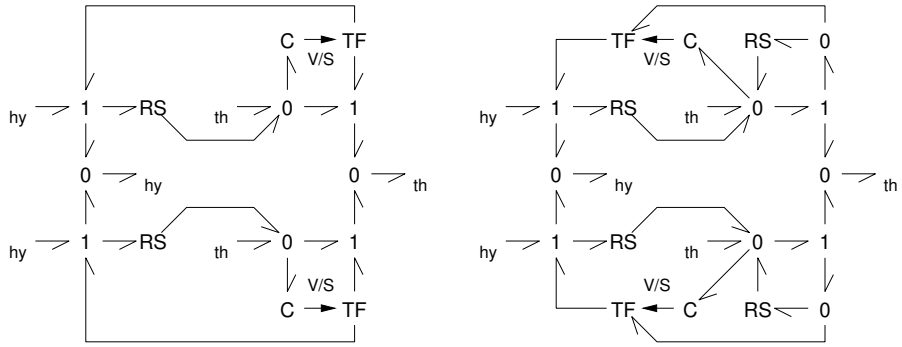
Mathematical Relations: hydraulic resistor (B.5.12), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



(a) (b)

Entropy Bondgraph: without internal conduction (a), with internal conduction (b)



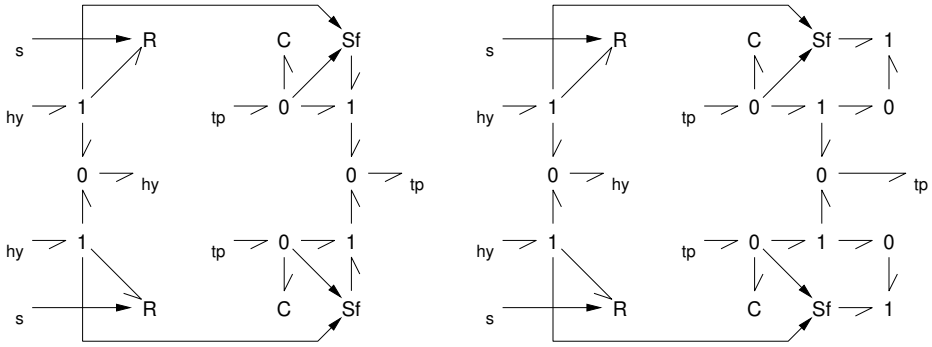
(a) (b)

B.4.16 Controlled Convection Mixing

Component Models: controlled mixer (B.3.12)

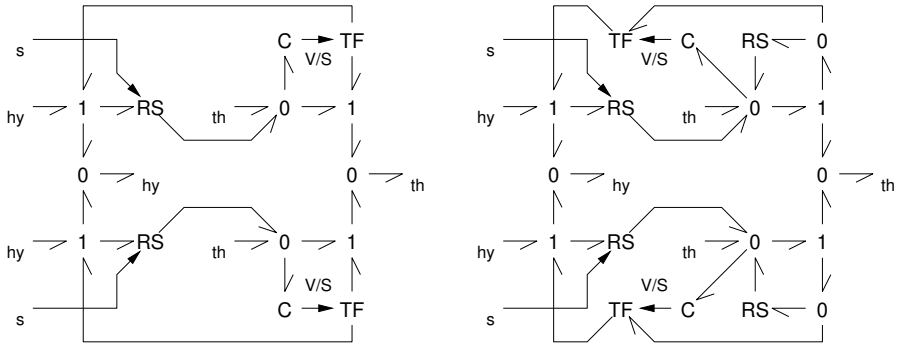
Mathematical Relations: controlled hydraulic resistor (B.5.13), heat capacitor (B.5.4), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



(a) (b)

Entropy Bondgraph: without internal conduction (a), with internal conduction (b)



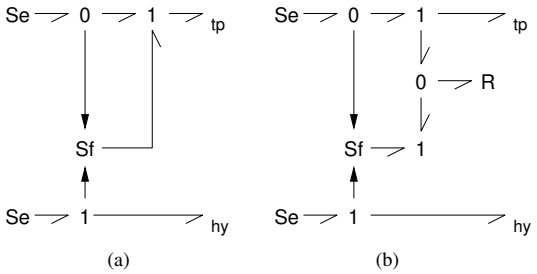
(a) (b)

B.4.17 Convective Temperature Source

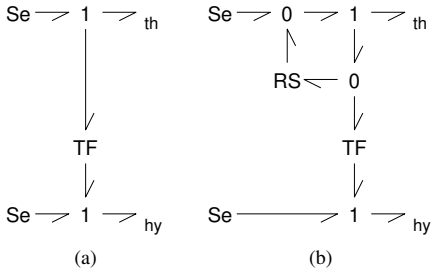
Component Models: convective heater (B.3.13)

Mathematical Relations: pressure source (B.5.15), temperature source (B.5.5), heat conduction resistor (B.5.1), convection source (B.5.10), convection transformer (B.5.11)

Pseudo Bondgraph: without internal conduction (a), with internal conduction (b)



Entropy Bondgraph: without internal conduction (a), with internal conduction (b)

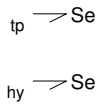


B.4.18 Convective Temperature Sink

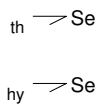
Component Models: convective heat sink (B.3.14)

Mathematical Relations: pressure source (B.5.15), temperature sink (B.5.6), pressure sink (B.5.16)

Pseudo Bondgraph:



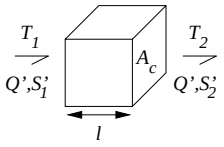
Entropy Bondgraph:



B.5 Mathematical Relations

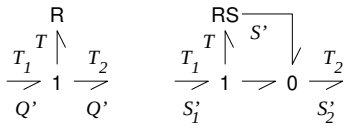
B.5.1 Heat Conduction Resistor

geometry:



processes: heat conduction (B.4.1 a–d), heat storage (B.4.4 b–d), convection (B.4.9 b), controlled convection (B.4.12 b), convection splitting/mixing (B.4.13/B.4.15 b), controlled convection splitting/mixing (B.4.14/B.4.16 b), convective temperature source (B.4.17 b), convection through a hydraulic source (B.4.11 b),

elements:



relations:

$$T = T_1 - T_2 \tag{B.1}$$

$$Q' = T/R \tag{B.2}$$

$$S_1' = \frac{Q'}{T + T_2} = \frac{T}{(T + T_2)R} \tag{B.3}$$

$$S' = \frac{TS_1'}{T_2} = \frac{T^2}{T_2(T + T_2)R} \tag{B.4}$$

$$S_2' = S_1' + S' \tag{B.5}$$

$$R = \frac{l}{\lambda A_c} \tag{B.6}$$

variables:

$Q'[W]$:	heat flow
$S_1'[WK^{-1}]$:	entropy in flow
$S_2'[WK^{-1}]$:	entropy out flow
$S'[WK^{-1}]$:	entropy flow produced by conduction process
$T_1[K]$:	temperature left side
$T_2[K]$:	temperature right side
$T[K]$:	temperature difference left-right
$R[KW^{-1}]$:	thermal resistance
$l[m]$:	thickness or length
$A_c[m^2]$:	heat conduction area
$\lambda[WK^{-1}m^{-1}]$:	heat conduction coefficient

parameters: Tables of heat conduction coefficients for various materials can be found in (Verein Deutscher Ingenieure 1977):

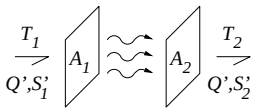
water:	chapter Db, tables 1 and 7
air:	chapter Db, tables 14 and 20
construction materials:	chapter Ea

notes: The heat conduction coefficient λ is assumed constant although it depends on pressure, temperature and humidity. For pipes with inner radius r_i and outer radius r_o, R becomes

$$R = \frac{\ln(r_o/r_i)}{2\pi\lambda l}$$

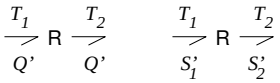
B.5.2 Heat Radiation Resistor

general geometry:



processes: heat radiation (B.4.2)

elements:



relations:

$$Q' = g(T_1^4 - T_2^4) \quad \text{(B.7)}$$
$$S_1' = g(T_1^3 - T_2^4/T_1) \quad \text{(B.8)}$$
$$S_2' = g(T_1^4/T_2 - T_2^3) \quad \text{(B.9)}$$
$$S' = S_2' - S_1' \quad \text{(B.10)}$$
$$g = \phi_{12} A_1 \epsilon_1 \epsilon_2 C_s \quad \text{(B.11)}$$

variables:

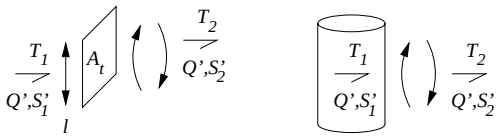
$Q'[W]$:	heat flow from surface 1 to 2
$S_1'[WK^{-1}]$:	entropy flow from surface 1
$S_2'[WK^{-1}]$:	entropy flow to surface 2
$S'[WK^{-1}]$:	entropy flow produced by radiation process
$T_1[K]$:	temperature surface 1
$T_2[K]$:	temperature surface 2
$A_1[m^2]$:	area radiating surface 1
$A_2[m^2]$:	area radiating surface 2
$g[-]$:	situation dependent parameter
$\phi_{12}[-]$:	absorption coefficient
$\epsilon_1[-]$:	emission coefficient (luminosity) surface 1

$\epsilon_2[-]$:	emission coefficient (luminosity) surface 2
$C_s[Wm^{-2}K^{-4}]$:	Stefan-Boltzmann constant

parameters: The Stefan-Boltzmann constant is approximately $5.6697 \cdot 10^{-8}$. The values for ϵ_1 and ϵ_2 for construction materials can be found in (Verein Deutscher Ingenieure 1977) chapter Ka table 2. For the determination of ϕ_{12} for various geometries of specific situations the reader is referred to section Kb in (Verein Deutscher Ingenieure 1977).

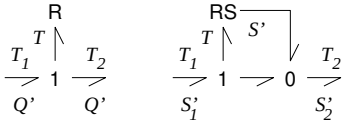
B.5.3 Free Convection Resistor

geometry:



processes: free convection (B.4.3)

elements:



relations:

$$T = T_1 - T_2 \tag{B.12}$$

$$Gr = \begin{cases} \frac{gl^3\beta T}{\nu^2} = \frac{g\rho^2 l^3 \beta T}{\eta^2} & \text{for vert. walls} \\ \frac{gd_o l'^2 \beta T}{\nu^2} = \frac{g\rho^2 d_o l'^2 \beta T}{\eta^2} & \text{for vert. tubes} \end{cases} \tag{B.13}$$

$$l' = \frac{\pi d_o}{2} \quad \text{for vert. tubes} \tag{B.14}$$

$$Nu = \begin{cases} 0.517(GrPr)^{1/4} & 10^4 < Gr, Pr < 10^8 \\ 0.10(GrPr)^{1/3} & Gr, Pr > 10^9 \end{cases} \tag{B.15}$$

$$Pr = \frac{\nu}{a} = \frac{\eta c_p}{\lambda} \tag{B.16}$$

$$\alpha = \frac{\lambda Nu}{l} \tag{B.17}$$

$$A_t = l\pi d_o \quad \text{for vert. tubes} \tag{B.18}$$

$$Q' = A_t \alpha T \tag{B.19}$$

$$S_1' = \frac{Q'}{T + T_2} = \frac{A_t \alpha T}{T + T_2} \tag{B.20}$$

$$S' = \frac{TS_1'}{T_2} = \frac{A_t \alpha T^2}{T_2(T + T_2)} \tag{B.21}$$

$$S_2' = S_1' + S \tag{B.22}$$

variables:

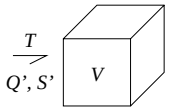
$T[K]$:	temperature difference wall-air (tube-air)
$T_1[K]$:	temperature wall (tube)
$T_2[K]$:	distant air temperature
$Gr[-]$:	Grashof number
$g[ms^{-2}]$:	gravitational acceleration
$l[m]$:	height of wall (length of tube)
$d_o[m]$:	outer diameter of the tube
$\beta[K^{-1}]$:	heat expansion coefficient air
$\nu[m^2s^{-1}]$:	kinematic viscosity air
$\eta[kgm^{-1}s^{-1}]$:	dynamic viscosity air
$\rho[kgm^{-3}]$:	specific mass air
$Nu[-]$:	Nußel number
$Pr[-]$:	Prandtl number
$a[m^2s^{-1}]$	temperature conduction coefficient air
$c_p[Jkg^{-1}K^{-1}]$:	specific heat capacity air at constant pressure
$\lambda[WK^{-1}m^{-1}]$:	heat conduction coefficient air
$\alpha[Wm^{-2}K^{-1}]$:	heat transfer coefficient
$A_t[m^2]$:	heat transfer area of wall or tube
$Q'[W]$:	heat flow wall to air
$S'_1[WK^{-1}]$:	entropy out flow wall (tube)
$S'_2[WK^{-1}]$:	entropy in flow air
$S'[WK^{-1}]$:	entropy flow produced by convection process

parameters: For ideal gases, $\beta = 1/T_2$. The values for ν, η, ρ, β, λ and c_p depend on the temperature and pressure of the air. For simulation, they are considered to be constant and can be obtained from the tables in chapter Db in (Verein Deutscher Ingenieure 1977) (take average temperature and pressure). The gravitational acceleration is normally taken as $g = 9.81ms^{-2}$.

notes: *The equations for both walls and tubes assume the movement of the air is solely caused by the temperature difference* T.

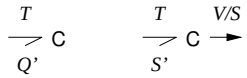
B.5.4 Heat Capacitor

geometry:



processes: heat storage (B.4.4), heat conduction (B.4.1 b–d), convection (B.4.9), controlled convection B.4.12), convection splitting/mixing (B.4.13/B.4.15), controlled convection splitting/mixing (B.4.14/B.4.16)

elements:



relations:

$$T = \begin{cases} Q/C, & Q = \int_0^t Q'\,dt + Q_0, \quad Q_0 = T_0/C \\ T_0 e^{S/C}, & S = \int_0^t S'\,dt + S_0 \quad S_0 = C \end{cases} \begin{array}{l} \text{pseudo bond graph} \\ \text{entropy bond graph} \end{array} \tag{B.23}$$

$$C = mc_v \tag{B.24}$$

$$m = \rho V \tag{B.25}$$

variables:

$t[s]$:	time parameter
$Q[J]$:	thermal energy stored
$Q_0[J]$:	thermal energy stored at $t = 0$
$Q'[W]$:	heat flow to object
$S'[WK^{-1}]$:	entropy flow to object
$S[JK^{-1}]$:	entropy stored
$T[K]$:	temperature object
$C[JK^{-1}]$:	heat capacity of object
$m[kg]$:	mass of object
$\rho[kg\,m^{-3}]$:	specific mass material of object
$V[m^3]$:	volume of object
$c_v[JK^{-1}kg^{-1}]$:	specific heat capacity material of object at constant volume

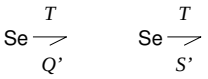
parameters: The values of ρ, c_v and V of the object depend on temperature and pressure. The values for ρ and c_v are assumed to be constant (take average temperature and pressure) and can be chosen from (Verein Deutscher Ingenieure 1977):

water:	chapter Db, tables 1 and 4
air:	chapter Db, tables 14 and 17
construction materials:	chapter Ea, tables 1 and 2

B.5.5 Temperature Source

processes: temperature source (B.4.5), convective temperature source B.4.17)

elements:



relations:

$$T = T(t) \tag{B.26}$$

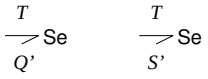
variables:

$t[s]$:	time parameter
$T(t)[K]$:	supplied temperature
$Q'[W]$:	heat flow from environment
$S'[WK^{-1}]$:	entropy flow from environment

B.5.6 Temperature Sink

processes: temperature sink (B.4.6), convective temperature sink B.4.18)

elements:



relations:

$$T = T(t) \tag{B.27}$$

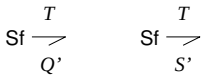
variables:

$t[s]$:	time parameter
$T(t)[K]$:	environment temperature
$Q'[W]$:	heat flow to environment
$S'[WK^{-1}]$:	entropy flow to environment

B.5.7 Heat/Entropy Source

processes: heat/entropy source (B.4.7)

elements:



relations:

$$Q' = Q'(t) \tag{B.28}$$
$$S' = S'(t) \tag{B.29}$$

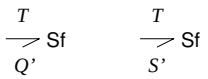
variables:

$t[s]$:	time parameter
$T[K]$:	environment temperature
$Q'(t)[W]$:	heat flow from environment
$S'(t)[WK^{-1}]$:	entropy flow from environment

B.5.8 Heat/Entropy Sink

processes: heat/entropy sink (B.4.8)

elements:



relations:

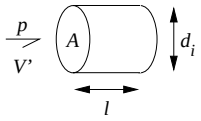
$$Q' = Q'(t) \tag{B.30}$$
$$S' = S'(t) \tag{B.31}$$

variables:

$t[s]$:	time parameter
$T[K]$:	environment temperature
$Q'(t)[W]$:	heat flow to environment
$S'(t)[WK^{-1}]$:	entropy flow to environment

B.5.9 Hydraulic Inertance

geometry:



processes: convection (B.4.9), convection with external conduction (B.4.10)

elements:

relations:

$$V' = \frac{1}{I}\int_0^t p\,dt \tag{B.32}$$

$$I = \rho l/A \tag{B.33}$$

$$A = \frac{1}{4}\pi d_i^2 \tag{B.34}$$

variables:

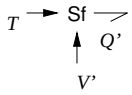
$t[s]$:	time parameter
$p[Pa]$:	pressure
$V'[m^3 s^{-1}]$:	water volume flow
$l[m]$:	tube length
$d_i[m]$:	tube inner diameter
$A[m^2]$:	water flow area
$\rho[kgm^{-3}]$:	specific mass water

parameters: The value for ρ depend on the temperature and pressure of the water. For simulation, it is considered to be constant and can be obtained from the tables in chapter Db in (Verein Deutscher Ingenieure 1977), tables 1 and 4 (take average temperature and pressure).

B.5.10 Convection Source

processes: convection (B.4.9), controlled convection (B.4.12), convection with external conduction (B.4.10), convection splitting/mixing (B.4.13/B.4.15), controlled convection splitting/mixing (B.4.14/B.4.16), convection through hydraulic source (B.4.11), convective temperature source (B.4.17)

elements:



relations:

$$Q' = c_v \rho V' T \tag{B.35}$$

variables:

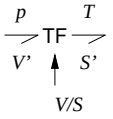
$T[K]$:	temperature
$Q'[W]$:	convective heat flow
$V'[m^3 s^{-1}]$:	water volume flow
$\rho[kgm^{-3}]$:	specific mass water
$c_v[JK^{-1}kg^{-1}]$:	specific heat capacity material of object at constant volume

parameters: The values of ρ and c_v depend on temperature and pressure. The values for ρ and c_v are assumed to be constant (take average temperature and pressure) and can be chosen from (Verein Deutscher Ingenieure 1977) chapter Db, tables 1 and 4 (take average temperature and pressure).

B.5.11 Convection Transformer

processes: convection (B.4.9), controlled convection (B.4.12), convection with external conduction (B.4.10), convection splitting/mixing (B.4.13/B.4.15), controlled convection splitting/mixing (B.4.14/B.4.16), convection through hydraulic source (B.4.11), convective temperature source (B.4.17)

elements:



relations:

$$V' = S'V/S \tag{B.36}$$

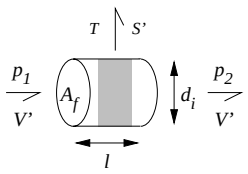
$$T = pV/S \tag{B.37}$$

variables:

$T[K]$:	temperature
$S'[W]$:	convective entropy flow
$V'[m^3 s^{-1}]$:	water volume flow

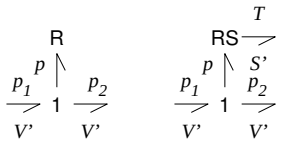
B.5.12 Hydraulic Resistor

general geometry:



processes: convection (B.4.9, convection with external conduction (B.4.10), convection through hydraulic source (B.4.11 *), convection splitting/mixing (B.4.13/B.4.15)

elements:



relations:

$$p = p_1 - p_2 \tag{B.38}$$

$$p = (\xi \frac{l}{d_i} + \xi_s) \frac{\rho}{2A_f^2} V'|V'| \tag{B.39}$$

$$S' = \frac{pV'}{T} = (\xi \frac{l}{d_i} + \xi_s) \frac{\rho}{2A_f^2 T} |V'|^3 \tag{B.40}$$

$$Re = \frac{V' d_i}{\nu A_f} = \frac{V' d_i \rho}{\eta A_f} \tag{B.41}$$

$$A_f = \frac{1}{4} \pi d_i^2 \tag{B.42}$$

variables:

$p[Pa]$:	pressure difference left-right
$p_1[Pa]$:	water pressure left
$p_2[Pa]$:	water pressure right
$V'[m^3 s^{-1}]$:	water volume flow
$T[K]$:	water temperature
$S'[WK^{-1}]$:	entropy flow to water
$Re[-]$:	Reynolds number
$l[m]$:	length of pipe
$A_f[m^2]$:	water flow area
$d_i[m]$:	inner diameter of pipe
$\xi[-]$:	resistance factor (full length effects)
$\xi_s[-]$:	Bernoulli resistance factor (short length effects)
$\nu[m^2 s^{-1}]$:	kinematic viscosity water
$\eta[kgm^{-1}s^{-1}]$:	dynamic viscosity water
$\rho[kgm^{-3}]$:	specific mass water

parameters: The values for ν, η and ρ depend on the temperature and pressure of the water. For simulation, they are considered to be constant and can be obtained from the tables in chapter Db in (Verein Deutscher Ingenieure 1977) (take average temperature and pressure).

Two types of resistance can be modelled: resistance due to effects that occur along the full length of the pipe and due to effects that occur in a short section of the pipe. The first type of effects are internal friction caused by turbulence and friction caused by a rough pipe surface. The influence of these effects on the total resistance is taken into account by the value of ξ, which depends on the Reynolds number and the diameter and roughness of the pipe. Short length effects, such as narrowings or broadenings of the water flow area determine the value of ξ_s.

parameter ξ for smooth surfaces:

Determine ξ with the equations below when the surface of the pipe is smooth.

$$\xi = \begin{cases} \frac{64}{Re} & Re < 2300 & \text{(Hagen - Poiseuille)} \\ \frac{0.3164}{\sqrt[4]{Re}} & 3000 < Re < 10^5 & \text{(Blasius)} \\ 0.00540 + \frac{0.3964}{Re^{0.3}} & 2 \cdot 10^4 < Re < 2 \cdot 10^6 & \text{(Hermann)} \end{cases} \tag{B.43}$$

For $Re > 10^6$ the following equation (Prandtl and Kármán) can be used to determine ξ:

$$\xi = \frac{1}{(-0.8 + 2 \cdot {}^{10}\log [Re\sqrt{\xi}])^2} \tag{B.44}$$

The value of ξ can be determined by using an estimate for ξ on the right hand side of this equation. When the result differs too much from this estimate, the equation is applied again, but now with the previously determined ξ value as the estimate. This is repeated until the computed value of ξ is sufficiently close to the estimate. Hermann's equation (B.43) can supply the first estimation.

parameter ξ for rough surfaces:

Determine ξ with the equations below when the surface of the pipe is rough.

$$\xi = \begin{cases} \frac{64}{Re} & Re < 2300, K \leq 0.07 & \text{(Hagen - Poiseuille)} \\ \frac{1}{(2 \cdot {}^{10}\log [d_i/K] + 1.14)^2} & \text{fully turbulent flow (see text)} & \text{(Prandtl and Kármán)} \end{cases} \tag{B.45}$$

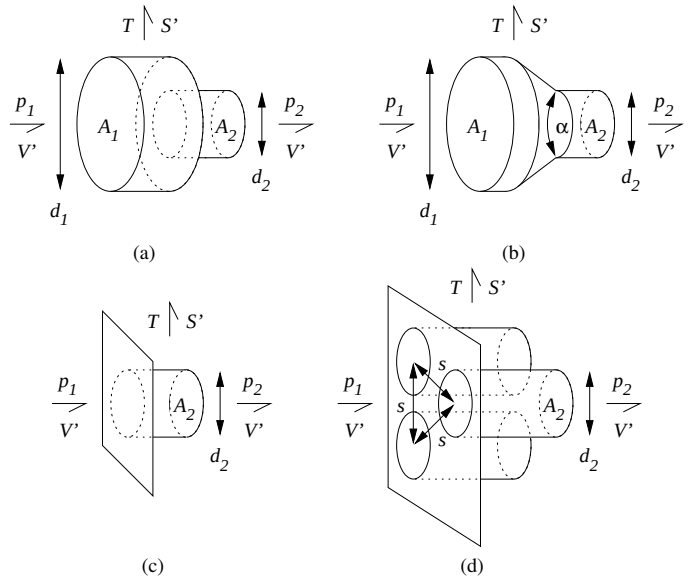
For mixed laminar and turbulent the following equation (Colebrook and White) can be used to determine ξ:

$$\xi = \frac{1}{\left(-2 \cdot {}^{10}\log \left[\frac{2.51}{Re\sqrt{\xi}} + \frac{K/d_i}{3.71}\right]\right)^2} \tag{B.46}$$

The value of ξ can be determined by using an estimate for ξ on the right hand side of this equation. When the result differs too much from this estimate, the equation is applied again, but now with the previously determined ξ value as the estimate. This is repeated until the computed value of ξ is close to the estimate. Prandtl's equation (B.45) can supply the first estimate.

The above scheme can also be used to determine whether Prandtl and Kármáns equation (B.45) can be used, i.e. whether the flow is fully turbulent. This is the case when the determined ξ value is close to the new value given by Eq. (B.46).

parameter ξ_s for narrowings:



For narrowings of the water flow area, determine ξ_s as described below. For all kinds of narrowings, use $d_i = d_2$.

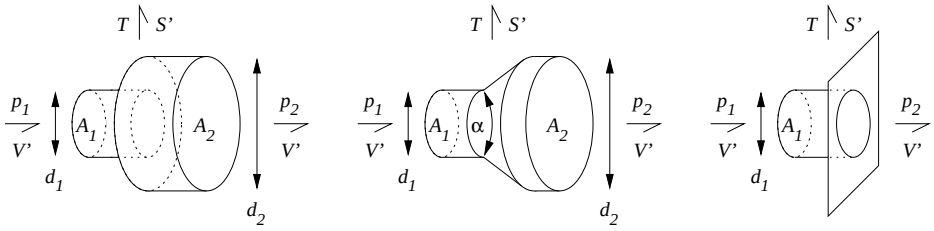
sharp narrowing: Take $\xi_s = \xi_E$ from figure 2 on page Lc 1 in (Verein Deutscher Ingenieure 1977). Note that $f_i/f_k = A_2/A_1$ and that ξ depends on Re, so an estimate has to be made for Re.

gradual narrowing: For gradual narrowings with $20^\circ \leq \alpha \leq 40^\circ$, use $\xi_s = 0.04$. For smooth surfaces and high Re values, ξ_s can be even taken lower.

flow from reservoir: The fluid in the tank is assumed to be at rest. Use $\xi_s = \xi_E$ with ξ_E from figure 4 on page Lc 1 in (Verein Deutscher Ingenieure 1977).

flow into a bundle of pipes: Use $\xi_s = \xi_E$ with ξ_E from figure 3 on page Lc 1 in (Verein Deutscher Ingenieure 1977). These values hold for $Re > 20000$. For laminar flow, use the values from the flow from reservoir case.

parameter ξ_s for broadenings:



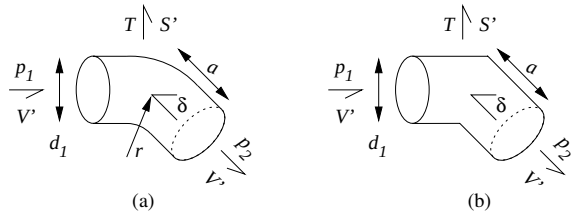
For narrowings of the water flow area, determine ξ_s as described below. For all kinds of narrowings, use $d_i = d_1$.

Sharp broadening: For turbulent flow, take $\xi_s = (1 - A_1/A_2)^2$.

Gradual broadening: For gradual broadenings use $\xi_s = \xi'(1 - A_1/A_2)^2$ with ξ' from figure 8 on page Lc 2 in (Verein Deutscher Ingenieure 1977). Note that ξ' has a minimum for a certain angle α, depending on Re and the roughness of the tube surface.

Flow to reservoir: Take $\xi_s = 1$ (turbulent flow).

parameter ξ_s for bends:

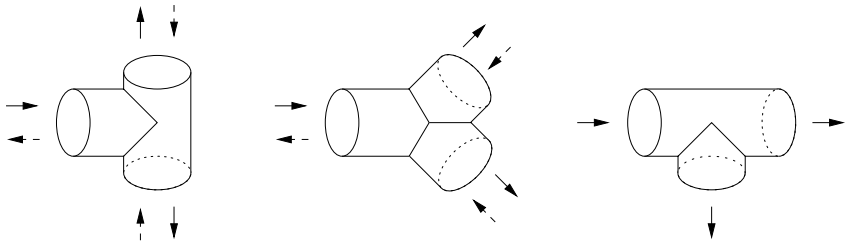


For resistance caused by bends obtain the ξ_s values as described below.

Smooth bends: Use $\xi_s = \xi_u$ from figure 15 on page Lc 5 in (Verein Deutscher Ingenieure 1977). These values are for $Re > 10^5$. The distance to the next effect that causes friction over a short length must be more than $10d_i$ ($a > 10d_i$). When there is a cascade of smooth bends with $\delta = 90°$ directly following each other, use figure 17 on page Lc 5 of (Verein Deutscher Ingenieure 1977) to get the ξ_s value for the whole cascade.

Sharp bends: For sharp bends with $\delta = 90°$ use the tables in paragraph 7.2 on page Lc 5 in (Verein Deutscher Ingenieure 1977) ($Re > 500$). For other angles see figure figure 18 on page Lc 6 in (Verein Deutscher Ingenieure 1977). For cascades of sharp bends see figure 19, also on page Lc 6 in (Verein Deutscher Ingenieure 1977).

parameter ξ_s for splitters and mixers



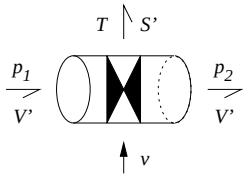
(a) (b) (c)

For symmetric T and Y splitters and mixers, use figure 12 and table 1 on page Lc 3 in (Verein Deutscher Ingenieure 1977) to determine $\xi_s = \xi_v$. For an asymmetric splitter of 90 degrees, use $\xi_s = 0.1$ for the main flow resistance and $\xi_s = 1$ for the flow that is tapped from the main flow. This is according to figure 13 on page Lc 4 in (Verein Deutscher Ingenieure 1977). For asymmetric splitting with angles other than 90 degrees and asymmetric mixing, ξ_s depends on the water volume flows too much. This makes an easy determination of ξ_s impossible.

notes: For non-cylindrical tubes take $d_i = 4A_f/s$, with $A_f[m^2]$ the water flow area and $s[m]$ the perimeter of A_f.

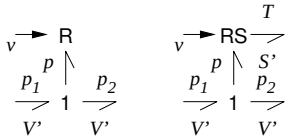
B.5.13 Controlled Hydraulic Resistor

geometry:



processes: controlled convection (B.4.12), controlled convection splitting/mixing (B.4.14/B.4.16)

elements:



relations:

$$p = p_1 - p_2 \tag{B.47}$$

$$p = \frac{p_0}{K_v^2} V' |V'| \tag{B.48}$$

$$K_v = \begin{cases} K_0 + v(K_1 - K_0) & \text{for linear valves} \\ K_0 e^{nv} & \text{for equipercentual valves} \end{cases} \tag{B.49}$$

$$n = \ln (K_1 / K_0) \tag{B.50}$$

$$S' = \frac{pV'}{T} = \frac{p_0}{TK_v^2} |V'|^3 \tag{B.51}$$

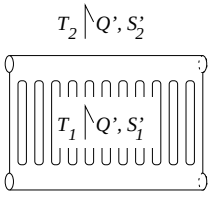
variables:

$p[Pa]$:	pressure difference left-right
$p_0[Pa]$:	nominal pressure difference left-right
$p_1[Pa]$:	water pressure left
$p_2[Pa]$:	water pressure right
$V'[m^3 s^{-1}]$:	water volume flow
$v[-]$:	valve position $0 \leq v \leq 1$; 0 is closed, 1 is opened
$K_v[m^3 s^{-1}]$:	water volume flow for $p = p_0$ and position v
$K_0[m^3 s^{-1}]$:	water volume flow for $p = p_0$ and $v = 0$
$K_1[m^3 s^{-1}]$:	water volume flow for $p = p_0$ and $v = 1$
$n[-]$:	equipercentage number
$T[K]$:	water temperature
$S'[WK^{-1}]$:	entropy flow to water

parameters: The valve manufacturer supplies the values for K_0 and K_1 or n for a certain p_0 value. Usually this value is $p_0 = 10^5 Pa$. For the controlled mixing/splitting processes, the mixing/splitting valve is modelled as two valves. The valve controlling the main flow is equipercentual with $v = v_c$ and the one for the bypass is linear with $v = 1 - v_c$ (see (Recknagel and Sprenger 1979)).

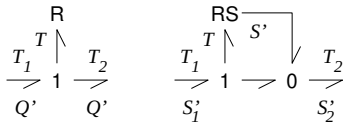
B.5.14 Radiator Resistor

geometry:



processes: convection with external conduction (B.4.10)

elements:



relations:

$$T = T_1 - T_2 \tag{B.52}$$

$$Q' = \frac{K}{N} T^m \tag{B.53}$$

$$S'_1 = \frac{Q'}{T + T_2} = \frac{K T^m}{N(T + T_2)} \tag{B.54}$$

$$S' = \frac{T S'_1}{T_2} = \frac{K T^{m+1}}{T_2 (T + T_2) N} \tag{B.55}$$

$$S'_2 = S'_1 + S \tag{B.56}$$

$$K = \frac{W_n}{T_n^m} \tag{B.57}$$

variables:

$T_1[K]$:	water temperature
$T_2[K]$:	air temperature
$T[K]$:	temperature difference water-air
$T_n[K]$:	nominal temperature difference water-air
$Q'[W]$:	heat flow water-air
$W_n[W]$:	nominal heat flow water-air
$S'_1[W K^{-1}]$:	entropy out flow water
$S'_2[W K^{-1}]$:	entropy in flow air
$S'[W K^{-1}]$:	entropy flow produced by transfer process
$N[-]$:	number of segments
$K[W K^{-m}]$:	radiator dependent parameter
$m[-]$:	radiator dependent parameter

parameters: The parameters T_n and W_n are supplied by the radiator manufacturer. In chapter Fb in (Verein Deutscher Ingenieure 1977), $\dot{q}_n$ is the W_n value per *physical* radiator segment. They use $T_n = 60K$ and $m = 1.30$.

notes: Ham (1988) measured that in some cases equation B.57 and the value $m = 1.30$ are not accurate.

B.5.15 Pressure Source

processes: convection through hydraulic source (B.4.11), convective temperature source (B.4.17)

elements:

relations:

$$p = p(t) \tag{B.58}$$

variables:

$t[s]$:	time parameter
$p(t)[K]$:	supplied pressure
$V'[W]$:	volume flow from environment

B.5.16 Pressure Sink

processes: convective temperature sink (B.4.18)

elements:

relations:

$$p = p(t) \tag{B.59}$$

variables:

$t[s]$:	time parameter
$p(t)[K]$:	external pressure
$V'[W]$:	volume flow to environment

B.5.17 Pressure Reference Source

processes: convection through hydraulic source (B.4.11 *)

elements:

relations:

$$p = 0 \tag{B.60}$$

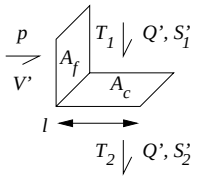
variables:

$p[K]:$	supplied pressure
$V'[W]:$	volume flow from environment

B.5.18 External Conduction Resistor

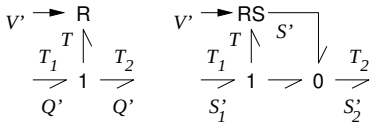
Flat Geometry

geometry:



processes: convection with external conduction (B.4.10)

elements:



relations:

$$T = T_1 - T_2 \tag{B.61}$$

$$Q' = A_t \alpha T \tag{B.62}$$

$$S_1' = \frac{Q'}{T + T_2} = \frac{A_t \alpha T}{T + T_2} \tag{B.63}$$

$$S' = \frac{TS_1'}{T_2} = \frac{A_t \alpha T^2}{T_2(T+T_2)} \tag{B.64}$$

$$S_2' = S_1' + S' \tag{B.65}$$

$$\alpha = \frac{\lambda Nu}{l} \tag{B.66}$$

$$A_t = l\pi d_i \tag{B.67}$$

$$Nu_{lam} = 0.664\sqrt{Re}\sqrt[3]{Pr} \tag{B.68}$$

$$Nu_{turb} = \frac{0.037Re^{0.8}Pr}{1+2.443Re^{-0.1}(Pr^{2/3}-1)} \tag{B.69}$$

$$Nu = \begin{cases} Nu_{lam} & Re < 10^5, 0.6 < Pr < 2000 \\ \sqrt{Nu_{lam}^2 + Nu_{turb}^2} & 10^5 < Re < 5\cdot 10^5, 0.6 < Pr < 2000 \\ Nu_{turb} & 5\cdot 10^5 < Re < 10^7, 0.6 < Pr < 2000 \end{cases} \tag{B.70}$$

$$Re = \frac{V'l}{\nu A_f} = \frac{V'l\rho}{\eta A_f} \tag{B.71}$$

$$A_f = \frac{1}{4}\pi d_i^2 \tag{B.72}$$

$$Pr = \frac{\nu}{a} = \frac{\eta c_p}{\lambda} \tag{B.73}$$

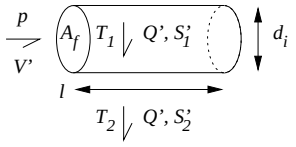
variables:

$T[K]$:	temperature difference water-surface
$T_1[K]$:	temperature water
$T_2[K]$:	temperature surface
$l[m]$:	length of surface in flow direction
$A_t[m^2]$:	heat transfer area
$V'[m^3s^{-1}]$:	water volume flow
$A_f[m^2]$:	water flow area
$\nu[m^2s^{-1}]$:	kinematic viscosity water
$\eta[kgm^{-1}s^{-1}]$:	dynamic viscosity water
$\rho[kgm^{-3}]$:	specific mass water
$Nu[-]$:	Nußel number
$Pr[-]$:	Prandtl number
$a[m^2s^{-1}]$	temperature conduction coefficient water
$c_p[Jkg^{-1}K^{-1}]$:	specific heat capacity water at constant pressure
$Re[-]$:	Reynolds number
$\lambda[WK^{-1}m^{-1}]$:	heat conduction coefficient water
$\alpha[Wm^{-2}K^{-1}]$:	heat transfer coefficient water
$Q'[W]$:	heat flow water to surface
$S_1'[WK^{-1}]$:	entropy out flow water
$S_2'[WK^{-1}]$:	entropy in flow surface
$S'[WK^{-1}]$:	entropy flow produced by transfer process

parameters: The values for ν, η, ρ, λ and c_p depend on the temperature and pressure of the water. For simulation, they are considered to be constant and can be obtained from the tables in chapter Db in (Verein Deutscher Ingenieure 1977) (take average temperature and pressure).

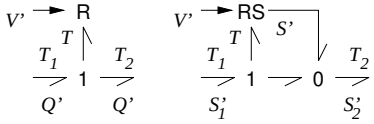
Cylindrical Geometry

geometry:



processes: convection with external conduction (B.4.10)

elements:



relations:

$$T = T_1 - T_2 \tag{B.74}$$

$$Q' = A_t \alpha T \tag{B.75}$$

$$S_1' = \frac{Q'}{T + T_2} = \frac{A\alpha T}{T + T_2} \tag{B.76}$$

$$S' = \frac{TS_1'}{T_2} = \frac{A\alpha T^2}{T_2(T + T_2)} \tag{B.77}$$

$$S_2' = S_1' + S' \tag{B.78}$$

$$A_t = l\pi d_i \tag{B.79}$$

$$\alpha = \frac{\lambda Nu}{d_i} \tag{B.80}$$

$$Nu = \begin{cases} 3.65 + \frac{0.19(RePrd_i/l)^{0.8}}{1+0.117(RePrd_i/l)^{0.467}} & Re < 2300, 0.6 < Pr < 2000 \\ \approx \sqrt[3]{3.66^3 + 1.61^3 RePrd_i/l} & \\ \frac{\xi/8(Re-1000)Pr}{1+12.7\sqrt{\xi/8}(Pr^{2/3}-1)}\left[1 + \left(\frac{d_i}{l}\right)^{2/3}\right] & 2300 < Re < 10^6, 0.6 < Pr < 2000 \end{cases} \tag{B.81}$$

$$\xi = \frac{1}{(1.82^{10}\log Re - 1.64)^2} \tag{B.82}$$

$$A_f = \frac{1}{4}\pi d_i^2 \tag{B.83}$$

$$Re = \frac{V' d_i}{\nu A_f} = \frac{V' d_i \rho}{\eta A_f} \tag{B.84}$$

$$Pr = \frac{\nu}{a} = \frac{\eta c_p}{\lambda} \tag{B.85}$$

variables:

$T[K]$:	temperature difference water-surface
$T_1[K]$:	temperature water
$T_2[K]$:	temperature surface
$l[m]$:	length of cylinder
$d_i[m]$:	inner diameter of cylinder
$A_t[m^2]$:	heat transfer area
$V'[m^3s^{-1}]$:	water volume flow
$\nu[m^2s^{-1}]$:	kinematic viscosity water
$\eta[kgm^{-1}s^{-1}]$:	dynamic viscosity water
$\rho[kgm^{-3}]$:	specific mass water
$Nu[-]$:	Nußel number
$Pr[-]$:	Prandtl number
$\xi[-]$:	pressure loss factor
$a[m^2s^{-1}]$	temperature conduction coefficient water
$c_p[Jkg^{-1}K^{-1}]$:	specific heat capacity water at constant pressure
$Re[-]$:	Reynolds number
$A_f[m^2]$:	water flow area
$\lambda[WK^{-1}m^{-1}]$:	heat conduction coefficient water
$\alpha[Wm^{-2}K^{-1}]$:	heat transfer coefficient water
$W[W]$:	heat flow water to surface
$S'_1[WK^{-1}]$:	entropy out flow water
$S'_2[WK^{-1}]$:	entropy in flow surface
$S'[WK^{-1}]$:	entropy flow produced by transfer process

parameters: For non-cylindrical tubes take $d_i = 4A_f/s$, with $A_f[m^2]$ the water flow area and $s[m]$ the perimeter of A_f. The values for ν, η, ρ, λ and c_p depend on the temperature and pressure of the water. For simulation, they are considered to be constant and can be obtained from the tables in chapter Db in (Verein Deutscher Ingenieure 1977) (take average temperature and pressure).

notes: *These equations only hold for long pipes, i.e.* $d_i/l < 0.1$

B.6 A Large Scale Modelling and Simulation Experiment

B.6.1 Introduction

The OLMECO library is meant to support the compositional modelling of technical systems. To test the usefulness of this library, large scale modelling experiments have been carried out. In this chapter, the modelling of a real world thermodynamic system is described. The system that has been modelled is the heating system of the Schieland Hospital, a hospital in Schiedam, the Netherlands. In this experiment the model components for the OLMECO library described previously were used. This chapter tries to find answers to the following questions:

1. How well does the OLMECO library support modelling?
2. Can a real life system be modelled with the thermodynamic model components in Section B.3–B.5?

The outline of this section is as follows. First, the heating system of the Schieland Hospital will be described. In section B.6.3 the composition of a simulation model of that system with components from the OLMECO library will be explained. Before simulation runs can be made, the parameters in the model have to be determined. This is done in section B.6.4. Two simulation experiments carried out with the model, are described in section B.6.5. The conclusions of the modelling experiment can be found in section B.6.6

B.6.2 The Schieland Hospital Heating System

The modelling experiment has been carried out on the Schieland Hospital heating system. This system has been modelled before by R. Andringa and E. van der Laan and is described in (Andringa and van der Laan 1991). This work has been done by the firm Kropman B.V., Rijswijk, one of the large building services company in The Netherlands (see also (Zeiler 1994)). This firm designed and installed the complete heating system of the Schieland hospital.

In the model of (Andringa and van der Laan 1991), the following simplifications have been incorporated:

- The heaters and boilers in the system are aggregated into one heater.
- Hot water supply has been left out of the system.
- The many radiator groups in the hospital have been aggregated into just one.

We have chosen to model the same system with the same simplifications for the following reasons:

- To be able to compare the results with (Andringa and van der Laan 1991).
- Because (Andringa and van der Laan 1991) gives a lot of information on the actual Schieland system.
- Even with the simplifications, the model is a large scale model.

Some data on the model described in this chapter support this last point:

statistics about the model	
19	model components
1	user defined component
18	components from the library
9	component classes from the library
4	decompositions from the library
26	coupled differential equations
± 150	equations

The drawing of the simplified system can be found in Figure B.4. Clearly can be seen that the system consists of two coupled subsystems: one subsystem around the heater (heater group, abbreviated hg) and one around the radiator (radiator group or rg).

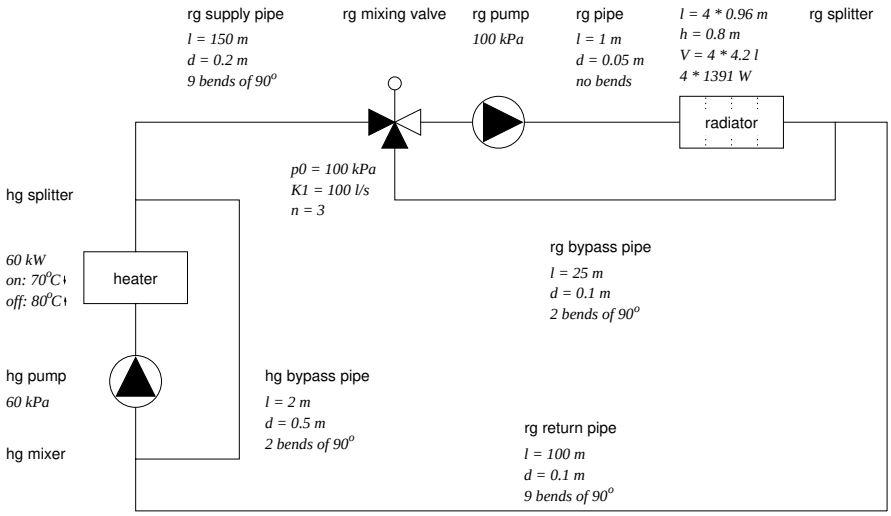


Figure B.4: The simplified Schieland Hospital heating system.

B.6.3 The Model

From Figure B.4, a model was constructed using component types from the OLMECO library. This model can be found in Figure B.5. For most components in the system the choice from the library is pretty straightforward. For the modelling decisions that were not so obvious, this section gives an explanation. But first the way that Figure B.5 has to be read needs further attention.

Because the model editor that was used to build the model lacks the concept of components, so word bond graphs had to be used to mimic components. In that sense, Figure B.5 gives the component structure of the system. The components in the system are the ovals which are in fact `misused' word bond graphs. For simple components, i.e. components which are not composed out of subcomponents, the upper label printed inside the oval is a label that identifies a *process description* in the OLMECO library that models the behaviour of the component. The lower label inside an oval is an abbreviation

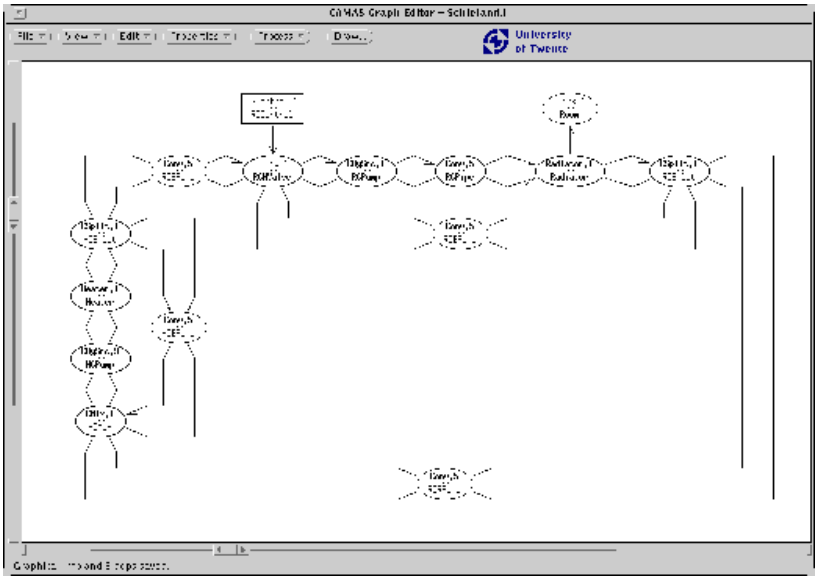


Figure B.5: Top level of the Schieland model.

of the *model component name*. For instance, the model component Heater Group Bypass Pipe has label HGBPipe. Table B.20 lists all simple components in the model together with the type of the component, a label that identifies this component type in the library, the primary and secondary processes that model the behaviour of the component and a label that identifies the process description in the library.

For complex components, the lower label in the oval is the model component name. The upper label is the name of a sub bond graph that is (mis)used to show the way the complex component is built out of its subcomponents, just like the bond graph in Figure B.5 gives the component structure of the system. Table B.21 lists the complex model components, their library component type, the model names of the subcomponents and the label identifying the decomposition in the library that was used. The subcomponents that are atomic (simple) can be found in table B.20.

Now that Figure B.5 is explained the modelling decisions can be given some attention.

At the bond graph level, pseudo thermal process descriptions are used. For process descriptions that are not in the library, bond graphs are given here. For bond graphs of the standard library process descriptions the reader is referred to the previous chapters. In all submodels used, no internal conduction is present. Furthermore, it is assumed that, except for the radiator, there is no heat loss to the environment. Hydraulic inertance is modelled for those components that contain a lot of water, i.e. the convection pipe components. This enables to simulate the acceleration of the water when the pumps are switched on. In other submodels, hydraulic inertance is neglected.

Hydraulic inertance is modelled mathematically with the linear equation $V' = \frac{1}{I} \int p \; dt$ with volume flow V' and pressure p. The mathematical relation for hydraulic resistance is $p = R_h V' |V'|$. Section B.6.4 discusses the way R_h can be determined and the validity of this relation.

component	type	component label	modelled processes	process label
HG Pump	Pump	Pump.1	Conv. through a Hydr. Source reference pressure	CHySrc.5
Heater.HSource	Heat Source	HSource.1	Temperature Source	ESource.1
Heater.HPipe	Pipe w. Ext. Cond.	PipeEC.1	Conv. thr. a Pipe w. Ext. Cond. hydraulic inertance	ConvEC.5
HG Splitter	Splitter	Splitter.1	Convection Splitting	CSplit.1
HG Bypass Pipe	Pipe	Pipe.1	Convection through a Pipe hydraulic inertance	Conv.5
HG Mixer	Mixer	Mixer.1	Convection Mixing	CMix.1
RG Supply Pipe	Pipe	Pipe.1	Convection through a Pipe hydraulic inertance	Conv.5
RG Mixing Valve	Controlled Mixer	CMixer.1	Controlled Convection Mixing	CCMix.1
RG Pipe	Pipe	Pipe.1	Convection through a Pipe hydraulic inertance	Conv.5
RG Pump	Pump	Pump.1	Conv. through a Hydr. Source	CHySrc.1
Radiator.Left.Seg_1	Pipe w. Ext. Cond.	PipeEC.1	Conv. thr. a Pipe w. Ext. Cond. hydraulic inertance transfer resistance	ConvEC.13
Radiator.Left.Seg_2	Pipe w. Ext. Cond.	PipeEC.1	Conv. thr. a Pipe w. Ext. Cond. hydraulic inertance transfer resistance	ConvEC.13
Radiator.Right.Seg_1	Pipe w. Ext. Cond.	PipeEC.1	Conv. thr. a Pipe w. Ext. Cond. hydraulic inertance transfer resistance	ConvEC.13
Radiator.Right.Seg_2	Pipe w. Ext. Cond.	PipeEC.1	Conv. thr. a Pipe w. Ext. Cond. hydraulic inertance transfer resistance	ConvEC.13
Room	Heat Sink	HSink.1	Temperature Sink	TSink.1
RG Splitter	Splitter	Splitter.1	Convection Splitting	CSplit.1
RG Bypass Pipe	Pipe	Pipe.1	Convection through a Pipe hydraulic inertance	Conv.5
RG Return Pipe	Pipe	Pipe.1	Convection through a Pipe hydraulic inertance	Conv.5

Table B.20: Simple model components: the columns give 1) the name of the component as used (abbreviated) in the model, 2) the component type from the library, 3) the library label of the component type, 4) the modelled processes and 5) the library label of the process description of the component.

component	type	subcomponents	decomposition label
Heater	Closed Syst. Heater	HSource, HPipe	CSHeater.2
Radiator	Pipe w. Ext. Cond.	Left, Right	PipeEC.2
Radiator.Left	Pipe w. Ext. Cond.	Seg_1, Seg_2	PipeEC.2
Radiator.Right	Pipe w. Ext. Cond.	Seg_1, Seg_2	PipeEC.2

Table B.21: Complex model components: the columns give 1) the model component name, 2) the component type from the library, 3) the subcomponent names in the model and 4) the library label of the decomposition.

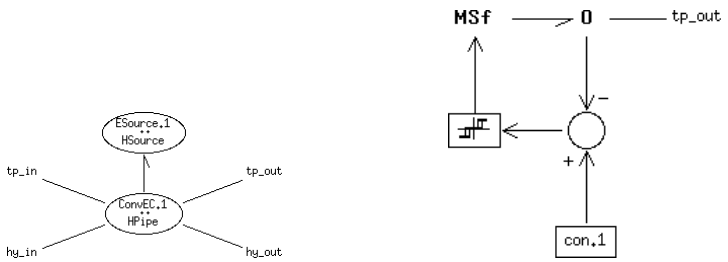


Figure B.6: Decomposition of the heater (left) and the heat source of the heater (right).

Splitters and Mixers The splitters and mixers connect pipes with different diameters. One possibility is to model the required narrowings and broadenings as short pipe segments with high hydraulic resistance. In this experiment however, the changes of diameter are viewed to be part of the splitters and mixers. This means that there are no separate narrowing or broadening pipe segments and that the friction is incorporated into the bond graphs for the mixers and splitters.

Pumps For the pumps in the heater and radiator groups, not surprisingly, the pump component was chosen. More interesting is the fact that, for the heater group pump, a bond graph with a reference pressure source was required to make simulation possible.

Heater The heater is a closed system heater component which is decomposed into a pipe with external conduction component and a heat source (Figure B.6). The pipe is $20m$ long, has a diameter of $0.1m$ and has 50 bends of 180 degrees. Because the heat source connected to the heater pipe is modelled as a controlled energy source (MSf), there is no need to model heat conduction in the heater pipe or the heat transfer from the pipe to the water. Therefore, no transfer resistance element is present in the bond graph. The bond graph of the heat source (Figure B.6) is an adapted version of the entropy source process description in the library. It is a modulated flow source which supplies a constant flow of energy when the temperature drops below a certain low temperature value. It is switched off as soon as the temperature reaches some higher temperature value.

Radiator Group Mixing Valve The radiator group mixing valve is modelled with a controlled mixer component. At the mathematical relation level, one modulated hydraulic resistance has an exponential (or equipercentual) equation and the other a linear one. The constitutive relation for controlled valves (Recknagel and Sprenger 1979) is given below. The value of v defines the state of the valve and can vary from 0.0 to 1.0: from fully closed to fully open. The parameters $K0$ and $K1$ are the volume flows through the valve at $p = p_0$ and $v = 0$ and $v = 1$, respectively.

$$
\begin{aligned}
p &= \frac{p_0}{K_v^2} V|V'| \\
K_v &= \begin{cases} K_0 + v(K_1 - K_0) & \text{for linear valves} \\ K_0 e^{nv} & \text{for exponential valves} \end{cases} \\
n &= \ln(K1/K0)
\end{aligned}
$$

In order to prevent undue complexity of the model it is assumed that pressure losses due to the narrowings and broadenings in the flow paths are included in the valve characteristics. The control signal is taken from a specially constructed submodel (Figure B.7) which has a step output signal.

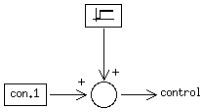


Figure B.7: The submodel of the radiator group mixing-valve controller.

Radiator According to (Ham 1988), the heat flow W from a radiator to the air can be determined with:

$$W = KT^m,$$

where T is the average temperature difference between the water and the air and m and K radiator dependent parameters. The parameter K is defined as

$$K = \frac{W_n}{T_n^m},$$

where W_n is the heat flow at a nominal average temperature difference T_n. Usually $T_n = 60K$ and $m = 1.30$ are good values. In the Schieland model, not the average temperature difference of the whole radiator is used, but instead the radiator is divided in N segments each causing a heat flow defined by:

$$W_i = \frac{K}{N} T_i^m.$$

The radiator is a pipe with external conduction connected to a heat source modelling the room. The pipe is decomposed twice (Figure B.8). Thus, the radiator is split into four radiator segments. The bond graphs of these segments include hydraulic inertance and a transfer resistance, with the mathematical relation above. The heat sink has a temperature source as process description. This source supplies the room temperature, which is assumed to be constant.

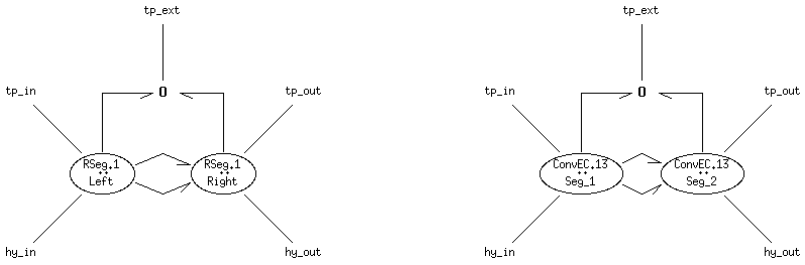


Figure B.8: First and second-level decomposition of the radiator.

B.6.4 Determination of the Model Parameters

Basic calculations

The calculations to obtain numerical values for the model parameters are not very difficult but, because many parameters need to be determined, requires a lot of work. From geometric data from the components, and characteristic values for water, air and the materials the components are made of, the parameters can be computed. In the ideal case, the library contains the parameter relations to do so and a modelling tool is able to obtain the geometric data and characteristic values from CAD drawings and databases. At present, the OLMECO library has the ability to store parameter relations, but unfortunately the modelling tool cannot work with them. Therefore, the parameters had to be determined by hand.

Because many components in the model are modelled as combinations of pipe segments, it is not very surprising that the component parameters depend on the parameters of these pipe segments. For instance, the heat capacity of a splitter is the sum of the heat capacity of the pipe segment where the water flows into the splitter, and of the capacities of the two pipes where water can leave the splitter. To facilitate and speed up the determination of the model parameters, utilities have been written that determine the parameters of the pipe segments. The utilities are `pipe`, `ncpipe`, `short`, `narrow` and `broad`. They are short programs written in C that take input values from the command line and output a table of the parameters of the segment which can be directly included in a LaTeXdocument. In the paragraphs below, these utilities are described.

The `pipe` utility The utility `pipe` computes parameters of a cylindrical pipe segment. Table B.22 shows the input values and the relations that are used.

The water flow area and water volume are calculated with the usual geometric relations. With formulas (1) and (9) on page Lb1 and (1) on page Lc1 in (Verein Deutscher Ingenieure 1977), the hydro-kinetic parameters are determined. The specific mass of water is assumed to be constant and is estimated to be $988kg\ m^{-3}$ (table 2 on page Db1 in (Verein Deutscher Ingenieure 1977)). The hydraulic resistance R_h is used in the model to relate pressure p to volume flow V' according to

$$p = R_h V' |V'|.$$

This relation holds for fully turbulent flow. To determine whether this assumption holds, an iterative method for relation (10) on page Lb3 in (Verein Deutscher Ingenieure 1977) can be used to determine a more accurate value for ξ (see table B.22). If this new value is close to the value computed by `pipe`, the flow is fully turbulent. Unfortunately, the simulator offers no facilities to check this. Roughness factors can be obtained from table 1 on page Lb3 in (Verein Deutscher Ingenieure 1977). A Bernoulli resistance factor can be specified to account for bends in the pipe or other reasons for loss of pressure such as flow into, or out of a reservoir, splitting/mixing losses, valves etc. (see chapter Lc in (Verein Deutscher Ingenieure 1977)).

The hydro-thermal parameters that are computed relate to the following equations for heat storage:

$$T = \begin{cases} Q/C, & Q = \int_0^t Q'dt + Q_0, \quad \text{pseudo thermal} \\ T_0 e^{S/C}, & S = \int_0^t S'dt + S_0, \quad \text{thermodynamic} \end{cases}$$

The parameters can be computed with the specific heat capacity c and the heat conduction coefficient λ of water. In these calculations, the influence of the material of the pipe on the heat storage and

conduction are neglected. Tables 1 and 7 in (Verein Deutscher Ingenieure 1977) chapter Db were used to estimate c to $4.18 \times 10^3\, J\ K^{-1}$ and λ to $0.66 W\ K^{-1}\ m^{-1}$. Note that these values are considered to be constant although in reality they depend on temperature and pressure.

Long cylindrical pipe			
geometry:			
length	l	input value	m
inner diameter	d_i	input value	m
flow area	A	$= \frac{1}{4}\pi d_i^2$	m^2
volume	V	$= lA$	m^3
hydro kinetic:			
mass	m	$= \rho V$	kg
inertance	I	$= \frac{\rho l}{A}$	$kg\ m^4$
roughness pipe surface	K	input value	
resistance factor	ξ	$= \frac{1}{[2\times^{10}\log(d_i/K)+1.14]^2}$	
bernoulli resistance factor	ξ_s	input value	
hydraulic resistance	R_h	$= (\xi\frac{l}{d_i} + \xi_s)\frac{\rho}{2A^2}$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= mc$	$J\ K^{-1}$
initial temperature	T_0	input value	K
initially stored heat	Q_0	$= CT$	J
initially stored entropy	S_0	$= C$	$J\ K^{-1}$
thermal resistance	R_t	$= \frac{l}{\lambda A}$	$K\ W^{-1}$

Table B.22: Computation of parameters of long cylindrical pipes.

The `ncpipe` utility This utility works the same as `pipe` with one exception. The hydraulic diameter d_h is used instead of d_i, and is determined from the flow area A, which is now an input value, and s, the perimeter of the flow area. This way, the parameters for non cylindrical pipes can be calculated. The parameters are only valid for turbulent water flow.

The `short` utility The utility `short` computes parameters of a cylindrical pipe where the hydraulic resistance is caused by an effect that occurs in a short segment of a pipe. This is called a Bernoulli resistance. Examples of Bernoulli resistances are resistances caused by bends, or other reasons for loss of pressure such as flow into, or out of a reservoir, splitting/mixing losses, valves etc. (see chapter Lc in (Verein Deutscher Ingenieure 1977)). The mathematical relations are the same as those of the `pipe` utility when ξ is assumed to be zero. Figure B.24 shows the input values and the relations which are used.

The `broad` utility The utility `broad` computes parameters of a cylindrical pipe which has a sharp broadening. The water is assumed to flow from the narrow to the broad part of the pipe. The equation used is (6) on page Lc2 in (Verein Deutscher Ingenieure 1977) which is valid for turbulent flow.

The `narrow` utility The utility `narrow` computes parameters of a cylindrical pipe which has a sharp narrowing. The water is assumed to flow from the broad to the narrow part of the pipe. The

Long noncylindrical pipe			
geometry:			
length	l	input value	m
flow area	A	input value	m^2
perimeter flow area	s	input value	m
hydraulic diameter	d_h	$= \frac{4A}{s}$	m
volume	V	$= lA$	m^3
hydro kinetic:			
mass	m	$= \rho V$	kg
inertance	I	$= \frac{\rho l}{A}$	$kg\ m^4$
roughness pipe surface	K	input value	
resistance factor	ξ	$= \frac{1}{[2\times^{10}\log(d_h/K)+1.14]^2}$	
bernoulli resistance factor	ξ_s	input value	
hydraulic resistance	R_h	$= (\xi\frac{l}{d_h}+\xi_s)\frac{\rho}{2A^2}$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= mc$	$J\ K^{-1}$
temperature	T_0	input value	K
initially stored heat	Q_0	$= CT$	J
initially stored entropy	S_0	$= C$	$J\ K^{-1}$
thermal resistance	R_t	$= \frac{l}{\lambda A}$	$K\ W^{-1}$

Table B.23: Computation of parameters of long noncylindrical pipes.

Short cylindrical pipe			
geometry:			
length	l	input value	m
inner diameter	d_i	input value	m
flow area	A	$= \frac{1}{4}\pi d_i^2$	m^2
volume	V	$= lA$	m^3
hydro kinetic:			
mass	m	$= \rho V$	kg
inertance	I	$= \frac{\rho l}{A}$	$kg\ m^4$
bernoulli resistance factor	ξ_s	input value	
hydraulic resistance	R_h	$= \xi_s\frac{\rho}{2A^2}$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= mc$	$J\ K^{-1}$
initial temperature	T_0	input value	K
initially stored heat	Q_0	$= CT$	J
initially stored entropy	S_0	$= C$	$J\ K^{-1}$
thermal resistance	R_t	$= \frac{l}{\lambda A}$	$K\ W^{-1}$

Table B.24: Computation of parameters of short cylindrical pipes.

Sharply broadening short cylindrical pipe			
geometry:			
length narrow part	l_1	input value	m
inner diameter narrow part	d_1	input value	m
flow area narrow part	A_1	$= \frac{1}{4}\pi d_1^2$	m^2
length broad part	l_2	input value	m
inner diameter broad part	d_2	input value	m
flow area broad part	A_2	$= \frac{1}{4}\pi d_2^2$	m^2
volume	V	$= l_1 A_1 + l_2 A_2$	m^3
hydro kinetic:			
mass	m	$= \rho V$	kg
inertance	I	$= \frac{\rho l_1}{A_1} + \frac{\rho l_2}{A_2}$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= (1 - A_1/A_2)^2$	
hydraulic resistance	R_h	$= \xi_s \frac{\rho}{2A_1^2}$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= mc$	$J\ K^{-1}$
initial temperature	T_0	input value	K
initially stored heat	Q_0	$= CT$	J
initially stored entropy	S_0	$= C$	$J\ K^{-1}$

Table B.25: Computation of parameters of sharply broadening short cylindrical pipes.

equation used is derived from Figure!2 on page Lc1 of (Verein Deutscher Ingenieure 1977) and is valid for Reynolds numbers of about 10^4 (in the narrow part).

Determination of the Model Parameters

Three assumptions have been made in the determination of the parameters:

- Where heat capacities and initially stored heat had to be determined, the heat capacity of the water only was calculated. The capacity of the material of the various components was not taken into account.
- Initial temperatures were assumed to be $293K$ ($= 20°\ C$, $x°\ C \approx x + 273\ K$).
- Initial water volume flows were assumed to be zero.

Heater Group Pump The heater group pump is modelled with the convection hydraulic source submodel with a reference pressure. The pressure of the incoming water (reference) is defined to be $0Pa$. The pump increases this pressure by $40 \times 10^3 Pa$. The pump is assumed to be ideal, i.e. there is no hydraulic resistance. The heat capacity of a $0.3m$ long tube with $d_i = 0.1m$ serves as an of estimate for the heat capacity of the pump (see table B.27).

Heater The heater can supply $60kW$ of heat at a maximum water temperature of $353K$. It is modelled as a temperature source connected to a convection pipe with external conduction. The pipe is $20m$ long and has $d_i = 0.1m$ and $d_o = 0.12m$. The inner surface has a roughness factor of 0.05

Sharply narrowing short cylindrical pipe			
geometry:			
length broad part	l_1	input value	m
inner diameter broad part	d_1	input value	m
flow area broad part	A_1	$= \frac{1}{4}\pi d_1^2$	m^2
length narrow part	l_2	input value	m
inner diameter narrow part	d_2	input value	m
flow area narrow part	A_2	$= \frac{1}{4}\pi d_2^2$	m^2
volume	V	$= l_1 A_1 + l_2 A_2$	m^3
hydro kinetic:			
mass	m	$= \rho V$	kg
inertance	I	$= \frac{\rho l_1}{A_1} + \frac{\rho l_2}{A_2}$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 0.5 - 0.4 A_2/A_1$	
hydraulic resistance	R_h	$= \xi_s \frac{\rho}{2A_2^2}$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= mc$	$J\ K^{-1}$
initial temperature	T_0	input value	K
initially stored heat	Q_0	$= CT$	J
initially stored entropy	S_0	$= C$	$J\ K^{-1}$

Table B.26: Computation of parameters of sharply narrowing short cylindrical pipes.

Short cylindrical pipe			
geometry:			
length	l	$= 3.000000e-01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 2.356194e-03$	m^3
hydro kinetic:			
mass	m	$= 2.327920e+00$	kg
inertance	I	$= 3.773882e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 0.000000e+00$	
hydraulic resistance	R_h	$= 0.000000e+00$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 9.730706e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.851097e+06$	J
initially stored entropy	S_0	$= 9.730706e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 5.787452e+01$	$K\ W^{-1}$

Table B.27: Parameters of the heater group pump.

(table 1, page Lb3 in (Verein Deutscher Ingenieure 1977)). The pipe has 50 sharp bends of 180 degrees. Each bend accounts for a Bernoulli resistance factor of 0.3 (Figure 15, page Lc5 in (Verein Deutscher Ingenieure 1977)). Table B.28 shows the parameters computed.

Long cylindrical pipe			
geometry:			
length	l	$= 2.000000e+01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 1.570796e-01$	m^3
hydro kinetic:			
mass	m	$= 1.551947e+02$	kg
inertance	I	$= 2.515921e+06$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 3.295139e-01$	
bernoulli resistance factor	ξ_s	$= 1.500000e+01$	
hydraulic resistance	R_h	$= 6.479040e+08$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 6.487138e+05$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 1.900731e+08$	J
initially stored entropy	S_0	$= 6.487138e+05$	$J\ K^{-1}$
thermal resistance	R_t	$= 3.858302e+03$	$K\ W^{-1}$

Table B.28: Parameters of the heater pipe.

The parameters of the heat source are a constant minimum temperature below which the heater is switched on ($343K$), the heat flow when the heater is off ($0W$), the heat flow when it is on ($60kW$), a 'dead zone' ($0K$), and a value for the hysteresis. This value determines the temperature at which the heater is switched off again and must be the switch-off temperature minus the switch-on temperature ($353 - 343 = 10K$).

Heater Group Splitter This splitter has one inflow with $d_i = 0.1m$, one outflow to the radiator group supply pipe with $d_i = 0.2m$ and another outflow to the heater bypass pipe with $d_i = 0.5m$. It is modelled with a convection splitter model. To estimate the heat capacity of the splitter, the capacity of a pipe with $l = 0.1m$, $d_i = 0.1m$ is added to the capacity of a pipe with $l = 0.1m$ with a sharp broadening from $d_i = 0.1$ to $0.2m$ in the middle and a pipe with $l = 0.1m$ with a sharp broadening from $d_i = 0.1$ to $0.5m$ in the middle. This gives $53518.881J\ K^{-1}$ for the heat capacity and $15681029.6J$ for the stored heat.

To determine the two hydraulic resistances of the splitter, the resistance of a splitter with $d_i = 0.1m$ is calculated with the `short` utility (table B.29). The Bernoulli splitting resistance factor is estimated with table 1 on page Lc3 in (Verein Deutscher Ingenieure 1977) at 0.03. This resistance is added to the resistance of each of the sharp broadenings, obtained with the `broad` utility (table B.30 and B.31). This gives $4744992.8Pa^2\ m^{-6}$ for the heater to supply pipe resistance and $7620818.8Pa^2\ m^{-6}$ for the heater to heater bypass pipe resistance.

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 7.853982e-04$	m^3
hydro kinetic:			
mass	m	$= 7.759734e-01$	kg
inertance	I	$= 1.257961e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 3.000000e-02$	
hydraulic resistance	R_h	$= 2.402528e+05$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 3.243569e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 9.503656e+05$	J
initially stored entropy	S_0	$= 3.243569e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 1.929151e+01$	$K\ W^{-1}$

Table B.29: Parameters of the heater group splitter (i).

Sharply broadening short cylindrical pipe			
geometry:			
length narrow part	l_1	$= 5.000000e-02$	m
inner diameter narrow part	d_1	$= 1.000000e-01$	m
flow area narrow part	A_1	$= 7.853982e-03$	m^2
length broad part	l_2	$= 5.000000e-02$	m
inner diameter broad part	d_2	$= 2.000000e-01$	m
flow area broad part	A_2	$= 3.141593e-02$	m^2
volume	V	$= 1.963495e-03$	m^3
hydro kinetic:			
mass	m	$= 1.939933e+00$	kg
inertance	I	$= 7.862254e+03$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 5.625000e-01$	
hydraulic resistance	R_h	$= 4.504740e+06$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+06$	J
initially stored entropy	S_0	$= 8.108922e+03$	$J\ K^{-1}$

Table B.30: Parameters of the heater group splitter (ii).

Sharply broadening short cylindrical pipe			
geometry:			
length narrow part	l_1	$= 5.000000e-02$	m
inner diameter narrow part	d_1	$= 1.000000e-01$	m
flow area narrow part	A_1	$= 7.853982e-03$	m^2
length broad part	l_2	$= 5.000000e-02$	m
inner diameter broad part	d_2	$= 5.000000e-01$	m
flow area broad part	A_2	$= 1.963495e-01$	m^2
volume	V	$= 1.021018e-02$	m^3
hydro kinetic:			
mass	m	$= 1.008765e+01$	kg
inertance	I	$= 6.541395e+03$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 9.216000e-01$	
hydraulic resistance	R_h	$= 7.380566e+06$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 4.216639e+04$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 1.235475e+07$	J
initially stored entropy	S_0	$= 4.216639e+04$	$J\ K^{-1}$

Table B.31: Parameters of the heater group splitter (iii).

Heater Group Bypass Pipe This is a $2m$ long pipe with $d_i = 0.5m$ and two sharp bends each accounting for $\xi_s = 2.2$ (section 7.2, page Lc5 in (Verein Deutscher Ingenieure 1977)). The roughness is again estimated at 0.05. The parameter values can be found in table B.32.

Heater Group Mixer The heater group mixer has one inflow from the radiator group return pipe with $d_i = 0.1m$, one from the heater group bypass pipe with $d_i = 0.5m$, and an outflow with $d_i = 0.1$. It is modelled with a convection mixer submodel. The heat capacity and the initially stored heat are obtained by addition of the parameters from one of the inflow pipes to one half of the capacity of the outflow pipe. For the radiator group return pipe to heater group pump flow this means that the capacity is one and a half times the capacity of a $0.1m$ long pipe with $d_i = 0.1m$ (table B.33). This gives $4865.3535J\ K^{-1}$ for the heat capacity and $1425548.40J$ for the initially stored heat. For the heater group bypass pipe to heater group pump flow this means that one half of the values for a $0.1m$ long pipe with $d_i = 0.1m$ (table B.33) has to be added to the parameters of a $0.1m$ long pipe with a sharp narrowing in the middle from $d_i = 0.5$ to $0.1m$ (table B.34). This gives $43788.1745J\ K^{-1}$ for the heat capacity and $12829932.80J$ for the stored heat.

The hydraulic resistance from the radiator group return pipe to the heater group pump consists only of the mixing resistance computed by `short` (table B.33) with $\xi_s = 0.075$ (table 1 on page Lc3 of (Verein Deutscher Ingenieure 1977)). Its value is $600632.0Pa^2\ m^{-6}$. The narrowing resistance displayed in table B.34 is added to this to obtain the resistance from the heater group bypass pipe to the heater group pump. This value is $4476710.0Pa^2\ m^{-6}$.

Radiator Group Supply Pipe This is a $150m$ long pipe with $d_i = 0.2m$ and 9 sharp bends each accounting for $\xi_s = 1.8$ (section 7.2 on page Lc5 in (Verein Deutscher Ingenieure 1977)). The roughness is again estimated at 0.05. See table B.35 for the results.

Long cylindrical pipe			
geometry:			
length	l	$= 2.000000e+00$	m
inner diameter	d_i	$= 5.000000e-01$	m
flow area	A	$= 1.963495e-01$	m^2
volume	V	$= 3.926991e-01$	m^3
hydro kinetic:			
mass	m	$= 3.879867e+02$	kg
inertance	I	$= 1.006369e+04$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 1.014240e-01$	
bernoulli resistance factor	ξ_s	$= 4.400000e+00$	
hydraulic resistance	R_h	$= 6.157770e+04$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 1.621784e+06$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 4.751828e+08$	J
initially stored entropy	S_0	$= 1.621784e+06$	$J\ K^{-1}$
thermal resistance	R_t	$= 1.543321e+01$	$K\ W^{-1}$

Table B.32: Parameters of the heater group bypass pipe.

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 7.853982e-04$	m^3
hydro kinetic:			
mass	m	$= 7.759734e-01$	kg
inertance	I	$= 1.257961e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 7.500000e-02$	
hydraulic resistance	R_h	$= 6.006320e+05$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 3.243569e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 9.503656e+05$	J
initially stored entropy	S_0	$= 3.243569e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 1.929151e+01$	$K\ W^{-1}$

Table B.33: Parameters of the heater group mixer (i).

Sharply narrowing short cylindrical pipe			
geometry:			
length broad part	l_1	$= 5.000000e-02$	m
inner diameter broad part	d_1	$= 5.000000e-01$	m
flow area broad part	A_1	$= 1.963495e-01$	m^2
length narrow part	l_2	$= 5.000000e-02$	m
inner diameter narrow part	d_2	$= 1.000000e-01$	m
flow area narrow part	A_2	$= 7.853982e-03$	m^2
volume	V	$= 1.021018e-02$	m^3
hydro kinetic:			
mass	m	$= 1.008765e+01$	kg
inertance	I	$= 6.541395e+03$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 4.840000e-01$	
hydraulic resistance	R_h	$= 3.876078e+06$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 4.216639e+04$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 1.235475e+07$	J
initially stored entropy	S_0	$= 4.216639e+04$	$J\ K^{-1}$

Table B.34: Parameters of the heater group mixer (ii).

Long cylindrical pipe			
geometry:			
length	l	$= 1.500000e+02$	m
inner diameter	d_i	$= 2.000000e-01$	m
flow area	A	$= 3.141593e-02$	m^2
volume	V	$= 4.712389e+00$	m^3
hydro kinetic:			
mass	m	$= 4.655840e+03$	kg
inertance	I	$= 4.717353e+06$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 1.819870e-01$	
bernoulli resistance factor	ξ_s	$= 1.620000e+01$	
hydraulic resistance	R_h	$= 7.642553e+07$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 1.946141e+07$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 5.702194e+09$	J
initially stored entropy	S_0	$= 1.946141e+07$	$J\ K^{-1}$
thermal resistance	R_t	$= 7.234316e+03$	$K\ W^{-1}$

Table B.35: Parameters of the radiator group supply pipe.

Radiator Group Mixing Valve This valve is modelled with a controlled mixer model in which the mixing valve is considered a convection mixer with two coupled controlled valves at the in flows. To estimate the two heat capacities of the mixing valve, the capacity of a $0.1m$ long pipe with $d_i = 0.1$ (table B.36) is multiplied by 1.5. This gives 4865.3535 for the heat capacities and 1425548.40 for the stored heat parameters.

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 7.853982e-04$	m^3
hydro kinetic:			
mass	m	$= 7.759734e-01$	kg
inertance	I	$= 1.257961e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 0.000000e+00$	
hydraulic resistance	R_h	$= 0.000000e+00$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 3.243569e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 9.503656e+05$	J
initially stored entropy	S_0	$= 3.243569e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 1.929151e+01$	$K\ W^{-1}$

Table B.36: Parameters of the radiator group mixing valve.

For the valve of the Schieland Heating System is given that $p_0 = 10^5 Pa$, $K_1 = 0.1m^3\ s^{-1}$ and $n = 3$. With these values K_0 can be determined: $4.978706838 \times 10^{-3} m^3\ s^{-1}$. The hydraulic resistance from the radiator group supply pipe to the radiator group pipe is exponential and controlled by $v = v_c$ whereas the resistance from the radiator group bypass pipe to the radiator group pipe is linear and has $v = 1 - v_c$. The radiator group control submodel defines v_c to be 0.2 for $t < 20s$ and 0.8 for $t \geq 20s$.

Radiator Group Pump The radiator group pump is modelled with the pump submodel, which assumes the pump to be ideal, i.e. there is no hydraulic resistance. The pump increases the pressure of the incoming water with $100 \times 10^3 Pa$. The heat capacity of a $0.3m$ long tube with $d_i = 0.1m$ (table B.37) serves as an of estimate of the heat capacity of the pump.

Radiator Group Pipe This is a $1m$ long straight pipe with $d_i = 0.05m$. The roughness is again estimated at 0.05. See the results in table B.38.

Radiator The parameters of the radiator are calculated with the information given by Ham in (Ham 1988). He lists the parameters of radiators he measured in experiments. The radiator in the Schieland system is much bigger than any of the radiators he mentions. Therefore, the parameters he measured for the radiator R2 are used here as the parameters for a radiator *segment*. Thus the values $m = 1.3$ and $K = 1391/60^{1.3} = 6.7878299$ are obtained.

Short cylindrical pipe			
geometry:			
length	l	$= 3.000000e-01$	m
inner diameter	d_i	$= 5.000000e-02$	m
flow area	A	$= 1.963495e-03$	m^2
volume	V	$= 5.890486e-04$	m^3
hydro kinetic:			
mass	m	$= 5.819800e-01$	kg
inertance	I	$= 1.509553e+05$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 0.000000e+00$	
hydraulic resistance	R_h	$= 0.000000e+00$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 2.432677e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 7.127742e+05$	J
initially stored entropy	S_0	$= 2.432677e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 2.314981e+02$	$K\ W^{-1}$

Table B.37: Parameters of the radiator group pump.

Long cylindrical pipe			
geometry:			
length	l	$= 1.000000e+00$	m
inner diameter	d_i	$= 5.000000e-02$	m
flow area	A	$= 1.963495e-03$	m^2
volume	V	$= 1.963495e-03$	m^3
hydro kinetic:			
mass	m	$= 1.939933e+00$	kg
inertance	I	$= 5.031843e+05$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 7.694675e-01$	
hydraulic resistance	R_h	$= 1.971912e+09$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+06$	J
initially stored entropy	S_0	$= 8.108922e+03$	$J\ K^{-1}$
thermal resistance	R_t	$= 7.716603e+02$	$K\ W^{-1}$

Table B.38: Parameters of the radiator group pipe.

The heat capacities and hydraulic resistances are determined as follows. From (Ham 1988) we know that each segment contains $4.2l$ water. This implies that the water flow area is $V/l = 4.2 \times 10^{-3}/0.96 = 0.004375\,m^2$. Assuming that the water flow area is rectangular, the depth of the radiator and the perimeter of the flow area can be determined: $d = A/h = 0.00546875m$ and $s = 2h + 2d = 1.6109375\,m$. This data can be fed into `ncpipe` (roughness 0.05) which then outputs the required parameters (table B.40).

For the two segments in the middle, the hydraulic resistance from table B.40 can be used. For the first and the last segments, the resistance caused by the in- and outflow has to be added. The inflow resistance is calculated with the `short` utility (table B.40) with $d_i = 0.05m$ and $\xi_s = 1$ according equation (6) on page Lc2 in (Verein Deutscher Ingenieure 1977). The outflow resistance is determined in the same way (table B.41) but with $\xi_s = 0.25$ from table 4 on page Lc1 in (Verein Deutscher Ingenieure 1977). Summarizing, the resistances of the segments are given in table B.39.

segment	$R_h[Pa^2\ m^{-6}]$
Radiator.Left.Seg_1	66039054800
Radiator.Left.Seg_2	65910920000
Radiator.Right.Seg_1	65910920000
Radiator.Right.Seg_2	65942953710

Table B.39: Summary of segment resistance parameters.

Long noncylindrical pipe			
geometry:			
length	l	$= 9.600000e-01$	m
flow area	A	$= 4.375000e-03$	m^2
perimeter flow area	s	$= 1.610937e+00$	m
hydraulic diameter	d_h	$= 1.086324e-02$	m
volume	V	$= 4.200000e-03$	m^3
hydro kinetic:			
mass	m	$= 4.149600e+00$	kg
inertance	I	$= 2.167954e+05$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 2.889846e+01$	
hydraulic resistance	R_h	$= 6.591092e+10$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 1.734533e+04$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 5.082181e+06$	J
initially stored entropy	S_0	$= 1.734533e+04$	$J\ K^{-1}$
thermal resistance	R_t	$= 3.324675e+02$	$K\ W^{-1}$

Table B.40: Parameters of a radiator segment (i).

Radiator Group Splitter The determination of the parameters for the radiator group splitter is analogous to the heater group splitter. Here, the splitter has one inflow with $d_i = 0.05m$, one outflow to the radiator group return pipe with $d_i = 0.1m$ and another outflow to the radiator bypass pipe,

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 5.000000e-02$	m
flow area	A	$= 1.963495e-03$	m^2
volume	V	$= 1.963495e-04$	m^3
hydro kinetic:			
mass	m	$= 1.939933e-01$	kg
inertance	I	$= 5.031843e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 1.000000e+00$	
hydraulic resistance	R_h	$= 1.281348e+08$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+02$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+05$	J
initially stored entropy	S_0	$= 8.108922e+02$	$J\ K^{-1}$
thermal resistance	R_t	$= 7.716603e+01$	$K\ W^{-1}$

Table B.41: Parameters of a radiator segment (ii).

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 5.000000e-02$	m
flow area	A	$= 1.963495e-03$	m^2
volume	V	$= 1.963495e-04$	m^3
hydro kinetic:			
mass	m	$= 1.939933e-01$	kg
inertance	I	$= 5.031843e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 2.500000e-01$	
hydraulic resistance	R_h	$= 3.203371e+07$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+02$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+05$	J
initially stored entropy	S_0	$= 8.108922e+02$	$J\ K^{-1}$
thermal resistance	R_t	$= 7.716603e+01$	$K\ W^{-1}$

Table B.42: Parameters of a radiator segment (iii).

also with $d_i = 0.1m$. It is modelled with a convection splitter model. To estimate the heat capacity of the splitter, the capacity of two pipes with $l = 0.1m$ and a sharp broadening from $d_1 = 0.05m$ to $d_2 = 0.1m$ in the middle (table B.43) is added to the capacity of a pipe with $l = 0.1m$ and $d_i = 0.05m$ (table B.44). This gives $2838.1222 J\ K^{-1}$ for the heat capacity and $831569.9J$ for the stored heat.

Short cylindrical pipe			
geometry:			
length	l	$= 1.000000e-01$	m
inner diameter	d_i	$= 5.000000e-02$	m
flow area	A	$= 1.963495e-03$	m^2
volume	V	$= 1.963495e-04$	m^3
hydro kinetic:			
mass	m	$= 1.939933e-01$	kg
inertance	I	$= 5.031843e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 1.500000e-02$	
hydraulic resistance	R_h	$= 1.922022e+06$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+02$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+05$	J
initially stored entropy	S_0	$= 8.108922e+02$	$J\ K^{-1}$
thermal resistance	R_t	$= 7.716603e+01$	$K\ W^{-1}$

Table B.43: Parameters of the radiator group splitter (i).

To determine the two hydraulic resistances of the splitter, the resistance of a splitter with $d_i = 0.05m$ is calculated with the `short` utility (table B.43). The Bernoulli splitting resistance factor is estimated with table 1 on page Lc3 in (Verein Deutscher Ingenieure 1977) at 0.015. The resistance parameter is added to the resistance of one of the sharp broadenings, obtained with the `broad` utility (table B.44). This gives $73997862 Pa^2\ m^{-6}$ for both hydraulic resistances.

Radiator Group Bypass Pipe This is a $25m$ long pipe with $d_i = 0.1m$ and 2 sharp bends each accounting for $\xi_s = 1.5$ (section 7.2, page Lc5 in (Verein Deutscher Ingenieure 1977)). The roughness is again estimated at 0.05. The parameters can be found in table B.45.

Radiator Group Return Pipe This is a $100m$ long pipe with $d_i = 0.1m$ and 9 sharp bends each accounting for $\xi_s = 1.5$ (section 7.2, page Lc5 in (Verein Deutscher Ingenieure 1977)). The roughness is again estimated at 0.05. See table B.46 for the parameter values.

B.6.5 Simulation

Two simulation runs of the Schieland hospital heating system will be presented here:

- Prediction of the hydraulic behaviour of the system when the pumps are switched on with valve control 0.2 which increases at $t = 20s$ to 0.8.
- Prediction of the thermodynamic behaviour of the system when the heater is switched on with valve control 0.8.

Sharply broadening short cylindrical pipe			
geometry:			
length narrow part	l_1	$= 5.000000e-02$	m
inner diameter narrow part	d_1	$= 5.000000e-02$	m
flow area narrow part	A_1	$= 1.963495e-03$	m^2
length broad part	l_2	$= 5.000000e-02$	m
inner diameter broad part	d_2	$= 1.000000e-01$	m
flow area broad part	A_2	$= 7.853982e-03$	m^2
volume	V	$= 4.908739e-04$	m^3
hydro kinetic:			
mass	m	$= 4.849834e-01$	kg
inertance	I	$= 3.144902e+04$	$kg\ m^4$
bernoulli resistance factor	ξ_s	$= 5.625000e-01$	
hydraulic resistance	R_h	$= 7.207584e+07$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 2.027230e+03$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 5.939785e+05$	J
initially stored entropy	S_0	$= 2.027230e+03$	$J\ K^{-1}$

Table B.44: Parameters of the radiator group splitter (ii).

Long cylindrical pipe			
geometry:			
length	l	$= 2.500000e+01$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 1.963495e-01$	m^3
hydro kinetic:			
mass	m	$= 1.939933e+02$	kg
inertance	I	$= 3.144902e+06$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 3.295139e-01$	
bernoulli resistance factor	ξ_s	$= 3.000000e+00$	
hydraulic resistance	R_h	$= 6.837473e+08$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 8.108922e+05$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 2.375914e+08$	J
initially stored entropy	S_0	$= 8.108922e+05$	$J\ K^{-1}$
thermal resistance	R_t	$= 4.822877e+03$	$K\ W^{-1}$

Table B.45: Parameters of the radiator group bypass pipe.

Long cylindrical pipe			
geometry:			
length	l	$= 1.000000e+02$	m
inner diameter	d_i	$= 1.000000e-01$	m
flow area	A	$= 7.853982e-03$	m^2
volume	V	$= 7.853982e-01$	m^3
hydro kinetic:			
mass	m	$= 7.759734e+02$	kg
inertance	I	$= 1.257961e+07$	$kg\ m^4$
roughness pipe surface	K	$= 5.000000e-02$	
resistance factor	ξ	$= 3.295139e-01$	
bernoulli resistance factor	ξ_s	$= 1.350000e+01$	
hydraulic resistance	R_h	$= 2.747002e+09$	$Pa^2\ m^{-6}$
hydro thermal:			
heat capacity	C	$= 3.243569e+06$	$J\ K^{-1}$
initial temperature	T_0	$= 2.930000e+02$	K
initially stored heat	Q_0	$= 9.503656e+08$	J
initially stored entropy	S_0	$= 3.243569e+06$	$J\ K^{-1}$
thermal resistance	R_t	$= 1.929151e+04$	$K\ W^{-1}$

Table B.46: Parameters of the radiator group return pipe.

Hydraulic behaviour

Figure B.9 shows a plot of some of the water flows in the system when it switched on with the valve at 0.2. At $t = 20s$ the valve control value is increased to 0.8. The initial water flows at $t = 0$ were set to zero.

The plot shows that the flow through the radiator and the radiator bypass pipe (D and E) reach their maximum almost directly. This is caused by the fact that the flow path from the RG-pump through the radiator and via the RG-bypass pipe back to the pump contains a relatively small amount of water. Because the RG-pump is the least powerful of the two and the flow path has a high hydraulic resistance, the two volume flows do not become very large.

Although the heater group pump is the most powerful pump, the flow through the heater, and the heater bypass pipe has a slow start because the flow path heater group pump, heater, heater bypass pipe, heater group pump contains a relatively large amount of water. Because of the high power of the pump and the fact that the flow path has a small hydraulic resistance, the flows become quite large.

The difference between flows A and B is exactly the water flow from the heater to the radiator group (and vice versa). This volume flow stays rather small due to the fact that with one radiator group, there is a great overcapacity. Flow C starts even slower than flows A and B because the radiator group supply and return pipes are very long and therefore contain large amounts of water.

What can be seen when the valve is opened from 20 to 80 percent at $t = 20$ is that the increasing flow through the radiator group supply pipe causes flows A and E to become smaller. It is interesting to see that the flows through the heater and the radiator do not change noticeably when the valve is opened.

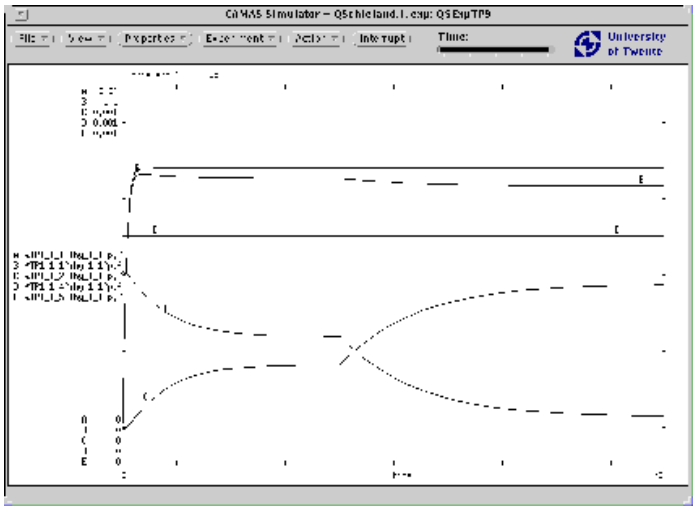


Figure B.9: Predicted hydraulic behaviour of the Schieland hospital heating system. The plotted values are water flows through A: the heater, B: the hg bypass pipe, C: the rg supply pipe, D: the rg pipe and E: the rg bypass pipe. Note that for A and B the vertical scale is 0 to $0.01\ m^3s^{-1}$ and for C, D and E 0 to $0.001\ m^3s^{-1}$. The horizontal axis is the time scale in seconds.

The results of the simulation are close to the results obtained by Andringa and Van der Laan in (Andringa and van der Laan 1991). The small numerical differences can be explained by the fact that their model is a simplification of the model used here. The differences are:

- The model described here models splitting/mixing resistances as well as hydraulic resistance caused by broadening and narrowing of flow paths. Both types of resistance are neglected in (Andringa and van der Laan 1991).
- The tables and formulas in (Verein Deutscher Ingenieure 1977) list slightly different characteristic values and produce slightly different parameter values than the ones used in (Andringa and van der Laan 1991).
- The dimensions of the radiator used here are different and a more accurate mathematical equation for the heat emission of the radiator is used.
- One of the controlled hydraulic resistances in the valve is linear (this is more realistic than two exponential valves, see (Recknagel and Sprenger 1979)) and an error in the relation of the exponential resistance in (Andringa and van der Laan 1991) has been corrected.

Thermodynamic Behaviour

The prediction of the thermodynamic behaviour can be found in Figure B.10. The initial temperatures were $293K$ and the initial water flow rates were zero. The control value of the valve was 0.8 throughout the simulation.

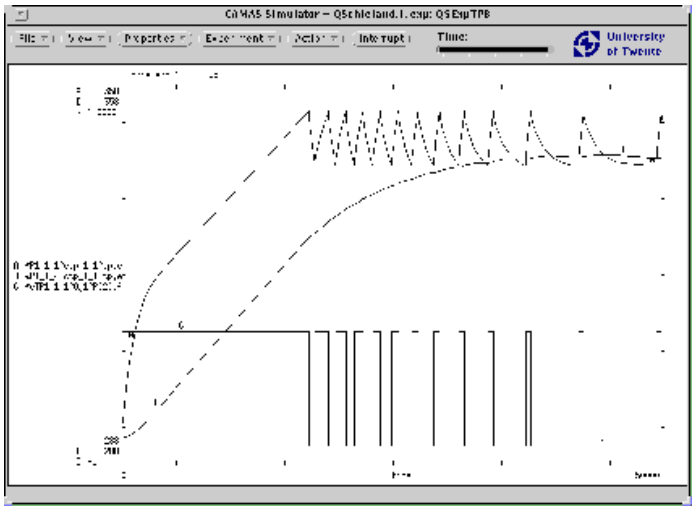


Figure B.10: Predicted thermodynamic behaviour of the Schieland hospital heating system. The plotted values are A: the temperature of the heater, B: the temperature of the radiator pipe, both in K. G is the heat flow from the heater which has $0\ W$ as minimum, and $60\ kW$ as maximum value. The horizontal axis is the time scale in seconds.

A notable result from the simulation is the big difference in time scales between the hydraulic and thermodynamic behaviour. While in the hydraulic case equilibrium is reached in about twenty seconds, in the thermodynamic case this time is more than twenty thousand seconds: more than five hours.

From $t = 0$, because the heater is on, the heater temperature A will increase. In the beginning it increases quickly because the water that flows into the heater is of almost the same temperature as the water that flows out of it. The net heat flow to the heater will then be approximately $60\ kW$. As the temperature of the heater increases, the amount of heat that flows out of the heater will become larger than the heat carried by the water that flows into it. The net heat flow to the heater will become smaller than $60\ KW$ and therefore the heater temperature will increase more slowly. When the radiator gets warm, the temperature of the water from the radiator group return pipe will increase and so will the temperature of the water that flows into the heater. This will cause the heater temperature to increase at a constant rate until the maximum heater temperature is reached and the heater is switched off.

Next, the heater will be switched on and off repeatedly. The simulation shows that the periods that the heater is on become shorter and that the on-off period becomes longer. This can be explained by the fact that the temperature of the water from the radiator group reaches a high value. Because of this, the net heat flow out of the heater will become smaller, so that it takes more time for the heater to cool down and less time to heat up again.

The simulation results show the same qualitative behaviour as can be seen in plots in (Andringa and van der Laan 1991). The numerical values produced do not seem to be the same, however. To explain this the model that is used in (Andringa and van der Laan 1991) needs to be discussed in more detail.

The model of Andringa and Van der Laan use is a purely thermodynamic model that is decoupled from the model that predicts the hydraulic behaviour. As a result of this, the water volume flows in the system

are free parameters in the thermodynamic model. Usually, this is a valid method to simplify the model; the time scale in the hydraulic part in the system is so much smaller than that of the thermodynamic model that the volume flows can be assumed to be constant. The problem however is that the numerical values for the volume flow parameters that Andringa and Van der Laan use, are not consistent with their own results from the hydraulic model. Furthermore, the heat capacities they use are different from what is to be expected from geometric data, i.e. different from what was computed by the `pipe`, `ncpipe` etc. utilities. Another point of criticism to the model in (Andringa and van der Laan 1991) is that some processes that could be of importance are neglected. An example of this is heat storage in the two bypass pipes.

These observations lead to the conclusion that the numerical results for the thermodynamic part of (Andringa and van der Laan 1991) are not valid and thus cannot be used for a comparison with our simulation results in Figure B.10. Nevertheless, the end temperature of the radiator group pipe appears realistic because the temperature of the water from the heater is approximately $348K$ and the ratio between the water volume flow from the heater and the radiator bypass pipe is $0.0005 : 0.0001 = 5 : 1$ (see Figure B.9).

B.6.6 Concluding Remarks

The remarks in this section can be put into three categories: first those that concern the thermodynamic library, second those that concern the OLMECO library in general, and third the remarks that concern the modelling tool that makes use of the library.

Thermodynamic Library The first conclusion that can be drawn from this experiment is that the thermodynamic library is diverse enough to support compositional modelling of real world systems. The modelled system is considered to be large and although it contained simplifications compared to the real world system, it is expected that a model without these simplifications can be constructed successfully because it will just have more components of the same types that were used here.

OLMECO **Library** The time it took to construct the model using the library components was short and the model turned out to be more accurate than the model in (Andringa and van der Laan 1991). The only step in the modelling process that took a lot of time was the determination of the model parameters. This activity is not difficult, but the amount of parameters that has to be determined is very large. Therefore we suggest an extension of the library in which it is possible to specify the way the parameters of a model component have to be determined. For the thermodynamic library, the needed parameter information base is given in Chapter B.6.4. The parameter relations in the library could then be used for automatic parameter computation from geometric data supplied by the user. The present way to store parameter relations in the library is not sufficient because the parameter relations that have to be used can depend on geometric aspects of the component that is modelled. For instance, cylindrical and non-cylindrical pipes are modelled with the same component and have the same equation for the hydraulic resistance, but the way to determine the parameters are different.

Furthermore, it would be nice to have a way to let the simulator check the validity domains of the submodels dynamically. The problem is that the validity of the model for hydraulic resistance, for instance, depend on dynamic model variables like the volume flows. This makes it impossible to check the validity before simulation. Changes in both the library and the simulator are required to incorporate dynamic checking. In the present library, specification of the validity domain for models are pure textual

annotations. To be able to store the more algorithmic checks like the one for hydraulic resistance this has to be changed. So we need an active form of management of model assumptions. The simulator must then be changed to be able to actually use these.

Modelling Tool In order to support modelling with the library optimally, the modelling tool must understand all concepts and distinctions in the library. Practically, this means that the modelling tool must be able to work with components, decompositions, processes and equations and that it must have a library browser to access the component taxonomy, the components itself, the decompositions, the process descriptions and the mathematical relations in the library. With such a browser, an index like the one in the Appendix will not be needed anymore.

Nonlinear mathematical equations cannot be inverted automatically by the present model processor although in many cases inversion is theoretically possible. An example of this is the relation for hydraulic resistance. As a result, the model processor is not able to choose one of the causal forms freely. This means that when causality cannot be assigned to the model, the user must replace some of the equations for hydraulic resistance by their inverse relations and try again. For large models this is very tedious. A solution would be to make the inversion algorithm used by the model processor more powerful. Another option is to make it possible to specify all causal forms of an equation in the library. The model processor could then, instead of trying to invert the equations themselves, just choose one of the causal forms.

The selection of physical processes that are modelled is very important. Selection of the wrong processes or disregarding important processes can lead to models in which causality cannot be assigned. This makes simulation impossible. The problem with the current model processor and simulator is that in these cases they do not offer facilities to indicate what the problem is and to what part of the model it refers. They just generates error messages that causality cannot be assigned, the model is algebraic and cannot be solved or that the matrix is ill-conditioned. These error messages do not give a clue as to what might be wrong with the model. In a future modelling tool, it would be nice if the messages were stated in terms of model components or physical processes. This is a topic for further research.

Bibliography

Addanki, S., R. Cremonini, and J. S. Penberthy (1991). Graphs of models. *Artificial Intelligence 51*, 145–177.

Akkermans, J. M., W. N. Borst, A. Pos, and J. L. Top (1995). Experiences in conceptual modelling for a mechatronic design library. In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the Ninth International Knowledge Acquisition Workshop KAW'95*, Volume 2, pp. 39.1–39.15. University of Calgary, SRDG Publications.

Alberts, M. (1993). *YMIR: an ontology for engineering design*. Ph. D. thesis, University of Twente.

Andringa, R. and E. J. van der Laan (1991). Bondgraaftechniek bij verwarmingsinstallaties. Technical report, Hogeschool Rotterdam e.o., Polytechnical Faculty at Dordrecht. Thesis, in Dutch.

Angele, J., S. Decker, R. Perkuhn, and R. Struder (1996, November 9–14). Modelling problem-solving methods in new KARL. In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'96 (Volume 1)*, Banff, Alberta, Canada, pp. 1.1–1.18.

Bateman, J., B. Magnini, and F. Rinaldi (1994, 8–12 August). The generalized upper model. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 35–45. ECCAI.

Beaman, J. J. and R. C. Rosenberg (1988). Constitutive and modulation structure in bond graph modelling. *Trans. ASME J. Dyn. Syst. Meas. Control 110*(4), 395–402.

Benjamin, J., W. N. Borst, J. M. Akkermans, and B. J. Wielinga (1996). Ontology construction for technical domains. In N. Shadbolt (Ed.), *Proceedings 9th European Knowledge Acquisition Workshop EKAW'96*, Berlin, pp. 98–114. Springer-Verlag. Lecture Notes in Artificial Intelligence No. 1076.

Bernaras, A. and I. Laresgoiti (1996). Building and reusing ontologies for electrical network applications. In W. Wahlster (Ed.), *Proceedings 12th European Conference on Artificial Intelligence ECAI'96 (Budapest, Hungary)*, Chichester, UK, pp. 298–302. John Wiley.

Bolc, L. and J. Cytowski (1992). *Search Methods for Artificial Intelligence*. London: Academic Press. ISBN 0-12-111240-3.

Borgo, S., N. Guarino, and C. Masolo (1996a). A pointless theory of space based on strong connection and congruence. In *Proceedings of Principles of Knowledge Representation and Reasoning (KR96)*, Boston, Massachusetts, pp. 220–229. Morgan Kaufmann.

Borgo, S., N. Guarino, and C. Masolo (1996b). Stratified ontologies: The case of physical objects. In *Proceedings of the ECAI-96 Workshop on Ontological Engineering*, Budapest, Hungary.

Borst, W. N. and J. M. Akkermans (1997, April). Disassembly analysis for LCA using PROMOD. Technical report, University of Twente and Netherlands Energy Research Foundation, Enschede, the Netherlands.

Borst, W. N., J. M. Akkermans, A. Pos, and J. L. Top (1995, May 16–19,). The PhysSys ontology for physical systems. In B. Bredeweg (Ed.), *Working Papers of the Ninth International Workshop on Qualitative Reasoning QR'95*, pp. 11–21. University of Amsterdam.

Borst, W. N., J. M. Akkermans, and J. L. Top (1996, November 9–14). Engineering ontologies (short version). In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'96 (Volume 1)*, Banff, Alberta, Canada, pp. 27.1–27.20.

Borst, W. N., J. M. Akkermans, and J. L. Top (1997). Engineering ontologies. *International Journal of Human-Computer Studies 46*, 365–406. Special Issue on Using Explicit Ontologies in KBS Development.

Borst, W. N., J. Benjamin, B. J. Wielinga, and J. M. Akkermans (1996a, August 11–16). An application of ontology construction. In P. E. van der Vet (Ed.), *Working papers of the Ontological Engineering Workshop W22 of the European Conference on Artificial Intelligence ECAI'96*, Budapest, Hungary, pp. 17–28.

Borst, W. N., J. Benjamin, B. J. Wielinga, and J. M. Akkermans (1996b, November 21–22). An application of ontology construction. In J.-J. C. Meyer and L. C. van der Gaag (Eds.), *Proceedings of the 8th Dutch Conference on Artificial Intelligence NAIC'96*, Utrecht, The Netherlands, pp. 47–60. Utrecht University.

Borst, W. N., A. Pos, J. L. Top, and J. M. Akkermans (1994, 8–12 August). Physical systems ontology. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 47–80. ECCAI.

Borst, W. N., J. L. Top, and J. M. Akkermans (1997). The THERMLIB library of generic thermodynamic model components. In J. J. Granda and G. Dauphin-Tanguy (Eds.), *Proceedings of the 1997 International Conference on Bond Graph Modeling and Simulation (ICBGM'97)*, San Diego, California. The Society of Computer Simulation International. Simulation Series, Volume 29, Number 1, ISBN 1-56555-103-6.

Breedveld, P. C. (1984). *Physical Systems Theory in Terms of Bond Graphs*. Ph. D. thesis, University of Twente, Enschede.

Buchanan, B. G. and E. A. Feigenbaum (1978). DENDRAL and METADENDRAL: Their applications dimension. *Journal of Artificial Intelligence 11*, 5–24.

Burghout, W., J. M. Akkermans, W. N. Borst, and A. Pos (1997). Linking computer aided design to product ecology assessment. Sustain deliverable, Netherlands Energy Research Foundation (ECN), Petten, The Netherlands.

Bylander, T. and B. Chandrasekaran (1988). Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In B. R. Gaines and J. Boose (Eds.), *Knowledge Acquisition for Knowledge Based Systems*, Volume 1, pp. 65–77. London: Academic Press.

Chandrasekaran, B. (1988). Generic tasks as building blocks for knowledge-based syatems: The diagnosis and routine design examples. *The Knowledge Engineering Review 3*, 183–210.

Clarke, B. L. (1981). A calculus of individuals based on `connection' . *Notre Dame Journal of Formal Logic 22*(3), 204–218.

Coad, P. (1992, September). Object-oriented patterns. *Communications of the ACM 35*(9), 152–159.

Cohn, A. G., D. A. Randell, and Z. Cui (1995). Taxonomies of logically defined qualitative spatial relations. *International Journal of Human-Computer Studies 43*, 831–846.

Davis, E. (1990). *Representations of Commonsense Knowledge*. San Mateo, California: Morgan Kaufmann Publishers Inc. ISBN 1-55860-033-7.

de Vries, T. J. A., P. C. Breedveld, and P. Meindertsma (1993). Polymorphic modelling of engineering systems. In *Proceedings of the ICBGM'93, San Diego*, pp. 17–22. SCS.

den Ouden, A. C. B., J. L. Top, and J. M. Akkermans (1994, March). EBIB: Dynamic model components for a heating systems library. Technical Report ECN-C–94-027, Netherlands Energy Research Foundation ECN, Petten, The Netherlands.

Eschenbach, C. and W. Heydrich (1995). Classical mereology and restricted domains. *International Journal of Human-Computer Studies 43*, 723–740.

Falasconi, S. and M. Stefanelli (1994, 8–12 August). A library of medical ontologies. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 81–91. ECCAI.

Falkenhainer, B., A. Farquar, D. Bobrow, R. E. Fikes, K. D. Forbus, T. R. Gruber, Y. Iwasaki, and B. Kuipers (1994, January). CML: A compositional modelling language. Technical Report Technical report KSL-94-16, Stanford Knowledge Systems Laboratory.

Falkenhainer, B. and K. D. Forbus (1991). Compositional modeling: Finding the right model for the job. *Artificial Intelligence 51*, 95–143.

Farquar, A., R. E. Fikes, and J. Rice (1996). The Ontolingua server: A tool for collaborative ontology construction. In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems Workshop, Volume 2*, University of Calgary, pp. 44–1–44–19. SRDG Publications.

Fazio, T. L. D. and D. E. Whitney (1987). Simplified generation of all mechanical assembly sequences. *IEEE Journal of Robotics and Automation RA-3*(6), 640–658.

Fensel, D., A. Schönegge, R. Groenboom, and B. J. Wielinga (1996, November 9–14). Specification and verivication of knowledge-based systems. In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'96 (Volume 1)*, Banff, Alberta, Canada, pp. 4.1–4.20.

Fiksel, J. (1996). *Design For Environment: Creating Eco-Efficient Products and Processes*. New York: McGraw-Hill, Inc.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence 24*, 85–168.

Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Menlo Park, California: Addison-Wesly. ISBN 0-201-89542-0.

Fox, M. S., J. Chionglo, and F. Fadel (1993). A common-sense model of the enterprise. In *Proceedings of the Industrial Engineering Research Conference 1993*.

Galton, A. (1996). Taking dimension seriously in qualitative spatial reasoning. In W. Wahlster (Ed.), *Proceedings 12th European Conference on Artificial Intelligence ECAI'96 (Budapest, Hungary)*, Chichester, UK, pp. 501–505. John Wiley.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesly. ISBN 0-201-63361-2.

Genesereth, M. R. (1990). The Epikit manual. Technical report, Epistemics Inc., Palo Alto, California.

Genesereth, M. R. and R. E. Fikes (1992, June). Knowledge interchange format, version 3.0 reference manual. Technical report logic–92–1, Computer Science Department, Stanford University.

Gennari, J. H., S. W. Tu, T. E. Rothenfluh, and M. A. Musen (1994). Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies 41*, 399–424.

Gerstl, P. and S. Pribbenow (1995). Midwinters, end games and body parts: A classification of part-whole relations. *International Journal of Human-Computer Studies 43*, 865–889.

Gruber, T. R. (1992). Ontolingua: A mechanism to support portable ontologies, version 3.0. Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, California.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, 199–220.

Gruber, T. R. (1994). Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli (Eds.), *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies 43*, 907–928.

Gruninger, M. and M. S. Fox (1994, 8–12 August). The design and evaluation of ontologies for enterprise engineering. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 105–128. ECCAI.

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies 43*, 625 – 640.

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

Ham, P. J. (1988, May). Dynamisch warmtegedrag van radiatoren en convectoren — metingen en modelvorming. *Verwarming en Ventilatie*, 315–334. In Dutch.

Hayes, P. J. (1985). Naive physics i: Ontology for liquids. In J. R. Hobbs and R. C. Moore (Eds.), *Formal Theories of the Commonsense World*, pp. 71–107. Norwood, New Jersey: Ablex Publishing Corporation. ISBN 0-89391-213-1.

Hobbs, J. R. (1995). Sketch of an ontology underlying the way we talk about the world. *International Journal of Human-Computer Studies 43*, 819–830.

Hobbs, J. R. and R. C. Moore (Eds.) (1985). *Formal Theories of the Commonsense World*. Norwood, New Jersey: Ablex Publishing Corporation. ISBN 0-89391-213-1.

Horacek, H. (1994, 8–12 August). A model for financial investment in a natural language consultation system. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 129–140. ECCAI.

Ishii, K. (1996, November). Material selection issues in design for recyclability. In *Proceedings of The Second International Conference on EcoBalance*, Tsukuba, Japan, pp. 435–440.

Ishii, K., C. F. Eubanks, and M. Marks (1993). Evaluation methodology for post-manufacturing issues in life-cycle design. *International Journal of Concurrent Engineering: Research and Applications 1*, 61–68.

Ishii, K. and B. H. Lee (1996, August). Reverse fishbone diagram: A tool in aid of design for product retirement. In *ASME Design for Manufacturability Conference*, Irvine, California. 96-DETC/DFM-1272, ASME DTC/CIE Proceedings CD, ISBN 0-7918-1232-4.

Ishii, K., B. H. Lee, and C. F. Eubanks (1995). Design for product retirement and modularity based on technology life cycle. In *Proceedings of the Winter Annual Meeting Symposium on Life Cycle Engineering*.

Karnopp, D. C. (1978). Pseudo bond graphs for thermal energy transport. *Trans. ASME J. Dyn. Syst. Meas. Control 100*(3), 165–169.

Karnopp, D. C. (1979). State variables and pseudo bond graphs for compressible thermofluid systems. *Trans. ASME J. Dyn. Syst. Meas. Control 101*(3), 201–204.

Karnopp, D. C., D. L. Margolis, and R. C. Rosenberg (1990). *System Dynamics: A Unified Approach*. New York: John Wiley & Sons. Second Revised Edition.

Kernighan, B. W. and D. M. Ritchie (1988). *The C programming language*. Englewood Cliffs, NJ: Prentice-Hall. Second edition, ISBN 0-13-110370-9.

Khosla, P. K. and R. Mattikali (1989). Determining the assembly sequence from a 3-d model. *Journal of Mechanical Working Technology 20*, 153–162.

Kühn, O. (1994, 8–12 August). An ontology for the conservation of corporate knowledge about crankshaft design. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 141–152. ECCAI.

Kuipers, B. (1994). *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, Massachusetts: The MIT Press.

Laresgoiti, I., A. Anjewierden, A. Bernaras, J. Corera, A. T. Schreiber, and B. J. Wielinga (1996, November 9–14). Ontologies as vehicles for reuse: A mini-experiment. In B. R. Gaines and M. A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'96 (Volume 1)*, Banff, Alberta, Canada.

Lenat, D. B. and R. V. Guha (1990). *Building large knowledge-based systems. Representation and inference in the Cyc project*. Reading, Massachusetts: Addison-Wesley.

Liu, Z.-Y. (1992). Integrating two ontologies for electronics. In B. Faltings and P. Struss (Eds.), *Recent Advances in Qualitative Physics*, pp. 153–168. Cambridge, Massachusetts: The MIT Press. ISBN 0-262-06142-2.

MacGregor, R. (1990). LOOM users manual. Isi/wp–22, USC/Information Sciences Institute.

Marcus, S. and J. McDermott (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence 39*(1), 1–38.

McDermott, J. (1989). Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems*, pp. 225–256. Boston, Masachusetts: Kluwer Academic Publishers.

Mowbray, T. J. and R. Zahavi (1995). *The Essential CORBA: System Integration Using Distributed Objects*. John Wiley and Object Management Group.

Naur, P. (1963). Revised report on the algorithmic language ALGOL 60. *Communications of the ACM 6*(1), 1–17.

Neches, R., R. E. Fikes, T. Finin, T. R. Gruber, T. Senator, and W. R. Swartout (1991). Enabling technology for knowledge sharing. *AI Magazine 12*(3), 36–56.

Newell, A. (1982). The knowledge level. *Artificial Intelligence 18*, 87–127.

Partridge, C. (1996). *Business Objects*. Oxford, United Kingdom: Butterworth-Heinemann. ISBN 07506-2082-X.

Pos, A. (1997). *Automated Redesign of Engineering Models*. Ph. D. thesis, University of Twente, Enschede, The Netherlands. ISBN 90-365-0960-2.

Pos, A., J. M. Akkermans, and J. L. Top (1996). Automated model revision. Submitted to IEEE Expert.

Pos, A., W. N. Borst, J. L. Top, and J. M. Akkermans (1996, April). Reusability of simulation models. *Knowledge Based Systems Journal 9*(2), 119–125.

Puerta, A. R., J. W. Egar, S. W. Tu, and M. A. Musen (1992). A multiple-method shell knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition 4*, 171–196.

Quine, W. O. (1961). *From a Logical Point of View, Nine Logico-Philosophical Essays*. Cambridge, Massachusetts: Harvard University Press.

Randell, D. A. and A. G. Cohn (1992). A spatial logic based on regions and connections. In B. Nebel, C. Rich, and W. R. Swartout (Eds.), *Proceedings of the third National Conference on Principles of Knowledge Representation and Reasoning.*, Los Altos, pp. 165–176. Morgan Kaufmann.

Randell, D. A., A. G. Cohn, and Z. Cui (1992). Naive topology: Modeling the force pump. In B. Faltings and P. Struss (Eds.), *Recent Advances in Qualitative Physics*, pp. 177–192. Cambridge, Massachusetts: The MIT Press. ISBN 0-262-06142-2.

Recknagel, H. and E. Sprenger (1979). *Taschenbuch für Heizung und Klimatechnik*. München: Oldenbourg. In German.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991). *Object-Oriented Modelling and Design*. Englewood Cliffs, New Jersey: Prentice Hall.

Sablayrolles, P. (1993, March). A two-level semantics for french expression of motion. DFKI research report RR-93-12, DFKI, Saarbrücken.

Schreiber, A. T. and P. Terpstra (1996). Sysyphus VT: A CommonKADS solution. *International Journal on Human-Computer Studies 44*, 373–402.

Schreiber, A. T., B. J. Wielinga, J. M. Akkermans, W. V. de Velde, and R. de Hoog (1994, December). CommonKADS — A comprehensive methodology for KBS development. *IEEE Expert 9*(6), 28–37.

Schreiber, A. T., B. J. Wielinga, J. M. Akkermans, W. van de Velde, and A. Anjewierden (1994). CML: The CommonKADS conceptual modelling language. In L. Steels, A. T. Schreiber, and W. van de Velde (Eds.), *A Future for Knowledge Acquisition. Proceedings of the 8th European Knowledge Acquisition Workshop EKAW'94*, Berlin/Heidelberg, pp. 1–25. Springer-Verlag.

Schreiber, A. T., B. J. Wielinga, and W. J. Jansweijer (1995, August 19–20,). The KACTUS View on the `O'Word. In D. Skuce (Ed.), *Working Papers IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Quebec, Canada. IJCAI.

Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*. North-Holland: Elsevier.

Simons, P. (1987). *Parts, A Study in Ontology*, Chapter 1–3, pp. 5–128. Oxford: Clarendon Press.

Skuce, D. (1993). A Review of `Building Large Knowledge-Based Systems' by D.B. Lenat and R.V. Guha. *Artificial Intelligence 61*, 81–94.

Skvarcius, R. and W. B. Robinson (1986). *Discrete Mathematics with Computer Science Applications*. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc. ISBN 0–8053–7044–7.

Sowa, J. F. (1995). Top-level ontological categories. *International Journal of Human-Computer Studies 43*, 669–685.

Steele Jr., G. L. (1990). *Common Lisp: the language*. Bedford, Massachusetts: Digital Press. Second edition, ISBN 1-55558-041-6.

Sturges Jr, R. H. and M. I. Kilani (1992, february). Towards an integrated design for an assembly evaluation and reasoning system. *Computer-aided design 24*(2), 67–79.

Takagaki, K. and Y. Wand (1991). An object-oriented information systems model based on ontology. In F. van Assche, B. Moulin, and C. Rolland (Eds.), *Object Oriented Approach in Information Systems*, pp. 275–296. North Holland: Elsevier Science Publishers B.V.

Top, J. L. and J. M. Akkermans (1994, December). Tasks and ontologies in engineering modelling. *International Journal of Human-Computer Studies 41*(4), 585–617.

Top, J. L., W. N. Borst, and J. M. Akkermans (1995, November). Reusable thermodynamic model components for design. OLMECO deliverable, ESPRIT-III project 6521 OLMECO/WP2T45/ECN/01/4.0, ECN and University of Twente.

Top, J. L., A. P. J. Breunese, J. F. Broenink, and J. M. Akkermans (1995, 15–18 January). Structure and use of a library for physical systems models. In *Proceedings International Conference on Bond Graph Modelling and Simulation ICBGM'95*, Las Vegas, pp. 97–102. SCS.

Top, J. L., A. P. J. Breunese, J. van Dijk, J. F. Broenink, and J. M. Akkermans (1994). Conceptual schema of the OLMECO library. OLMECO deliverable, ESPRIT project 6521 OLMECO/WP3.3/ECN/01/2.0, ECN and University of Twente.

Tu, S. W., H. Erikson, J. H. Genari, Y. Shahar, and M. A. Musen (1995). Ontology-based configuration of problem-solving methods and generation of knowledge acquisition tools: The application of PROTÉGÉ-II to protocol-based decsision support. *Artificial Intelligence in Medicine*.

Umeda, Y., T. Tomiyama, T. Kiriyama, and Y. Baba (1995). The green browser: A proposal of green information sharing and a life cycle design tool. In F.-L. Krause and H. Jansen (Eds.), *Life Cycle Modelling for Innovative Products and Processes*, pp. 106–115. London: Chapman & Hall.

Ushold, M. F. (1992). The use of domain information and higher order logic for model management. In Holsapple and Whinston (Eds.), *Recent Developments in Decision Support Systems*.

van der Vet, P. E., P.-H. Speel, and N. J. I. Mars (1994, 8–12 August). The plinius ontology of ceramic materials. In N. J. I. Mars (Ed.), *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, Amsterdam, pp. 187–206. ECCAI.

van der Vet, P. E., P.-H. Speel, and N. J. I. Mars (1995). Ontologies for very large knowledge bases in materials science: A case study. In N. J. I. Mars (Ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing 1995*, Amsterdam, NL, pp. 73 – 83. IOS Press.

van Heijst, G., A. T. Schreiber, and B. J. Wielinga (1997). Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies 46*, 365–406.

Verein Deutscher Ingenieure (1977). *VDI Wärme-Atlas — Berechnungsblätter für den Wärmeübergang*. Düsseldorf: VDI-Verlag GmbH. Dritte durchgesehene Auflage. In German.

Wand, Y. and R. Weber (1990, November). An ontological model of an information system. *IEEE Transactions on Software Engineering 16*(11), 1282–1292.

Warmer, C. (1997, April). Process models for lca. Technical report, Netherlands Energy Research Foundation, Petten, the Netherlands.

Wilson, B. H. and J. L. Stein (1992). An algorithm for obtaining minimim-order models of distributed and discrete systems. In *Proc. ASME 1992 Winter Annual Meeting, DSC-Vol.41, Automated Modeling*, pp. 47–58.

Wilson, R. H. and J.-C. Latombe (1994). Geometric reasoning about mechanical assembly. *Artificial Intelligence 71*, 371–396.

Winston, M., R. Chaffin, and D. Herrmann (1987). A taxonomy of part–whole relations. *Cognitive Science 11*, 417–444.

Zeiler, W. (1994, March). Een object-georiënteerde bibliotheek van fysische standaardmodellen voor energie-installaties. Technical report, Kropman B.V., Rijswijk, The Netherlands. In Dutch.