AUTOMATED REDESIGN
OF
ENGINEERING MODELS

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

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Pos, Anita

Automated Redesign of Engineering Models /
Anita Pos. - [S.I. : s.n.]. - Ill.
Thesis Enschede. - With ref. - With summary in Dutch.
ISBN 90-365-0960-2
Subject headings: knowledge based systems / engineering design.

AUTOMATED REDESIGN
OF
ENGINEERING MODELS

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 voor Promoties
in het openbaar te verdedigen
op vrijdag 20 juni 1997 te 13.15 uur.

door
Anita Pos
geboren op 18 december 1968
te Utrecht

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

Contents

Samenvatting

Het construeren van (reken-)modellen is een belangrijke taak in technisch ontwerpen. Bijvoorbeeld, voordat een brug gebouwd wordt, wordt er eerst een model van deze brug gemaakt om te berekenen of de constructie sterk genoeg is om het voorgeschreven gewicht te kunnen dragen. Zo worden modellen in technisch ontwerpen gebruikt om verschillende aspecten van het gedrag en de structuur van een systeem te voorspellen of te verklaren. Een model is altijd een abstractie van het echte systeem, waarbij alleen eigenschappen die relevant zijn voor de huidige analyse worden meegenomen. Bijvoorbeeld, om de sterkte van een brug te berekenen is de kleur van de brug niet relevant, maar wel het materiaal waar de brug van gemaakt is.

Het precies bepalen welke aspecten en eigenschappen van belang zijn in een gegeven situatie is vaak een van de moeilijkste aspecten in een modelvormingsproces. In de meeste technische toepassingen wordt modelleren gezien als een repeterende taak: een model wordt voorgesteld, getest en aangepast tot het voldoet aan de behoeften van de modelleerder. Dit wil zeggen dat een belangrijk deel van modelleren bestaat uit *model-revisie*. Wij denken dat automatisering van het complete modelvormingsproces niet de beste manier is om het iteratieve karakter van modelvorming te ondersteunen. We denken echter wel dat een aantal verschillende modelleertaken *ondersteund* kunnen worden door gebruik te maken van een intelligente modelleer-omgeving. Deze omgeving vult de modelleerder aan, en ondersteunt hem in zijn modelleertaak, maar probeert niet om de modelleerder volledig te vervangen door het complete modelleerproces te automatiseren. In dit proefschrift hebben we onderzocht hoe kennis-gebaseerde technieken en methoden uit de traditionele engineering-wereld gecombineerd kunnen worden voor het ontwerpen van een intelligente modelleer-omgeving die ondersteuning biedt aan het proces van *model-revisie*.

Deze modelleer-omgeving biedt ondersteuning bij de volgende taken in het modelvormingsproces:

- *Specificeren van eisen aan een model.* Er zijn veel verschillende redenen om een

model te ontwerpen. Deze ‘modelleer-doelen’ blijven vaak impliciet en informeel, maar hebben wel grote invloed op de eisen waaraan het uiteindelijke model zal moeten voldoen. Om deze eisen expliciet boven tafel te krijgen, biedt onze modelleer-omgeving automatische ondersteuning bij het formaliseren van (impliciete en informele) modelleer-doelen tot expliciete eisen aan het uiteindelijke model.

- *Automatisch testen en repareren van modellen.* Onze modelleer-omgeving biedt faciliteiten voor het automatisch testen van een model met betrekking tot een verzameling (expliciete) eisen. Tevens worden er suggesties gedaan voor bruikbare model-aanpassingen indien het model niet voldoet aan de eisen die er aan gesteld zijn.

- *Specificeren van modelleer-aannamen.* Het proces van modelvorming wordt eenvoudiger als er gebruik gemaakt kan worden van een bibliotheek met vaak gebruikte (deel)modellen. Een manier om te zorgen dat deze (deel)modellen makkelijker herbruikt kunnen worden is het expliciet opnemen van hun *context*. Dit kan gebeuren door het expliciet specificeren van de aannamen die aan het (deel)model ten grondslag liggen.

De hoofdstukken 2 en 3 bespreken literatuur relevant voor dit proefschrift. Hoofdstuk 2 bespreekt de huidige stand van zaken op het gebied van automatisch modelleren, en bespreekt in hoeverre bestaande benaderingen ondersteuning bieden aan het iteratieve karakter van het modelvormingsproces. Dit leidt tot de formulering van bovenstaande onderzoeksvraagstukken. Hoofdstuk 3 beschrijft de model-representatie die door dit hele proefschrift heen gebruikt zal worden om verschillende aspecten van de inhoud van technische modellen te representeren.

Hoofdstukken 4 tot en met 7 beschrijven de modelleer-omgeving zoals die ontworpen is in het kader van dit onderzoek. In hoofdstuk 4 beschrijven we hoe een semi-automatische ‘requirement assistant’ ondersteuning kan bieden aan de modelleerder bij het expliciet maken van zijn doelen, en het bepalen van de eisen aan het model die hieruit voortvloeien. Deze eisen vormen het startpunt voor het testen en aanpassen van modellen.
In hoofdstuk 5 ontwikkelen we een algemene theorie voor het testen en repareren van modellen van technische systemen. Hiertoe maken we gebruik van ‘reparatie-plannen’. Reparatie-plannen leggen een directe link tussen soorten problemen en de manier waarop deze opgelost kunnen worden, en zijn bovendien automatisch executeerbaar. Met behulp van deze reparatie-plannen kunnen zowel structurele als parametrische model-veranderingen automatisch worden uitgevoerd.
In hoofdstuk 6 presenteren we een voorbeeld van een geparametriseerd reparatie-plan.

Het doel van dit plan is de parameters in een model zodanig aan te passen tot het gedrag van dit model overeenkomt met bepaalde observaties aan het echte systeem. In technisch ontwerpen wordt dit ook wel systeem-identificatie genoemd. Om dit probleem op te lossen worden in onze aanpak traditionele, numerieke technieken gecombineerd met kennis-gebaseerde technieken. Hierdoor kan tijdens de identificatie gebruikt gemaakt worden van een abstractere beschrijving van geobserveerd gedrag.
In hoofdstuk 7 beschrijven we hoe aannamen die ten grondslag liggen aan (deel)modellen in een bibliotheek georganiseerd en gestructureerd kunnen worden. We laten tevens zien hoe deze structurering gebruikt kan worden om het specificeren van modelleer-aannamen en (deel)modellen semi-automatisch te ondersteunen.

Hoofdstuk 8 plaatst model-revisie in een bredere context. In dit hoofdstuk laten we zien dat model-revisie een speciaal geval is van de algemenere taak van herontwerp. We presenteren een kennis-gebaseerde analyse van herontwerp als taak, en laten zien dat herontwerp beschouwd kan worden als een familie van gerelateerde methoden die alle gebaseerd zijn op bepaalde gemeenschappelijke principes. We analyseren welke dimensies relevant zijn in het karakteriseren en vergelijken van verschillende herontwerp-methoden. Deze dimensies weerspiegelen belangrijke keuzes die gemaakt moeten worden in het ontwerpen van een kennis-gebaseerd systeem voor het herontwerpen van een willekeurig object. Deze keuzes, en hun invloed op het uiteindelijke systeem, worden geïllustreerd aan de hand van de keuzes die wij zelf gemaakt hebben bij de ontwikkeling van onze eigen modelleer-omgeving.

In hoofdstuk 9 worden de conclusies van dit onderzoek samengevat. Volledige automatisering van de modelleertaak is geen realistische. In dit proefschrift hebben we echter laten zien dat omgevingen voor modelleren en simulatie meer en intelligentere ondersteuning kunnen bieden dan tot nu toe het geval is.

Summary

Constructing *engineering models* is an important part of the design of technical systems. For example, before a bridge is actually built, a model is used to compute whether the construction will be able to carry a certain prespecified weight. In general, engineering models are employed during the design process to analyze different aspects of the behavior of a technical system, by providing predictions and explanations. Making engineering models is always concerned with making an abstraction of the system: only properties relevant to the task at hand are taken into account. For example, when analyzing the strength of a bridge, the color of the bridge is not important but the flexibility of its construction material is.

The difficult part in making a model is to determine precisely which aspects and properties of a system are relevant in the situation at hand. In most engineering applications, modeling is seen as an iterative task: a model is proposed, assessed and adapted until it satisfies the designers needs. This means that a large part of model construction consists of *model revision*. We think that automating the complete modeling process is not the best way to support the iterative nature of this process. However, a variety of different modeling tasks can be supported by making use of an *intelligent modeling environment*. Such an environment supplements and supports the modeler in his modeling task, but does not attempt to replace the human modeler by automating the complete modeling process. In this thesis we investigated how knowledge-based techniques and traditional engineering methods can be combined to design an intelligent modeling environment which supports this iterative *model revision* process.

This intelligent modeling environment provides the following functions:

- *Support for specification of modeling requirements.* User goals in engineering modeling are manifold, often implicit and not formally defined. Therefore, support is needed for clarifying and formalizing modeling goals, as well as for deriving explicit model requirements for each modeling goal.

- *Automated assessment and adaptation of engineering models.* Our environment provides facilities for *automated assessment* of an engineering model with respect to a set of (explicit) model requirements. It also provides suggestions for suitable *model repairs* if the current model does not satisfy these requirements.

- *Support for specification of modeling assumptions.* Engineering modeling can be supported by constructing a library of frequently used models and model fragments. However, constructing and maintaining such a library is not a simple task. One way to extend the re-usability of a library model is to explicitly specify the *context* in which it can be used, in the form of its underlying *modeling assumptions*. In our modeling environment, specification of models and model fragments is supported by organizing the modeling assumptions which underly the construction of a new model fragment.

Chapters 2 and 3 discusses literature relevant to this thesis. Chapter 2 looks at the state of the art in automated modeling, and discusses how well existing approaches support the iterative nature of the engineering modeling process. This discussion leads to the formulation of the research issues described above. Chapter 3 discusses the model representation used throughout this thesis to describe structural, physical and mathematical aspects of engineering models.

Chapters 4 to 7 discuss the modeling environment developed in this thesis. Chapter 4 proposes a semi-automated requirement assistant to guide the modeler in the clarification and formalization of his modeling goals. The result of this interactive process is a set of explicit model requirements. These model requirements form the starting point for model assessment. Chapter 5 outlines a general theory of assessment and adaptation by means of repair plans. Repair plans tie together the engineer's modeling expertise and the requirement knowledge in a convenient way. Thus, they provide a link between model-requirement discrepancies and generic solutions. With these repair plans both structural and parametric repairs can be performed automatically. Chapter 6 presents an example of a complex, parametric repair plan. In this approach, numerical and knowledge-based techniques are combined to perform a complex engineering task: parametric system identification. Finally, chapter 7 discusses how knowledge on modeling assumptions underlying model fragments can be organized and structured, and how specification of modeling assumptions and associated model fragments can be supported.

Chapter 8 positions model revision in a wider context. It shows that model revision is a special form of the more general task of redesign. We present a knowledge-level analysis

of redesign, and show that redesign can be viewed as a family of related methods based on some common principles. We also discuss relevant dimensions in characterizing different methods for redesign. These dimensions reflect relevant design decisions in constructing a knowledge based system for a redesign application. The modeling environment developed in this thesis is used to illustrate these dimensions.

In chapter 9 we present the conclusions of our research. Although full automation is not feasible, environments for engineering modeling and simulation can be made much more intelligent and supportive than they currently are.

Acknowledgments

This thesis describes the result of four years of research. During these years, a number of people contributed to this thesis in one way or another. Here, I would like to take the opportunity to thank them.

In the first place, I would like to thank my promotor Hans Akkermans, who initiated and supervised my research. Without his support this thesis would not be what it is now.

Furthermore, I would like to thank all members in the REVISE project, and especially the OIOs Niek Wijngaards and Remco Straatman, for our many valuable discussions.

Especially, I would like to thank the club of people who helped me by proofreading drafts of this thesis: Arno Breunese, Jan Top, my promotor Hans Akkermans and my husband Dirk-Jan Out. Your comments were invaluable.

Of course, there are a lot of other people who have in other ways contributed to this thesis. I would like to name just a few of them here: Jan-Erik Strömberg, for his comments on system identification, Ashok Goel, for our discussion on model revision and redesign, and Neil Smith, for our (email) discussion on bond graphs and identification.

Finally, I would like to thank my husband Dirk-Jan Out for his support during the long months of writing my thesis.

The research described in this thesis was financed by NWO/SION within project 612-322-316 “Evolutionary design in knowledge-based systems” (the REVISE-project). It has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge System.

Chapter 1

Introduction

1.1 Subject of this thesis

Constructing *physical models* is an important part of the engineering process. Physical modeling is concerned with making an abstraction of an observed system, taking into account only properties that are relevant to the task at hand. A physical model is an artificial structure that can be employed to predict, analyze and explain the behavior and structure of an engineering system. Physical models can be used in different engineering tasks: in design, alternative models can be used to test different variants of an intended design before actually building it. In diagnosis, a model can be used to explain the faulty behavior of a misbehaving system. In general, a model can be used to predict or explain the behavior of a system by filling it in for a specific situation and analyzing it to derive (aspects of) its behavior.

Model construction is a difficult task. Although the simulation and reasoning aspects, once a model is given, have been thoroughly investigated, the model construction aspects appear to be much less understood. There is a large gap between current AI approaches to physical modeling and engineering practice. In most engineering applications, modeling is seen as an iterative task: a model is proposed, assessed and adapted until it satisfies the (explicit and implicit) user needs. These user needs often evolve during the modeling process, along with the construction of the model itself. Most AI approaches to modeling aim at automating the entire modeling process by selection and construction from a predefined library of models or model fragments. Such an approach leaves the modeler with hardly any means to influence the outcome of the modeling task, and there are almost no facilities for tuning the model to the specific requirements that the modeler might have. This approach results from the idea that extensive specification of requirements this

early in the modeling process is not possible because the problem is not yet well understood. Although this stance seems suitable in learning situations, it seems too restricted for physical modeling in practical engineering situations. In engineering practice, a significant part of the requirements can be specified early in the modeling process and it is essential for a modeler to influence the outcome of the modeling process by specifying his (goal-specific) needs and demands.

In this thesis we stress that automating the complete modeling process, as is attempted in many (qualitative) modeling approaches in AI, is not the best way to support the iterative nature of the modeling process. However, a variety of different modeling tasks can be supported by making use of *intelligent modeling environments*. Intelligent modeling environments *support* the modeler in his modeling task, but do not attempt to automate the complete modeling process. As such, they are more useful in supporting the inherently iterative nature of engineering modeling. These modeling environments should, for example, aid the user in specifying model requirements, perform assessment and adaptation of physical models and provide library support for specification and verification of reusable model entities.

We have investigated how knowledge-based techniques and traditional engineering methods can be combined to design an intelligent modeling environment which supports the iterative nature of the modeling task. In this thesis, we present the results of this research.

1.2 Research goals

The research presented in this thesis is embedded in two research projects: REVISE and TWIST. Both projects have influenced the definition of research goals for this thesis.

REVISE is a research project in which four research groups participate: the Netherlands Energy Research Foundation (ECN), the University of Twente (UT), the University of Amsterdam (UvA) and the Vrije Universiteit van Amsterdam (VUA). The main theme of this project is *redesign in knowledge based systems*, and the overall goal of the project is to contribute to the development of a general design methodology for a new generation of knowledge-based systems. Within this project our focus is on reuse and redesign of computational models in engineering. Other objects of redesign studied in REVISE are: technical engineering systems (UT), control in knowledge-based systems (UvA) and compositional architectures of knowledge-based systems (VUA).

The research presented in this thesis is also embedded in the research of the TWIST-group at the university of Twente. The aim of TWIST is to enhance computer support for engineering tasks, in particular the model construction process, simulation and other forms of large-scale computational and mathematical analysis. A special research emphasis is on the utilization of ideas and methods from Artificial Intelligence for engineering applications, and on the integration of these techniques with conventional approaches and techniques in application fields.

The aim of the work reported here is to investigate how knowledge-based techniques and traditional engineering methods can be combined to support engineering modeling. We focus hereby on how an *intelligent modeling environment* can best support the inherently iterative nature of the engineering modeling process. To this end we distinguished a number of functions such a modeling environment should provide:

- *Assessment and adaptation.* To support the iterative nature of engineering modeling, an intelligent modeling environment should provide facilities for automated assessment of the model against a set of (explicit) requirements. It should also suggest suitable repairs if the model does not satisfy these requirements. These repairs should combine existing engineering techniques with a knowledge-based view on repair.
- *Support for specification of modeling requirements.* The specification of the demands a computational model should satisfy is not a trivial task. Current (qualitative) modeling approaches in AI mainly focus on model composition and do not pay much attention to the specification of user needs and demands. In contrast, user goals in engineering modeling are manifold, often implicit and often not formally defined. Therefore, support is needed for clarifying modeling goals and formulating explicit requirements based on these modeling goals.
- *Support for specification and verification of reusable model fragments.* The idea of supporting modeling by supplying libraries of reusable model fragments has been around for quite a while, both in AI and in various engineering disciplines. However, most of these approaches do not address the issue of how to construct and maintain such libraries. *Modeling assumptions* describe the context in which a model fragment is to be used. Specifying these assumptions explicitly extends the sharability and re-usability of a library of model fragments. *Support* for explicit specification and verification of modeling assumptions underlying the model fragments in a library supports the human modeler in keeping these assumptions consistent, explicit and complete.

1.3 Advances

To realize these functions, we develop a general framework for *model revision*. In particular, the work presented here leads to the following advances:

1. A theory for *automated* support for the *specification of modeling goals and requirements* is developed.
2. A general theory for *automated assessment and adaptation* by means of *generic repair plans* is developed. These repair plans include both structural and parametric repairs, and are able to solve a range of modeling problems.
3. We show how *automated support* can be useful in the context of *specification of modeling assumptions* and associated model fragments.
4. A general view on *redesign as a family of problem-solving methods* is developed, and *relevant dimensions* in characterizing this family of problem-solving methods are distinguished. We also show that model revision is a specific form of the more general task of redesign.

In order to validate and substantiate this framework, we implement a significant part of it in a knowledge-based system, called 007.

Parts of this work have been published in international fora - both in AI and in engineering - or have been submitted recently.

- (Pos & Akkermans, 1997a), to be presented at the Florida Artificial Intelligence Symposium (FLAIRS-97), discusses requirement specification in engineering modeling.
- Initial ideas on assessment and adaptation of engineering models have been presented at the Workshop on Models and Techniques for Reuse of Designs at the European Conference on Artificial Intelligence (Pos *et al.*, 1994), and in an article that appeared in the Knowledge-based Systems journal (Pos *et al.*, 1996a).
- A more recent version of these ideas has been presented at the European Simulation MultiConference (ESM'96) (Pos & Akkermans, 1996a).
- An extensive paper on model revision will appear in the journal IEEE Expert (Pos *et al.*, 1997b).

- (Pos & Akkermans, 1996b), presented at the Workshop on Modeling and Reasoning with Function at the US National Conference on Artificial Intelligence (AAAI-96), discusses model identification of physical models. A more recent version of this paper will be presented at the poster sessions of the International Joint Conference on Artificial Intelligence (IJCAI'97) in Japan (Pos & Akkermans, 1997b).
- A paper on redesign problem solving will be presented at the IJCAI'97 Workshop on Problem-solving Methods for Knowledge-based Systems (Pos *et al.*, 1997a). This paper has also been submitted to the European Knowledge Acquisition Workshop (EKAW'97).

1.4 Structure of the thesis

In chapter 2, we look at the state of the art in automated modeling, and discuss how the approaches discussed support the iterative nature of engineering modeling.

In chapter 3, the previous research on which this thesis builds is described. The model representation developed in (Top, 1993) is described in some detail. This model representation provides us with a rich framework for describing structural, physical and mathematical aspects of computational engineering models.

In chapter 4, our approach to providing support for the specification of modeling goals and explicit model requirements is discussed. A semi-automated requirements assistant is proposed to guide the modeler in the clarification and formalization of his modeling goals into a set of explicit model requirements.

In chapter 5, we discuss the tasks of model assessment and model adaptation. Existing techniques from the field of engineering are incorporated in a larger framework to support the (semi-)automated assessment and adaptation of physical models.

In chapter 6, we describe a specific technique for model adaptation, pertaining to the field of system identification. This technique is a good example of how a combination of AI and standard (mathematical) techniques can be used to solve a significant problem in mathematical modeling: fitting the behavior of a model to measured data.

Specifying and keeping track of modeling assumptions is essential in extending the sharability and re-usability of model fragments in a library. In chapter 7, we discuss how modeling assumptions can be structured, and how specification of modeling assumptions and associated model fragments can be supported.

In chapter 8, we extend our scope from model revision to the more general task of redesign. We present a general view of redesign as a family of methods based on some common principles, and we identify some important dimensions along which redesign approaches can differ.

In the final chapter, chapter 9, conclusions of the research described in this thesis as well as suggestions for further research are presented.

Chapter 2

Background of the research

This chapter describes the background of the research described in the rest of this thesis. First, a short summary of different types of model representations is presented. Then, an extensive survey is given of the state of the art in automated modeling approaches. In particular, we discuss how these approaches support the iterative nature of the engineering modeling process. This discussion leads to the formulation of a number of research issues that will be covered in the remainder of this thesis.

2.1 Introduction

During design, many different facets of an engineering system need to be analyzed. A technique

to obtain insight in the behavior of an engineering system is *physical modeling*. A physical model is an artificial structure that can be employed to predict, analyze and explain the behavior and structure of an engineering system. In physical modeling, many different types of models are used, ranging from geometric (CAD) models for describing and analyzing the topology of a system to finite element models for performing stress analysis.

In this thesis we concentrate on a small part of models and modeling techniques. We focus on automated support for *iterative design* of *dynamic* models for engineering systems.

- *Dynamic models for engineering systems.* During the design of engineering systems, many aspects of the system to be built need to be considered.

  Since each model describes only a limited number of aspects of a system, many dif-

ferent models are used in the design process. Of these, dynamic models are used to describe, analyze or explain the dynamic behavior of a system. Dynamic models can be used to answer questions about achievable performance of the system and to support the design of control systems for controlling the behavior of physical systems.

- *Iterative modeling.* There is a large gap between current AI approaches to automated modeling and current practices in engineering. In most engineering applications, modeling is seen as an iterative task: a model is proposed, assessed and adapted until it satisfies the (explicit and implicit) user needs. These user needs often evolve during the modeling process, along with the construction of the model itself. Most AI approaches aim at automating the model construction process entirely, leaving the modeler with almost no influence on the modeling process. In this chapter, we show that this approach to model construction does not sufficiently support the iterative nature of the engineering modeling process.

In this chapter we discuss the state of the art in automated modeling.

The chapter is organized as follows. First, in section 2.2 different representations of dynamic models are discussed. Then, section 2.3 describes approaches to (semi-)automated modeling. Three categories of approaches are distinguished: model *composition*, model *selection* and model *induction*. Each of these approaches is evaluated on how well it supports the iterative process of engineering modeling. This evaluation leads to the formulation of a number of research issues that will be covered in the remainder of this thesis, described in section 2.4.

2.2 Dynamic model representations

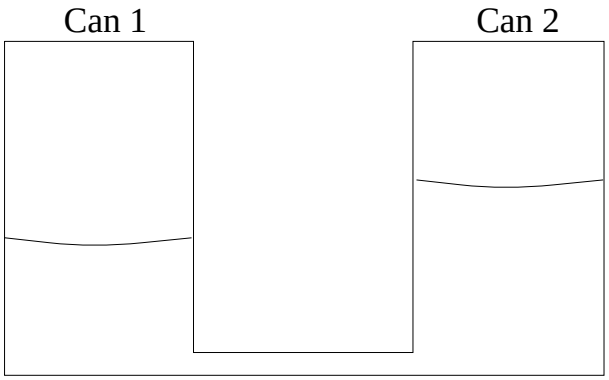
2.2.1 Quantitative models

Dynamic physical models can be represented in many different ways. Mathematical equations are the most common representation used for describing describing the behavior of an engineering system. Mathematical equations are familiar to all engineers, can be used to describe behavior independent of the systems domain and have a well-defined syntax. Furthermore, mathematical models can be used to answer many different questions on the numerical behavior of a system. However, representing dynamical models solely in terms of mathematical equations hides the physical background of the equations. Without this physical context it becomes very hard to provide explanations of how the system works.

Research in AI has tried to solve this problem in the field of Qualitative Reasoning, by abstracting away from mathematical equations to qualitative, non-numerical descriptions of physical systems. The basic concepts of qualitative models will be described in subsection 2.2.2. Subsection 2.2.3 describes models that combine both qualitative and quantitative features. The bond graph language, based on physical-energetic processes, is considered to be one of these. Bond graphs provide a domain-independent, consistent description of physical processes taking place in the system, and can be used for both qualitative and quantitative analysis (Top & Akkermans, 1991).

2.2.2 Qualitative models

The purpose of Qualitative Reasoning is to capture both common-sense knowledge that every one of us has about how things work, as well as the engineering principles used by engineers and scientists. Qualitative reasoning can be used in situations where precise mathematical equations are not available, too expensive to compute or not suitable for the task at hand, e.g. in explaining how a system works. Specific tasks in qualitative reasoning are predicting and explaining qualitative system behavior. Figure 2.1 presents some simple questions and answers qualitative reasoning can help to solve (examples taken from (Forbus & Falkenhainer, 1990)).



Q: What affects the water level in can 2?
A: The level of the water in can 2 is changing
as a function of the amount of water in can 2.

Q: How is the water level in can 2 changing?
A: The level of the water in can 2 is decreasing.

Figure 2.1: Some examples of qualitative reasoning with a two-container example.

Dynamic qualitative models represent time-varying aspects of the behavior of system variables and dependencies between these variables, both in a qualitative manner. The time-

varying behavior of system variables is usually represented as a sequence of qualitative states. Each qualitative state represents a set of variables with their value and their derivative. Qualitative models use symbolic values, instead of numerical values, to describe the value and derivative of system variables. Qualitative derivatives are usually described in terms like *decreasing, steady* and *increasing*, while qualitative values are described by a set of landmarks denoting relevant points and intervals, for example *negative, zero* or *positive*. In order to predict and/or explain the qualitative behavior of a system from a qualitative model an *inference engine*, appropriate for the representation chosen, is also a necessity. For a compilation of papers on principles of qualitative reasoning see (Weld & de Kleer, 1990).

Three important approaches can be distinguished in qualitative reasoning about physical processes, based on the primitive elements from which a qualitative model can be constructed (Bobrow, 1984): the device-centered approach (de Kleer & Brown, 1984), the process-centered approach (Forbus, 1984) and the constraint-centered approach (Kuipers, 1986). Each of these has a slightly different focus on representing and reasoning with qualitative information.

- The constraint-centered approach (Kuipers, 1986) is most strongly related to traditional mathematics. Qualitative differential equations are derived directly from ordinary differential equations, and *qualitative simulation* is used to derive the system behavior in terms of qualitative system states. QSIM (Kuipers, 1986) is a well-known example of an inference engine for qualitative simulation. QSIM produces a tree of all possible sequences of qualitative states, given a set of qualitative equations.

- The device-centered approach (de Kleer & Brown, 1984) has been strongly influenced by network ideas from electrical engineering. According to this approach, the behavior of a device can be constructed from its structure. The primitives are devices like pipes, valves and springs or resistors, capacitors and transistors. Network laws provide qualitative constraints at points in which the primitives connect. The possible behaviors of a system are derived by employing constraint-satisfaction techniques.

- The process-centered approach (Forbus, 1984) has physical processes as its modeling primitives. Processes are described in terms of *influences* indicating causal relationships between system variables. Also, the conditions under which a process is valid and/or applicable are represented explicitly. Given the set of applicable pro-

cesses, the qualitative history of a system can be generated by determining the influences that are active and by propagating their effects.

Each of these approaches has its own advantages and disadvantages, mainly in terms of the domains for which they are suitable. For example, (Forbus & Falkenhainer, 1990) indicate that the process-centered approach is most suitable in domains like thermodynamics and chemistry, while the device-centered approach works best when the idealizations supposed in network theory hold. No single qualitative approach is suitable for representing all physical domains used in engineering and science. Also, most qualitative approaches suffer a lot from ambiguity in their predictions; in many cases spurious behaviors are predicted together with the actual system behavior, and it is difficult to separate the true behavior from the spurious behaviors. This led to the introduction of *combined* models, incorporating both qualitative and quantitative information.

2.2.3 Combined qualitative-quantitative models

Interest in combining qualitative and quantitative aspects in model representations came from two sides: On one hand, the Qualitative Reasoning field needed to incorporate quantitative elements to reduce the qualitative ambiguity and to provide more accurate predictions of system behavior. On the other hand, approaches from traditional physics wanted to incorporate qualitative knowledge to supplement traditional methods in situations where they do not behave satisfactorily, to provide a more common-sense description of complex behavior, or to simplify the process of equation formulation.

Qualitative models have been extended by incorporating different forms of quantitative knowledge, like fuzzy values, restriction ranges for system variables and ordinary mathematical equations. For example, (Shen & Leitch, 1990) combine qualitative models with fuzzy values, and uses a combination of fuzzy logics and qualitative reasoning to generate more accurate predictions of system behavior. (Forbus & Falkenhainer, 1990) extend qualitative models by incorporating a library of quantitative model fragments. The resulting tutoring system uses both qualitative and quantitative knowledge to answer questions and provide explanations on numerical and qualitative aspects of system behavior. (Kuipers & Berleant, 1988) incorporate incomplete quantitative knowledge, in the form of ranges to which variables are restricted, in a qualitative model in order to refine qualitative predictions and exclude spurious behavior predictions.

One approach to integrating qualitative aspects with quantitative equations, originating

from physics, is given by the *bond graph* representation (Karnopp *et al.*, 1990), an energy-based representation which makes use of generic physical processes to describe system behavior. The *bond graph* language is based on the idea that abstract processes like *storage, transformation, transportation* are used in many physical domains. It uses *energy* as a unifying concept, since this concept remains invariant in different physical domains, and thus forms the basis for analogies across different domains. These correspondences among different domains are especially helpful for modeling *multidisciplinary* systems in which elements from different engineering domains are integrated to perform a specific function. For a more thorough description of the bond graph as a representation of physical processes, together with an example, see section 3.1.2.

In traditional engineering, bond graphs are typically used to generate the differential equations for numerical analysis and simulation (Karnopp *et al.*, 1990). However, bond graphs are more versatile than that: they can also be used to perform different forms of causal reasoning (Top & Akkermans, 1991), to generate qualitative equations for qualitative simulation (Xia *et al.*, 1992), or to generate typical aggregate behaviors such as ”decreasing oscillation” or ”exponential growth”(Top & Akkermans, 1991). An example of how bond graphs can be used to extend traditional methods is given in (Söderman & Strömberg, 1991), where bond graphs are used to extend a traditional system identification method from linear to piecewise-linear models.

In this thesis we will use a model representation based on bond graphs, described in chapter 3, to perform both qualitative and quantitative reasoning in the assessment and adaptation of dynamic physical models.

2.3 Automated modeling

Automated modeling in AI is based on the notion that the computer *solves* a modeling problem by itself, after the user has provided a problem statement. The process usually takes place without further human intervention. In automated modeling three different approaches can be distinguished: model *composition*, model *selection* and model *induction* approaches. The first two approaches either compose or select a model from a set of predefined (partial or complete) models, while the third kind of approach infers a model from behavioral data. Each of these approaches will be described in subsequent subsections (2.3.1 to 2.3.3). For this section, extensive use was made of two recent surveys on automated modeling: (Schut & Bredeweg, 1996) and (Xia & Smith, 1996).

2.3.1 Model composition

Model composition approaches construct a model by combining predefined model fragments.The human modeler specifies a scenario description, and usually some query to be answered by the system. A scenario description describes the system to be modeled in terms of components and their connections. The model construction system then uses a library of model fragments to construct a physical model. Figure 2.2 presents a graphical representation of model construction. In this subsection several approaches to model composition will be discussed: Compositional Modeling, extensions to Compositional Modeling and other approaches not directly related to the Compositional Modeling approach. For each approach, the user input and features of the algorithm used for model composition are discussed.

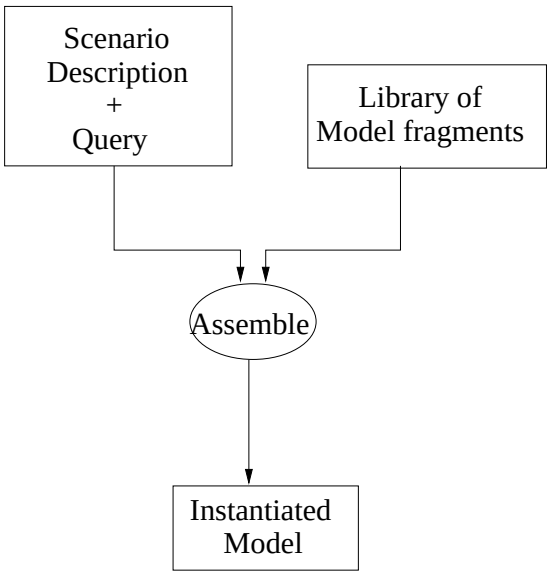


Figure 2.2: Model composition.

Compositional Modeling (CM). (Falkenhainer & Forbus, 1991) The Compositional Modeling approach is concerned with the construction of a qualitative model from a set of predefined partial models. All domain knowledge is represented in the form of model fragments. Each model fragment describes some piece of the domain physics, such as processes, devices and components. Each model fragment is also explicitly conditioned on a set of *modeling assumptions*. These assumptions are ordered in assumption classes. Each assumption class represents one dimension along which a modeling decision can be made, therefore all assumptions in a single assumption class are mutually exclusive.

The process of model composition consists of selecting and configuring a set of model fragments in response to a query and a scenario description, both given by the user. A query is a question about a specific aspect of the system that the model should answer, e.g. 'explain which influences affect the water level in can 2' (figure 2.1). A scenario description consists of a set of abstract components, describing the structure of the system to be modeled. After the user has provided the system with a query and a scenario description, the model construction process proceeds along the following lines: First, the scenario description is used to collect all model fragments corresponding to the abstract components in the scenario description. Second, *query analysis* is performed to identify a set of relevant objects, quantities and relations of interest. All these must be included in the final model. Third, *object expansion* is performed to make sure that all relevant components are included in the final model. The intuition behind the object expansion is that not only the objects resulting from the query analysis must be present in the final model, but that additional objects may need to be considered in order to capture all relevant interactions. Fourth, the *candidate completion* step is performed to find sets of consistent assumptions for each component. The final step consists of *candidate evaluation and selection*, i.e. each candidate model is evaluated to select the 'best' candidate.

Extensions to compositional modeling. Several extensions/modifications to the compositional modeling approach have been described in literature. These extensions differ not only in the algorithm used to derive a compositional model, but also in the amount of information to be input by the human modeler.

Nayak (Nayak *et al.*, 1992; Nayak & Joskowicz, 1996) extends the compositional modeling approach by automatically generating the *simplest* model for a system. Nayak formalizes the notion of model simplicity, and introduces a method for identifying the simplest model by using a class of modeling approximations called *causal approximations*. His approach to model composition needs extra information from the user to select the appropriate set of model fragments. This means that the scenario description has to be extended. The scenario description no longer consists only of abstract components, but needs also to include topological information and information in the expected model behavior. Topological information is represented in the form of typed connections between components (e.g. connected-to, coiled-around and meshed). The expected behavior of the model is expressed by means of causal relations between variables and threshold variables.

Nayaks model construction algorithm produces the simplest, adequate model consisting of both qualitative and quantitative equations by first identifying an adequate model, i.e.

a model satisfying the expected behavior of the system, and then simplifying it. An adequate model is identified in a number of steps: step 1 extends the original device description by including all expected behavior parameters specified by the user. The next steps augment the device model using the structural and behavioral constraints present in the scenario description. The last step checks the expected behavior. If the model does not satisfy the expected behavior, an extra model fragment for some component is added to the model, and the cycle starts again. When an adequate model has thus been constructed, the model is simplified by removing irrelevant model fragments or replacing them by one of their causal approximations.

Another extension to Compositional Modeling is described in (Rickel & Porter, 1992). Input to their modeling system TRIPEL is the same as for conventional CM: a scenario description, a query and a library of model fragments. However, TRIPELs model fragments are annotated with time scale conditions. Modeling in TRIPEL involves two issues: identifying the system boundary and selecting the proper time scale. Composing a model at the proper time-scale is based on finding a causal influence path from some given quantity to a quantity of interest, in which each influence operates at the same time scale.

Identifying the systems boundary, i.e. deciding which variables should be modeled as exogenous and which as endogenous, is then implemented by selecting all influences that are (in)directly connected to one of the quantities of interest and operate at the proper time scale.

(Iwasaki & Levy, 1993; Levy *et al.*, 1992) describe another extension to compositional modeling, based on *relevance reasoning*. The input, consisting of a scenario description, a query and a library of model fragments, is similar to the input in (Falkenhainer & Forbus, 1991). However, in relevance reasoning model fragments are annotated with (ir)relevance conditions. Relevance conditions state when a fragment is relevant to a goal. Relevance heuristics, e.g. ‘all model fragments explicitly mentioned in the query are relevant’, can now be expressed in terms of these (ir)relevance conditions. These relevance heuristics are used to select appropriate model fragments.

Other model composition approaches. There are also a number of approaches related to, but not based on, Compositional Modeling. Most of them are based on the *bond graph* representation. In these approaches, *bond graph* representations are used as an intermediate description between components and (qualitative or quantitative) equations.

Possible bond graph representations for each component are described in a library of model

fragments, and model composition makes use of this library.

The KEMS system (Xia, 1994) presents an example of such an approach. Input to the system is a model composed from generic components (comparable to a scenario description in Compositional Modeling), a set of explicit *model satisfaction criteria* and a library of generic components with associated bond graph representations. Some generic components may have more than one bond graph representation associated with them. Qualitative equations can be derived directly from the bond graph representation. Behavioral properties are described in terms of domain-dependent constraints. Examples of these domain-dependent constraints are: domain(model), static or dynamic(model-type), order(model), etc. Initially, the simplest bond graph representation is chosen for each generic component. More complex partial bond graphs are selected incrementally until the model satisfies all behavioral properties. KEMS is the first approach in which explicit attention is paid to the specification of a wide range of model satisfaction criteria, which is also one of the main topics in our own research.

AIM (the Automated Intelligent Modeler) (Smith *et al.*, 1996) has been developed from the KEMS system. Input to AIM is again a model description in terms of generic components and connections between them, a set of model satisfaction criteria and two libraries of model fragments, a *component* library and an *m-component* library. The *component* library contains domain-dependent knowledge about all real-world components known to AIM, as well as default information about physical parameters. The *m-components* library is domain-independent, and contains all knowledge necessary to model the components in the model. Each m-component includes a bond graph representation together with causal information indicating the directions in which energy can flow between ports. The m-components library also contains information on the improvements that can be applied to each individual m-component. Improvements allow addition of physical effects like friction, compliance and induction to be added to the bond graph representation for this m-component. This means that there is no ”maximum” model for an m-component, but m-components can always be extended if greater detail is required. AIM starts the modeling process by identifying m-components to describe all variables of interest mentioned in the scenario description and in the task-specific model satisfaction criteria. Then additional m-components are added to the model until there are no more free variables in the model. Then the model is assessed with respect to the task-specific model satisfaction criteria, the most suitable corrections to m-components are identified and the model is altered accordingly. This process continues until all model satisfaction criteria are satisfied.

QPAS (Ishii & Tomiyama, 1996), which stands for Qualitative Process Abduction System, is not based on the bond graph representation but rather on a library of *physical phenomena* and *physical features*. Input to the system is a desired behavior expressed in the form of a state-transition graphs (STG) of concrete sub functions requested of the device. Given a state-transition graph, QPAS derives appropriate physical features that can realize each specified function. This reasoning procedure proceeds in two steps: First, QPAS searches for physical phenomena that are able to cause the transitions in the STG, e.g. 'torque generation'. In the second step, QPAS searches for physical features that can cause these physical phenomena. For example, for the physical phenomena 'torque generation', a physical feature that models the mechanism of electromagnets used in a motor can be suggested. Selection between alternative features is typically left to the designer.

KA (Goel, 1996) takes a different approach to model construction. The context of the model construction task here is to acquire a functional model for a new device which is closely related to the original device but with some structural differences. For example, to acquire a functional model of a fire extinguisher, KA is supplied with a complete functional model of a related device, in this case a spray can, and a structural specification of the fire extinguisher itself, describing its components and interactions. KA autonomously acquires a functional model for the fire extinguisher by adapting the functional model of the spray can. Adaptation knowledge is described by means of *skeletal model-revision plans*, relating structural differences between devices to executable plans for adapting the functional model. The focus of this work is limited to adaptation based on structural differences between devices, and does not take into account other reasons for model revision, like situations where models are not appropriate for fulfilling a specific task.

(Schut & Bredeweg, 1993b) focuses on *supporting* a modeler in specifying, assessing and debugging a qualitative model, rather than on automation of the complete modeling process. One of the issues addressed is model construction where the necessary domain knowledge is incompletely available. Specification of models and model fragments is supported by a set of dedicated editors. Another important issue in this research is assessment and debugging of qualitative models. Assessment of a qualitative model against observed qualitative behavior is supported by a taxonomy of discrepancies in different level of detail. A number of *specific techniques*, some fully automated and some in interaction with the user, are being developed to remedy each discrepancy (Schut & Bredeweg, 1993a; Schut & Bredeweg, 1995).

2.3.2 Model selection

In model selection approaches models are not constructed bottom-up from a library of sub models, but complete models are *selected* iteratively from a predefined library of models. Figure 2.3 illustrates the process of model selection. When selecting a model from a library of models, the modeling problem now becomes how to select a model 'appropriate' for the task at hand. However, opinions on when a model is 'sufficient' for a task differ considerably. This subsection describes different approaches to model selection, and discusses which 'measures of appropriateness' are used in each of them.

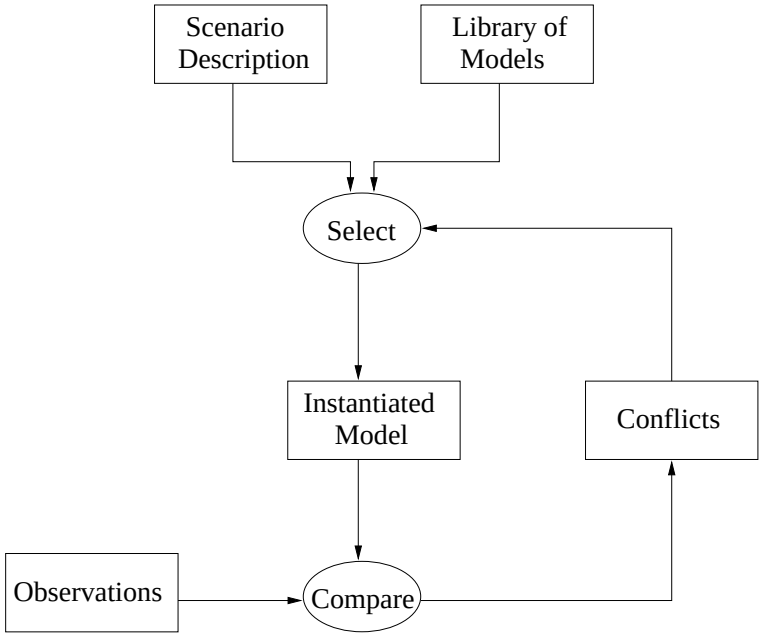


Figure 2.3: Model selection.

Graphs of Models (GoM)

The problem addressed in the Graphs of Models approach is how to select the appropriate model, given a set of *measured values* of the system to be modeled. The Graphs of Models (GoM) (Addanki *et al.*, 1991) approach depends heavily on a previously defined Graph of Models. Each Graph consists of a fixed set of quantitative models that contain knowledge of some applied physics domain (e.g. gear transmission, ideal gas thermodynamics and fluid mechanics). Each model is a formulation of a piece of domain knowledge that can only be used if its approximations lead to an acceptable approximation of the world. These approximations are described explicitly by means of modeling assumptions. The models provide the nodes in the graph, and the edges are the assumptions that have to be changed in going from one model to another.

The input for a modeling session with the Graphs of Models approach consists of a choice for which set of models to use (e.g. gear transmissions, fluid mechanics), and a set of measurements from the real system to be modeled. The graph of models is supposed to be predefined for a specific domain by an expert in the field, and is assumed to be fixed during the model selection process itself. Model selection proceeds in a number of steps: Initially the simplest model is selected. Then a numerical behavior prediction is generated from this model. This prediction is assessed against the set of measurements from the real system. When a behavior prediction for a specific variable does not match the measured behavior, a *delta vector* is created that indicates which assumptions are to be changed to produce a better model. A list of all delta-vectors is used to determine the best transition from the current model to one of its direct neighbors. This process continues until the behavior predictions from the model satisfy the measured behavior of the real system within a user-specified tolerance.

MODEL (Weld, 1992a; Weld, 1992b) uses the Graph of Models representation to reason explicitly about the *accuracy* of models. Shifts in accuracy between models can be achieved by the basic operations of (goal-directed) model simplification and (discrepancy-driven) model refinement. In goal-directed model simplification, assumptions are introduced in a complex initial model in order to simplify the model, while in discrepancy-driven model refinement assumptions are retracted until the model behavioral predictions agree with measured data. These shifts in model accuracy can be represented in a Graph of Models, as long as each transition between adjacent models is a *fitting approximation* (Weld, 1990), i.e. when the differences between the behaviors predicted by adjacent models can be brought arbitrarily close to zero.

2.3.3 Model induction

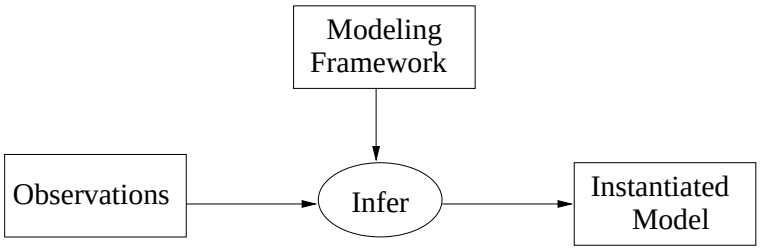


Figure 2.4: Model induction.

Model induction approaches aim at inferring a model from the behavior it exhibits. Its input consists of a description of the behavior of a system, and its output consists of a

model of this system in a specific modeling framework. In numerical contexts, this task of behavior-based model induction is the subject of a field called *system identification* (see (Ljung & Glad, 1994a; part 3) for a recent book on system identification and its role in modeling). In this field several efficient methods and algorithms for identifying a model from its behavior have been produced. These algorithms have as input numerical measurements on the behavior of a system, and deliver as output a mathematical model of the system. There are two important subtasks in this paradigm: determining the *structure* of the model to be used (i.e. the form of the equations) and determining the values of constants (parameters) which appear in these equations. As model structures either ready-made, black-box models or tailor-made models can be used. Tailor-made models are constructed from basic physical principles, and parameters in these models represent unknown values of system parameters. So, parameters in these models have a physical interpretation. Black-box models are families of flexible models of general applicability. In black-box models, parameters have no physical interpretation but are only used to ensure that the resulting model satisfies the input-output behavior. Physical modeling has directed its attention to making tailor-made models, while the field of system identification has mainly focussed on producing efficient algorithms for estimating the value of parameters in ready-made model structures. However, recent work has shown that the approaches can be used to complement each other (Nagy & Ljung, 1992; Gawthrop *et al.*, 1992).

Most model induction approaches in AI originate from machine learning techniques. They concentrate on deriving structure from behavior, producing a qualitative model based on a qualitative description of system behavior. Input to these model induction systems consists of system behavior and the modeling framework in which the model is to be expressed. Different model induction approaches impose different restrictions on this input: some systems require that all possible behaviors are described, some require that also impossible behaviors are described (negative examples) and others require additional structural information on the system to be modeled, like the relevant components in the system. Output of these model induction approaches is a qualitative or quantitative model structure, for example in the form of a set of qualitative equations (Richards *et al.*, 1992; Say & Kuru, 1996) or a bond graph (Amsterdam, 1992; Redfield, 1992). In this section we will describe two of these approaches in more detail. Both make use of the bond graph description: MM (Amsterdam, 1992) constructs a qualitative bond graph model based upon a qualitative description of behavior, while MODA (Wilson & Stein, 1995) constructs a quantitative bond graph model based upon a quantitative behavior description in terms of

the Frequency Range of Interest (FROI). For a more extensive review of qualitative model induction techniques in general see (Schut & Bredeweg, 1996).

MM (Amsterdam, 1992) is a good example of a model induction approach using bond graphs. Input to MM is a structural description (either geometrical, component-based, or a combination of these two) of the system to be modeled and the qualitative behavior the model should exhibit. MM first constructs a basic (bond graph) model for the system by identifying spatial sections in which behavior is uniform (the ”lumping problem”), and then combines basic bond graph models for each of these sections into a minimal model. This bond graph is then tested against a qualitative description of system behavior. Based on qualitative differences between observed and predicted behavior, global *modeling rules* are used to extend the bond graph model incrementally till its predictions fit the observed system behavior. The main feature of MM is that there are no definite model fragments for any given part of the system: only bond graph models are defined for different components and these can be augmented as needed with basic bond graph elements.

The MODA system (Wilson & Stein, 1995), for Model Order Detection Algorithm, is an example of an approach which has grown out of traditional quantitative engineering traditions. In this approach a bond graph representation is used as an intermediate description between the system description in terms of components and the numerical mathematic equations used for simulation and analysis. Inputs to MODA are a generic components model and a specific numerical measurement for the behavior of a system, the Frequency Range of Interest (FROI). Output of the MODA system is a bond graph extended with numerical values for each parameter. This is the main difference between MODA and qualitative approaches based on bond graphs (e.g. KEMS and MM): not only the structure of the bond graph needs to be determined, but also the numerical value for each parameter in this structure. The resulting model can be used for numerical simulation or other forms of mathematical analysis.

2.3.4 Summary

In *model composition* approaches, the user provides the initial system description, in more or less detail, and usually a question to be answered by the model. After that, the model construction process is performed fully automatically. Additional model satisfaction criteria, like model parsimony and model sufficiency, are hard-coded into the modeling system, and can not be altered by the human modeler. Most of the model composition approaches do not take the flexible and iterative nature of the modeling process into account: model

satisfaction criteria are fixed and can not be altered by the human modeler, and interactive model assessment and adaptation is not supported. Also, though model construction approaches rely heavily on the presence of a complete and detailed library of model fragments, almost no support is provided for the task of specifying and maintaining this library.

Current *model selection* approaches also typically focus on a small, fixed set of model satisfaction criteria: behavioral adequacy ((Addanki *et al.*, 1991), model simplicity or model accuracy ((Weld, 1992b). Although model selection approaches, like the GoM approach, often focus on assessment and adaptation, the restriction to complete models strongly influences the re-usability aspects, since for each new application system a new Graph of Models must be constructed. No support is provided for building the (rather complex) Graph of Models, and extending the graph with new models is difficult due to the intricate interactions among models in the graph.

Model induction approaches are data-driven, in the sense that they focus on inferring a model of a specific system directly from the behavior it exhibits.

Neither quantitative system identification techniques nor qualitative model induction techniques pay much attention to the flexible and iterative nature of physical modeling. The only criterion the resulting model should satisfy is to reproduce a given behavior, so there is no scope for reasoning about other task-dependent criteria. Most model induction approaches do not provide support for assessment and adaptation of previously constructed models, since they focus on generating new models directly from behavioral data. Also, models constructed by model induction are usually difficult to reuse since they provide no validity information on the scope of situations for which the inducted model is supposed to be valid.

Table 2.1 presents an overview of automated modeling systems, their modeling strategy and the model satisfaction criteria they use in model construction.

2.4 What is to be done?

There is a large gap between current AI approaches to physical modeling and the current practice in engineering. In most engineering applications, modeling is seen as an iterative task: a model is proposed, assessed and adapted until it satisfies the (explicit and implicit) user needs. Most AI approaches to physical modeling are based on the idea of automat-

System	Modeling Strategy	Model satisfaction criteria
CM	Model Composition	model structure & model parsimony
Causal approximations	Model Composition	model structure & model parsimony
TRIPEL	Model Composition	time scale & model structure
AIM	Model Composition	causal consistency
KEMS	Model Composition	behavioral, task-specific
QPAS	Model Composition	user-defined
(Schut & Bredeweg, 1993b)	Model Composition	model parsimony & qualitative model behavior
KA	Model Revision	device structure
GoM	Model Selection	observed behavior
MODEL	Model Selection	model accuracy
MODA	Model Induction	observed behavior (FROI)
MM	Model Induction	observed behavior (qualitative)
007 *(this thesis)*	*Model Revision*	*behavioral & structural, goal-specific*

Table 2.1: Summary: Automated modeling approaches.

ing the entire modeling process by selection and construction from a library of models or model fragments. This approach leaves the modeler with hardly any means to influence the outcome of the modeling task, and there are almost no facilities for tuning the model to the specific requirements that the modeler might have.

What is needed is an integrated theory on how the iterative nature of the modeling process can be supported. This theory should cover a wide range of different goals in modeling and simulation, and associated model satisfaction criteria. By examining the state of the art in automated modeling, and the abilities of current approaches to support the iterative nature of engineering modeling we have identified a number of issues that will need to be elaborated in such a theory. These issues will be dealt with in depth in the remainder of this thesis.

- **Support for specification of task-specific model satisfaction criteria**

  Most automated modeling systems support model construction for only a very limited number of model satisfaction criteria. In real-life engineering applications, model requirements are diverse and depend on the *analysis goal* the modeling process is meant to achieve: many different models and methods can be used to satisfy a large number of possible analysis goals.

  Human modelers have difficulties in accurately and completely specifying their goals, and the model requirements associated with these goals. They could use help in de-

termining the consequences of alternative analysis goals, in deciding which analysis method to use to achieve a specific analysis goal and in keeping track of requirements associated with each analysis goal.

Providing automated support for *specification* of these modeling goals and requirements, as well as guidelines on what method and what type of model to use for each goal, can help in easing the burden of the specification of model satisfaction criteria.

In this thesis, we plan to support the process of iterative modeling by supporting the specification of modeling goals and goal-related model requirements. We will show that knowledge-based structuring of and reasoning with modeling goals is essential in supporting specification in engineering modeling.

- **Model assessment and adaptation**

  Modeling in engineering usually is an iterative process: models are proposed and gradually refined and modified until they meet the modelers needs and demands. An intelligent environment supporting this paradigm should provide support for assessment and adaptation of models. It should also enable the user to specify a wide range of model requirements, and provide assessment and adaptation suggestions for a wide range of modeling problems.

  Many current automated modeling systems feature some form of assessment and adaptation in their internal operation. However, this does not necessarily mean that they are suitable for supporting the iterative nature of the process of engineering modeling: almost all of them are restricted to assessment and adaptation of a simulation model based on a single or a limited number of model requirements. No attention is paid to the dependencies between goals, methods and goal-specific model requirements.

  Assessment and adaptation are essential tasks in supporting the iterative nature of the modeling process. In this thesis, we will develop a general theory for model assessment and adaptation. This theory will use qualitative and quantitative model adaptations to solve a wide range of modeling problems.

- **Support for specification of modeling assumptions**

  Many current automated modeling approaches depend heavily on the availability of a library of models or model fragments. Specification and verification of model fragments in such a library is a difficult task. However, with a few exceptions ((Schut & Bredeweg, 1993b; Breunese & Breedveld, 1996)), automated modeling approaches provide almost no support for the construction and maintenance of this library. Sup-

port for specification of model fragments is usually not provided, and extending the current library with additional model fragments is difficult due to the fact that the automated modeling approach is highly dependent on the coherence between the models in each category.

When model fragments must be reusable, it is not enough to correctly specify the model fragments themselves but it is also essential to describe the context in which the model fragment can be used, in terms of its underlying modeling assumptions.

The large space of possible modeling assumptions and their multiple interactions makes explicit specification of assumptions a difficult task for most human modelers.

In this thesis we focus on supporting the specification of the *modeling assumptions* underlying the construction of a model fragment. To this end, we suggest a semi-automated structured approach to guide the modeler through the assumption specification process. This approach also provides feedback on consistency and completeness of the assumptions specified.

Chapter 3

Multiple engineering ontologies

This chapter describes a general framework for representing physical models developed in the OLMECO[1] *project (Top* et al.*, 1995b). This framework is based on the idea that during the modeling process, an engineering system is considered from many different viewpoints.* Functional components, physical processes *and* mathematical relations *are proposed as essential ontological perspectives in describing the dynamics of engineering systems. This separation of different ontological viewpoints supports knowledge sharing and reuse, and helps to make modeling decisions in a more incremental fashion. The model representation described above will be used throughout this thesis as the framework in which the content of physical models is described.*

3.1 Multiple engineering ontologies

The model representation developed in the OLMECO project is based on the observation that during the modeling process, an engineering system is considered from many different angles. We can see it as a device constructed out of various functional components; in physical modeling we ask what kind of physical processes are at work in the system; when running a simulation we are dealing with a system as a collection of differential and algebraic equations. A physical model representation should reflect these different ontological viewpoints. As extensively discussed elsewhere (Top & Akkermans, 1994; Borst *et al.*, 1997), this separation of different ontological viewpoints supports knowledge sharing and reuse, and also helps to make modeling decisions in a more incremental fashion.

[1]The OLMECO project has been supported 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 were PSA Peugeot-Citroën (France), BIM (Belgium), Fagor (Spain), Ikerlan (Spain), Imagine (France) UT (the Netherlands) and ECN (the Netherlands).

To illustrate the concepts in this representation of engineering systems, we will use the following simple running example, consisting of a string of railroad cars impacting on a snubber (adapted from (Karnopp *et al.*, 1990)). The schematic in Figure 3.1 shows the system just as contact occurs. A possible (informal) question with this model could be: 'How will the most left railroad car move in time?' Variations on this example will be used throughout this thesis to illustrate our approach.

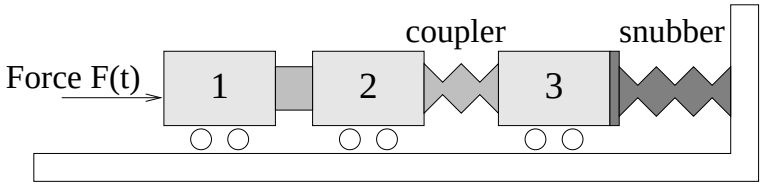


Figure 3.1: Example system: a string of railroad cars impacting on a snubber.

3.1.1 Functional components

Many engineering systems can be represented *compositionally*, by means of components and connections between them. It is common engineering practice to construct physical systems by configuring off-the-shelf components of which the function is known. Therefore, the first viewpoint considered in the OLMECO model representation is that of functional components and their connections. Functional components express the interfaces that exist between a (sub)system and its environment and, furthermore, they carry a label to indicate a class of engineering functions — such as springs and masses in the mechanical domain, or pipes and pumps in fluid flow systems. The component level in a physical model is also used to represent information about *is-a* and *part-of* hierarchies. Is-a hierarchies are taxonomies based on invariant properties in which more detailed models can inherit general properties from less detailed ones. Part-of hierarchies are defined by the decomposition of functional components. Figure 3.2 shows the component level for our railroad car system. The train is decomposed into individual components for each of the three railroad cars and for the coupler. Each item in the engineering drawing (depicted in Figure 3.1) is modeled as a separate functional component. All interactions between components in this system take place in the mechanical translation domain.

3.1.2 Physical processes

The process level of the OLMECO model representation describes the physical processes that underly system behavior and shows the way component functions are realized by

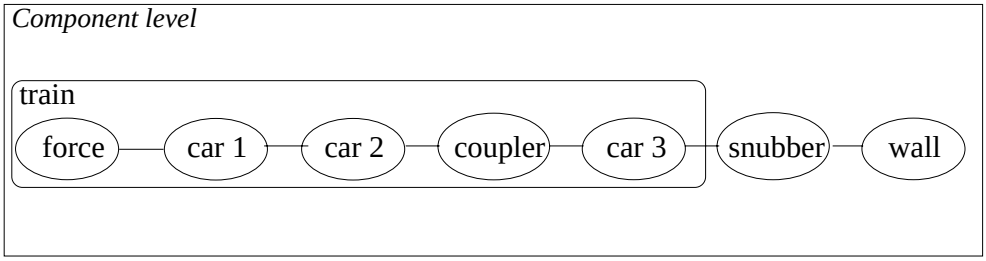


Figure 3.2: Functional component model of the railroad car system. All interactions between components take place in the mechanical translation domain.

physical laws and mechanisms. In this model representation the processes taking place in a functional component are represented in a graphical way by means of so-called bond graphs (Karnopp *et al.*, 1990).

A bond graph is essentially a network model (a labeled digraph) for physical systems, where the edges represent energy flows and the nodes represent elementary physical mechanisms. An important underlying idea is that different physical domains can be described in terms of a complementary pair of physical variables. For example, in mechanics these are force and velocity, in electromagnetism voltage and current, in hydraulics pressure and volume flow, in thermodynamics temperature and entropy or heat flow. These pairs can be chosen in different ways, but in bond graphs this is done such that their product always represents power (energy per unit of time). In the parlance of bond graphs they are called effort and flow variables: effort times flow has the dimension of power. As a result, each edge of the graph can be interpreted as a flow of energy, and is labeled by the corresponding pair of physical variables.

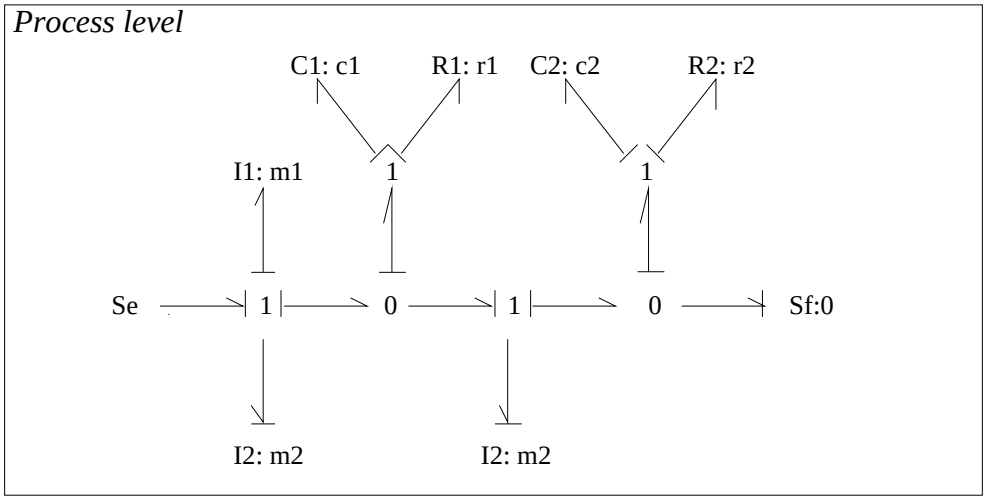


Figure 3.3: Process level description of the railroad car system. All bonds are in the mechanical translation domain, with force as effort and velocity as flow.

The nodes are elementary physical mechanisms indicating different possible operations on energy: storage (C, I), dissipation (R), energy sources (Se, Sf), and energy distribution (0, 1). The bond graph in Figure 3.3 for the train system can thus be read as follows. The driving engine is indicated as an external power source (Se, left). Each car is represented by an I-node: an inertial mass, acting as a store of kinetic energy. Both the coupler and the snubber are modeled as a damped spring: a C-node, representing ideal spring-like behavior, plus a resistance effect (R). Finally, the wall is again modeled as an external power source (Sf:0, right), whereby the velocity is imposed to be zero. The connecting structure with 0's and 1's tells us how the total available energy is divided over all physical mechanisms, taking into account the rules of conservation and continuity of physics. So, from the graph we can rather easily get a qualitative *conceptual* picture of the physical processes at work in the system.

Construction of the mathematical model from a bond graph is rather straightforward. Every node typically yields one equation. The C and I nodes give first-order differential equations, and the other nodes generate algebraic relations. Physically, every node expresses an important law. In Figure 3.3 for the train system, the C nodes yield Hooke's law, the I nodes give Newton's law of inertia, the R nodes gives the rule for friction (the friction force equals the velocity times a resistance parameter), the Sf:0 gives a boundary condition for the velocity, and the 0 and 1 nodes tie everything together in agreement with the law of energy conservation.

A nice feature is that these graphs generalize over different physical domains and clearly bring out the analogies between them. For example, the principle of conservation of momentum in mechanics is analogous to the law of charge conservation in the electrical domain. The C and I nodes which in mechanics typically model springs and masses, denote capacitors (stores of electrical energy) and inductors (stores of magnetic energy) in electro magnetic theory — hence the name of these nodes. The R denotes a universal energy loss phenomenon (resistance, friction). The energy distribution nodes 0 and 1 are the well-known parallel and serial connections in electricity. These are characterized by equal voltages and currents, respectively, at the point in question. Analogously, in mechanics they stand for, respectively, equal forces and velocities (cf. Newton's 'action equals reaction').

Thus, bond graphs have a number of features that makes them suitable for automated modeling and model revision:

- They form a convenient diagrammatic notation which is relatively simple to read and understand.

- They contain a lot of conceptual physical semantics, having built in many conservation laws and principles of physics.

- They provide a graphical front-end to equation formulation, which reduces errors and, in the linear case, even automatically provides the system of differential and algebraic equations.

- Due to the built-in analogies they are very attractive for modeling modern multidisciplinary physics (e.g. thermodynamic and mechatronic systems).

3.1.3 Mathematical relations

The mathematical level of the OLMECO model representation specifies the set of equations necessary for symbolic analysis and numerical simulation. This description is typically in the form of (ordinary) differential and algebraic equations. The mathematical level consists of variables, parameters, constants and relations between them. At the mathematical level, symbolic and numerical calculus methods of mathematics can be used for analysis of the system, including numerical simulation.

Mathematical level

q1=int(f6)
q2=int(f12)
p1=int(e9)
p2=int(e2)

e1=Se=1.0
e2=e1-e3-e4
e3=d(p3)/dt
e4=e5
e5=e6+e7

e6=(1/C1)*q1=1.0*q1
e7=R1*f7
e8=e5
e9=e8-e10
e10=e11
e11=e12-e13
e12=(1/C2)*q2=1.0*q2
e13=R2*f13=2.0*f13
e14=e11

f1=f2
f2=(1/I2)*p2=1.1*p2

f3=f2
f4=f2
f5=f4-f8
f6=f5
f7=f5
f8=f9
f9=(1/I3)*p1=1.1*p1
f10=f9
f11=f10-f14
f12=f11
f13=f11
f14=0

Figure 3.4: Mathematical description of the railroad car system. All equations are default since they are linear.

Figure 3.4 present a mathematical description of our railroad car system. Generally, each C- and I-process node gives rise to one first order differential equation. The algebraic equations indicate the external inputs, the conservation rules governing the system, and which variables are shared between different parts of the system. In this example all equations, both ordinary differential and algebraic, are supposed to be linear, and can be automatically derived from the bond graph representation.

3.2 Conclusions

In this chapter we have described a three-layered model representation for describing the content of physical models (Top & Akkermans, 1994; Top, 1993). In this model representation three important ontological viewpoints are distinguished: *functional components, physical processes* and *mathematical relations*. Separation of these viewpoints supports knowledge sharing and reuse, and helps to make modeling decisions in a more incremental fashion. In this thesis this three-layered model representation will be used as a framework for describing the content of physical models. The structured nature of this model representation provides sufficient handles for the automated assessment and adaptation of physical models.

This model representation also forms the organizational framework for describing and structuring a library of model fragments (Breunese, 1996; Top & Akkermans, 1994). In chapter 7 this framework will be described and extended with the structured specification of modeling *assumptions*.

Chapter 4

Requirement specification

Specification of requirements in physical modeling is a difficult and knowledge-intensive task, and yet there is little support for this early stage of physical modeling. In this chapter we present a knowledge-based approach to support for requirement specification. This approach results in a requirement assistant *that supports the specification of explicit model satisfaction criteria, from an informal, vague problem statement to a set of formal criteria which can be automatically tested against the model and its results.*

The early stage of requirement specification in engineering modeling can be partly automated. Due to the relative stable and formal structure of goals and requirements in engineering modeling, we show that it is possible to provide a taxonomy *of generic engineering modeling goals, and to guide the specification of modeling requirements by making use of a library of predefined* goal cliches. *Each goal cliche contains information on important and optional aspects of a specific type of modeling goal, as well as associated constraints. This library allows the requirement assistant to automatically detect and solve forms of incompleteness, inconsistency and abstractions in modeling requirements.*

4.1 Introduction

In the design of physical systems it is customary to start with the specification of a set of requirements. Often, especially in routine design, these requirements are already formalized into a set of mathematical constraints on the artifact to be designed. In physical modeling however, requirements often remain implicit, and are usually not formalized. A

human modeler usually starts the modeling process with an informal problem statement, indicating the goals and the specific demands of the situation in which the model will be used. During the modeling process this informal problem statement becomes formalized in various steps that go along with the actual construction of the model itself. This first phase of the modeling process is often considered to be an art, rather than a science (Paynter, 1961).

However, in automated model revision it is necessary for model requirements to be specified in a format suitable for automated assessment. Specifying these explicit requirements is a difficult task. This calls for automated support for the process of *requirement specification* in modeling. In our opinion, this has been given insufficient attention in current automated modeling approaches.

In physical modeling, requirements arise from three origins: from the user, from the 'real' world and indirectly from the analysis methods used to answer a modeling query. Requirements originating from the user describe the goals to be achieved by modeling and analyzing a specific system. Requirements from the 'real' physical world describe available domain knowledge on a class of engineering systems, as well as specific knowledge on the real system to be modeled and the conditions under which it operates. In automated modeling, available domain knowledge on a class of engineering systems is often presented in the form of a library of generic model fragments, while specific knowledge on the system to be modeled and the condition under which it operates is often referred to as the *scenario model* (Falkenhainer & Forbus, 1991). Finally, requirements arising from the analysis method are related to the features and limitations of each method. These limitations often pose constraints on (structure and content of) the model to be constructed.

What is needed is support for the task of specifying modeling requirements. In this chapter we will show that the early stage of *requirement specification* in modeling can be partly automated by using a *requirements assistant* to support the modeler in specifying and formalizing his needs and demands. In this chapter we focus on the purpose, or modeling *goal*, for which a model is constructed, the analysis *methods* applicable to this goal, and the explicit *constraints* posed on (structure or content of) the model by the limitations of these analysis methods. To support the specification of these explicit requirements, we propose an interactive 'requirement assistant' which supports different facets of requirements specification in model revision. This assistant is part of the automated model revision system 007 (Pos & Akkermans, 1996a; Pos *et al.*, 1997b).

Section 4.2 describes the role of requirement specification in model revision, and intro-

duces the concept of a requirement assistant to support requirement formulation. The requirement assistant forms a bridge between the informal needs and demands of the engineer and the automated model revision system, which requires explicit, testable requirements in order to perform assessment and adaptation. Some important problems in communicating informal requirements between the engineer and the requirement assistant are discussed. Section 4.3 describes the ways in which the information in requirements is structured in our requirement assistant. We focus herein on the *modeling goals* for which a model is constructed. *Goal cliches* form a central concept in structuring requirement information. We also show how these goal cliches help to handle some of the problems in communicating and formalizing informal requirements. Section 4.5 describe an example session with our requirement assistant, in which an engineer is supported by the requirement assistant in specializing and formalizing his informal needs and demands. Finally, section 4.6 describes related research on the topic of requirement specification, in our own field of physical modeling and in related areas like (requirement specification for) engineering design and software engineering.

4.2 Requirement specification in model revision

In physical modeling, it is often hard to draw a clear line between specification of *model requirements* and the model (re)construction task itself. In most automated modeling approaches, specification involves the specification of a *scenario model*, usually in the form of some sort of component-connection diagram, and the specification of some form of *modeling query*. Since this specification seems rather straightforward, usually no support is deemed necessary for the process of problem elaboration and requirement specification. Instead, attention is directed mainly to the process of model construction from a set of predefined model fragments.

We claim that in current engineering practice, specification of modeling requirements is not that straightforward a task for the following reasons: In the last decades, the systems to be modeled have become increasingly complex, and accurate predictions are essential for controlling their behavior. Furthermore, many engineering systems today do not belong to a single engineering domain, but require the combined work of several engineering disciplines. Also, more and more facets of the design process are supported by some form of physical modeling, and the number of available modeling and analysis techniques grows and grows. All this puts a tremendous strain on the human modeler (or, as is more often the case today, the modeling team), and makes it harder to specify the problem at hand

and its influences on the design process.

However, in automated model revision testable requirements are essential in automated assessment and adaptation of the physical model. Automated support for the process of requirements elicitation and requirements specification should help the modeler in clarifying and making explicit his specific modeling goals. It should also provide him with information on available analysis techniques to satisfy these goals. The purpose of the requirement assistant described in this chapter is to guide the modeler through this process.

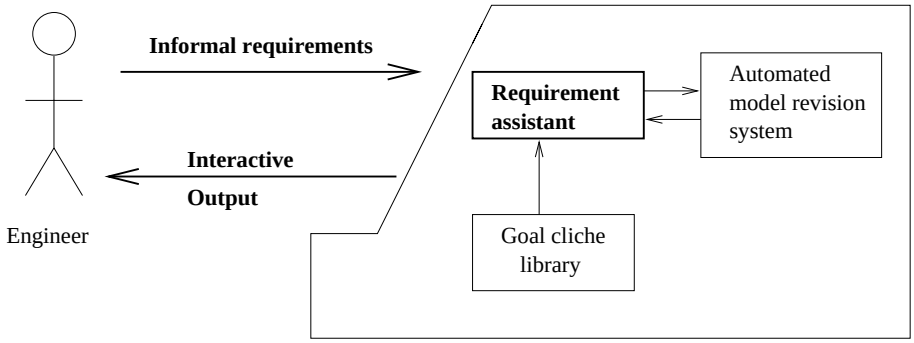


Figure 4.1: The role of the requirement assistant.

Figure 4.1 shows the role of our requirement assistant in relation to other agents involved in the model revision process. The *requirement assistant* forms a bridge between the informal needs and demands of the *engineer* and the *automated model revision system* which needs testable requirements to start the process of model assessment and model revision. The requirement assistant thus supports the engineer in formulating and making explicit his *needs and demands*. A *library of goal cliches* provides the requirement assistant with information relevant to different types of goals. The requirement assistant produces two kinds of output. *Interactive output* notifies the engineer of conclusions drawn, choices to be made and inconsistencies detected during the requirement specification process. A *set of testable requirements* is directed to the automated model revision system. The automated model revision system uses these testable requirements to assess and adapt the current model until it meets the requirements. An example session where the requirement assistant supports the engineer in formulating and making explicit his requirements is presented in section 4.5.

Requirement specification concentrates on the issue of informality, and the process by which informal requirements become formal ones. Communicating informal requirements between a human modeler and an automated model revision system raises a number of problems:

- Incompleteness - Human modelers may forget to specify aspects of their modeling goal. A requirement assistant can help by *structuring* and presenting important aspects of often used modeling goals.

- Abstractions - Human modelers often use more abstract descriptions to indicate their requirements. These abstractions are usually just a shorter way to describe a cluster of related requirements. An automated requirement assistant can support this process by automatically *deducing consequences* of abstractly formulated requirements, thereby obtaining the set of testable requirements implicit in the abstractly formulated requirement used by the human modeler. These testable requirements can then be used to automatically assess and adapt the actual model.

- Inconsistency - When requirement sets get larger, contradictions may occur between different requirements. A requirement assistant can help by automatically *detecting inconsistencies* in the requirement set.

We handle these problems, and thereby support the requirement formulation task, in two ways: first, by presenting a taxonomy of generic modeling goals and, secondly, by providing a form of semi-automated requirement management. A library of goal cliches, each describing the aspects that are necessary for a proper description of a specific type of modeling goal, provides the requirement assistant with information relevant to different types of goals and with associated testable requirements. The structure and content of the goal taxonomy and associated goal cliches will be discussed in section 4.3. In section 4.4 we will show how these goal cliches can be used to solve problems of incompleteness, abstractions and inconsistency in communicating informal requirements between a human modeler and an automated system.

4.3 Goal cliches: Structuring modeling requirements

Structuring the information available in requirements is essential in supporting requirement specification. It not only gives structure to the communication between the engineer and the requirement assistant, but it also supplies the requirement assistant with information essential to detect inconsistencies, incompleteness and abstractions. In this section we describe the structure imposed on modeling goals and requirements, and we discuss how this structure helps to support the engineer in the transformation of an informal problem statement to a set of explicit, testable requirements on the model to be (re)constructed.

In the domain of physical modeling a knowledge-based approach is very suitable, because the structure of goals and requirements is rather stable and quite amenable to formalization. Thus, it is possible to distinguish and predefine frequently used generic modeling goals and the ways in which they interact.

Physical models are always constructed to answer certain questions about a physical system. Therefore, the question to be answered, or the *modeling goal* is the central concept in specification of modeling requirements. During the requirement specification process support is to be provided to the human modeler in clarifying these goals, in selecting an analysis method appropriate for satisfying these goals, and in deriving a set of explicit, testable requirements on the model to be reused.

Another observation to be made is that, although there are in principle an unlimited number of questions to be asked on any physical system, there seems to be a limited number of *goal types* that are often reused to indicate significant types of questions on a physical system. We claim that for each of these frequently used modeling goals a *goal cliche* can be constructed. This goal cliche describes the goal aspects that need to be specified to provide a complete description of each type of modeling goal. Goal cliches provide a medium for effective communication of modeling goals between an engineer and an automated model revision system.

In order to structure the information present in informal problem statements in physical modeling we make use of three forms of structure: we provide a *taxonomy* of modeling goals, a *goal cliche* specifying which aspects are important for each type of modeling goal and a set of *constraints* describing how these goal aspects interact.

4.3.1 Goal taxonomy

Physical models are always constructed to answer certain questions about a system. A modeling *goal* is therefore in the first place a description of the expected results of an analysis process. In the context of physical modeling, several frequently used types of goals can be distinguished: prediction, (parametric) optimization, control and comparative analysis.

- *Prediction goals* answer questions on how the model behaves. Prediction takes as input the model and the model inputs, and gives as output some aspect of system response. Results from this prediction task can be either in qualitative or numeric form. Note that the fact that qualitative results are desired does not necessarily mean that

qualitative methods can or should be used to derive these results. For example, the qualitative question whether a model exhibits oscillatory behavior can be answered by both qualitative (causal analysis) and quantitative (numerical simulation) methods. Prediction is used in the design process to analyze the response of a system before actually building the system.

- *Optimization goals* try to optimize a parametric model in accordance with a specific optimization criterion. This criterion can be specified in terms of (abstractions of) the dynamic response of a system parameter, or in terms of static system characteristics. Optimization takes as input some description of system response, and gives as output a (revised) model showing this response. For example, the algorithms described in chapter 6 modify the parameter values of a given model to satisfy requirements on its dynamic response. Optimization is used in optimal parametric design to find a combination of parameter values for which the results are optimal with respect to some predefined criterion.

- *Control goals* are used to determine if and how a system can be controlled. The task of controller design takes as input a model and some description of desired behavior, and delivers as output a controller. In order to design controllers, however, it is often necessary to first analyze different qualitative aspects of model behavior, like its controllability and its (relative) stability. Therefore, both qualitative and quantitative control goals are included in our taxonomy.

- *Comparative analysis* is used to answer 'what if' questions. 'What if' questions can include the effects of local changes in the model, like 'what happens if I change the type of motor used in the system' or 'what happens if I raise the value of this variable', or they can describe the effects of global system changes like 'what happens if I consider only slow time-scales in system behavior' or 'what happens if thermal effects are to be included'. It takes as input an original model and a (local or global) change to this model, and delivers as output a (qualitative or quantitative) description of changes in model behavior. Comparative analysis is used in design to compare alternative designs, or to determine the sensitivity of a design to small variations.

These general types of goals can be used as the top-level organization of a taxonomy of modeling goals, in which each child is more specific than its parent. Engineers can 'walk through' this taxonomy in order to select the modeling goal which best fits their current informal needs and demands. For each goal, a goal cliche is defined in the library of goal cliches. Figure 4.2 presents (part of) the goal taxonomy used in our requirement assistant.

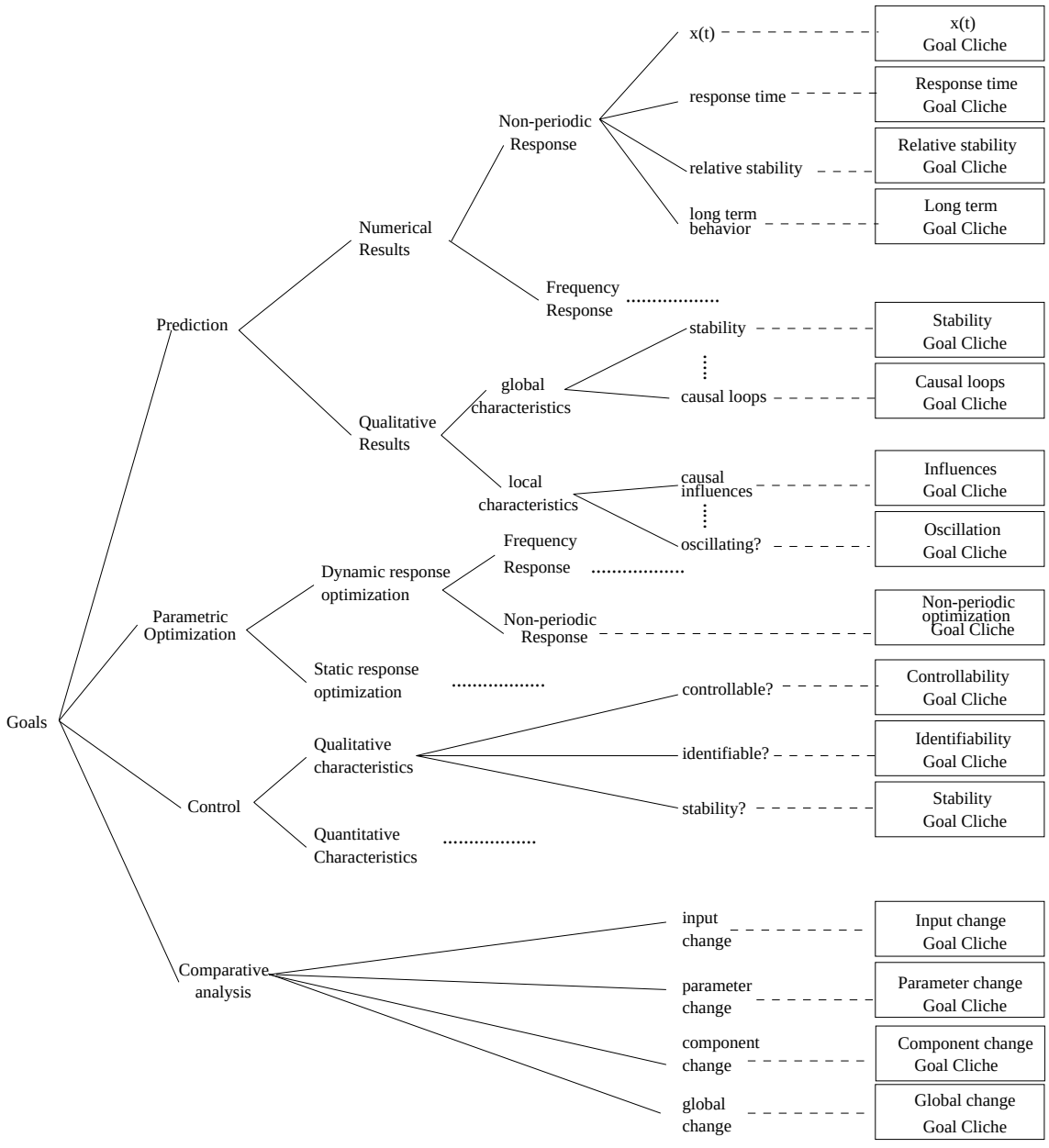


Figure 4.2: Goal taxonomy used in our requirement assistant. This taxonomy focuses on analyzing lumped, dynamic systems.

The taxonomy focuses on both qualitative and quantitative questions in analyzing lumped, dynamic systems. We do not claim that the taxonomy presented here is complete, but it provides a good coverage of important types of modeling goals in the context of physical modeling of lumped, dynamic systems. Additional, more specialized goals can easily be added in the goal taxonomy to support analysis in a more specific context.

4.3.2 Goal aspects

By examining which aspects are important in the specification of each of the modeling goals in the goal taxonomy we derived a set of typical goal aspects of modeling goals. Typical aspects of the modeling goals in Figure 4.2 are the form of results expected, the analysis method by which a goal can be achieved, observations or experimental data on the physical system to be modeled, and possibly additional requirements on model performance, like complexity, accuracy or running time. Collectively, these elements constitute the human modelers 'needs and demands' for the modeling task. Therefore, they should one one hand be close to the concepts familiar to engineering modelers, and on the other hand they should be specific and formal enough to be suitable for automated assessment. Not all goal aspects are relevant in each modeling goal. We identified the following classification of goal aspects:

- *Response description*. This goal aspect describes the 'format' in which the results from the analysis should be presented. The response description is restricted by the goal type in the sense that a specific goal type can produce only a limited number of possible types of response. The response description may put additional restrictions on the analysis method to be used, since not every analysis method can produce a specific behavior description. For example, typical behavior descriptions for the response time are delay time, rise time, peak time and settling time.

- *Quantities of Interest.* Quantities of interest are the variables and parameters of interest to the modeler with respect to his modeling goal. They can describe dynamic input- and output-variables or static parameter values.

- *Observations.* Measured data or safety regulations to which the results of a model should be tuned are a typical example of observations on (desired or actual) system output. Observations are mainly important in optimization goals and in some control goals, and are usually used to tune model behavior.

- *Method.* Each analysis goal can be achieved by one or more methods. For example, the goal 'predict response time' can be achieved by a range of different numerical simulation and analysis methods. Applicability of these methods depends on the goal to be achieved, specific structural aspects of the model to be used and sometimes on the domain in which modeling takes place. Each method can have its own method features that must be specified before an actual experiment can start. For example, a numerical simulation method like the RungeKutta method has as features a step size, a start time and a stop time. Methods may pose additional structural and behavioral requirements on the model structure to which they can be applied. Deducing these additional requirements is one of the tasks of the requirement assistant, and will be discussed in more detail later on in this section.

- *Behavioral requirements.* Requirements on system behavior can be either derived by the requirement assistant from other goal aspects like the response description or the analysis method selected, or they can be stated explicitly by the engineer. Behavioral requirements can be specified in qualitative or quantitative terms. An example of a behavioral requirement is the oscillatory nature of model response. This requirement can be automatically derived by the requirement assistant whenever response descriptions like the delay time are used, since this type of response description is only defined for responses of oscillatory nature.

- *Structure requirements.* Structure requirements describe restrictions on the structure of (parts of) the model. These structure requirements can often be automatically derived by the requirement assistant from other goal aspects like the analysis method and the response description. An example is the desired linearity of a model, which is a necessity for analysis methods like transfer function analysis.

- *Performance requirements.* For large models it may be necessary to put restrictions on the performance aspects of the (simulation) model, like the computation time necessary to produce results, or the accuracy with which the analysis results should be computed. These restrictions can be expressed by means of performance requirements.

4.3.3 Goal cliches

Goal cliches describe which goal aspects need to be specified for a specific type of modeling goal can be represented in the form of *goal cliches*. For each modeling goal in the goal taxonomy a *goal cliche* is provided in a library of goal cliches. Which goal aspects

are necessary varies from one goal to another: some aspects may be mandatory in one goal type, optional in another and non-existent in a third goal type. As an illustration, Figure 4.3 presents the goal cliche for the modeling goal 'Predict Response Time'. Mandatory attributes are (1) the response description, (2) the Quantities of Interest, and (3) the analysis method to be used. These mandatory goal aspects all need to be filled in in order for the goal cliche to be completely specified. An optional goal aspect for this goal is the maximum computation time allowed. A selection of choices is provided for each goal aspect. In Figure 4.3 only two selections are shown: those for the goal aspects *response description* and *method*.

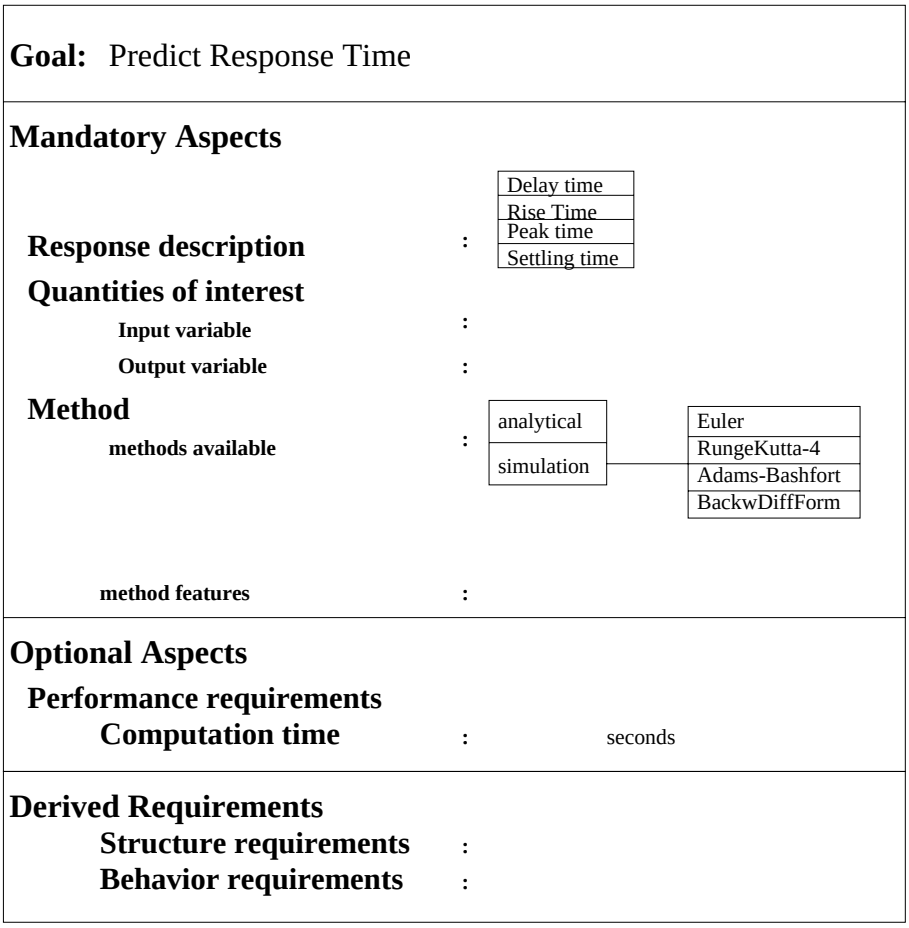


Figure 4.3: Goal cliche for the modeling goal 'Predict Response Time'.

Goal aspects are often not independent; a choice made for one goal aspect may restrict the possible choices for other goal aspects. For example, the set of method features in Figure 4.3 depends on the analysis method selected. Dependencies between goal aspects and requirements are available to the requirement assistant in the form of predefined constraints. Constraints can be used in different ways: to restrict the selection for a goal aspect based

on the choices made in earlier goal aspects, to add additional requirements to the requirement set based on the goal aspects specified, or to detect inconsistencies in the requirement set.

Constraints:

IF response-description = delay time
THEN input-type = step-input

IF selected-method = RungeKutta4
THEN method-features(start-time, stop-time, stepsize)

IF selected-method = RungeKutta4
THEN structure-requirements-add(SystemSelfContained(yes, global))
structure-requirements-add(BondgraphComplete(yes, global))
structure-requirements-add(ZeroOrderPaths(no, global))

IF selected-method = analytical
THEN structure-requirements-add(MaximumModelOrder(2, global))
structure-requirements-add(ModelLinear(yes, global))

IF response-description = delay time(output variable)
THEN behavioral-requirements-add(QualitativeBehavior(OsciallatoryDamped, output variable))

IF QualitativeBehavior(OscillatoryDamped, _)
THEN structural-requirements-add(MinimumModelOrder(2, global))
structural-requirements-add(SystemContainsDamping(yes, global))

IF selected-method = simulation
THEN stop-time >= start-time

Figure 4.4: Some constraints between goal aspects in the goal 'Predict Response Time'.

The upper part of Figure 4.4 presents some examples of the first type of constraints. These constraints describe the interactions between possible choices for related model aspects. They reflect that making a choice for one goal aspect may restrict the possible choices for other goal aspects, and thus place limits on how subsequent goal aspects can be filled. For example, the response description 'delay time' is only valid for a step input, and therefore selection of the delay time as the response description restricts the choice for the type of input to the step-input.

The middle part of Figure 4.4 presents some examples of yet another kind of constraint: this type of constraints specifies which, more specific, additional requirements ought to be added to the requirement set. For example, the second constraint in the lower part of Figure 4.4 specifies the fact that in order for an analytical method to be used the model should be linear and its order should not be more than 2. These additional requirements are automatically added to the requirement set by the requirement assistant once the user has selected the analytical method to be used for prediction of the response time. This *require-*

ment refinement can be used to solve problems of abstraction by allowing the engineer to state his requirements in a high-level format. The requirement assistant can now automatically deduce the associated, more specific, requirements from this high-level specification.

The lower part of Figure 4.4 presents a constraint on the relation between two explicit values: the stop-time should never be smaller than the start-time, regardless of the simulation method used. This type of constraints can be used to detect direct (mostly syntactical) inconsistencies in the set of requirements. Inconsistencies can be detected within a single goal cliche, as well as between requirements originating from different goal cliches.

Section 4.4 discusses how goal cliches and constraints can be used to solve some of the problems in requirement specification: incompleteness, inconsistency and the use of abstract, high-level requirement specifications.

4.3.4 Requirements

Every requirement consists of a *statement*, a *value* and a *scope*. The requirement statement describes the aspect of model structure or model behavior governed by the requirement, the requirement value describes the required value for this model aspect and the requirement scope describes the scope for which the requirement is supposed to hold. For example, the structural requirement QualitativeBehavior(DampedOscillatory, output variable) describes a restriction on the qualitative behavior of the model, the required value for this restriction is DampedOscillatory behavior, and the scope for this requirement is a specified output variable. The value of a requirement can be either numeric, alpha-numeric (as in the example presented above) or boolean. The scope of a requirement can be global, i.e. pertaining to the complete model, or restricted to one or more model elements, like the single model output variable in the example presented above.

Our requirement assistant is implemented in an object-oriented fashion: testable requirements are represented as classes, in which each class has its own methods for assessment, testing inconsistencies, etc. Requirements are also structured in a *parent-child hierarchy*. This means that all requirements automatically derived from another requirement (by means of predefined constraints) are considered as children of this requirement, and all requirements originating from a specific goal cliche are descendants of this cliche. For example, the behavioral requirement QualitativeBehavior(DampedOscillatory, output variable) is a direct descendant of the requirement Delay Time(yes, output variable). This

hierarchical representation of requirements makes it possible to keep track of all requirements and their origin, and thereby makes it easier to backtrack to the relevant goal cliches in case of inconsistencies between requirements.

4.4 Using goal cliches

In this section we describe how the goal cliches defined in the previous section can be used to handle three important problems in communicating informal requirements between a human modeler and an automated model revision system: abstractions, incompleteness and inconsistency.

4.4.1 Enabling abstractions

Engineers tend to communicate most easily in terms of abstract, high-level descriptions to describe complex concepts in physical modeling. To allow them to communicate their needs and demands in these terms, the requirement assistant needs to have a certain amount of specific knowledge that 'explains' the abstract concepts and translates them into more basic requirements. Our requirement assistant can automatically deduce consequences of abstract requirements in terms of explicit requirements on the physical model to be (re)constructed. This feature is based on the observation that certain types of requirements, specified in terms of abstractions familiar to the engineer, give rise to additional requirements on behavior or structure of the physical model other than those directly described by the original requirement itself. These additional restrictions are in fact derived requirements on the physical model, and are therefore included in the requirement set. The requirement assistant automatically deduces and keeps track of these additional requirements, allowing the engineer to state requirements in high-level terms without having to worry about the indirect consequences these high-level requirements have on the physical model.

Example 4.1 *Again, consider the goal cliche presented in Figure 4.3 and some of the constraints depicted in Figure 4.4. Now suppose the delay time is selected to describe the response time. This statement is a form of abstraction: the delay time can only be determined if the model response is of an under-damped oscillatory nature. This is captured in the following constraint (depicted in the middle part of Figure 4.4):* IF RESPONSE-DESCRIPTION = DELAY TIME(OUTPUT VARIABLE) THEN BEHAVIORAL-REQUIREMENTS-

ADD(QUALITATIVEBEHAVIOR (DAMPEDOSCILLATORY, OUTPUT VARIABLE)). *As soon as the abstract requirement 'delay time' is selected, the additional requirement 'QualitativeBehavior (DampedOscillatory, output variable) can be added automatically to the requirement set by the requirement assistant, thereby allowing the human modeler to use an abstract, high-level description of his modeling goal.*

4.4.2 Detecting incompleteness

Incompleteness is fundamentally harder to detect and remedy than contradiction. There is no way to detect the absence of information that is orthogonal to the current state of a requirement (or requirement set). For example, it may be possible to detect incompleteness *within* a specific goal, but it is not possible to detect whether the engineer has selected all the proper analysis goals for meeting his informal, unspecified goals and demands. As a result, the engineer will have to be the final arbiter of completeness. Nevertheless, it is possible to detect some kinds of incompleteness and even to help complete the specification. Our requirement assistant can detect incompleteness in the specification of a specific type of goal, and can sometimes help to complete part of the specification by using default values and constraints among different goal aspects.

Each time the engineer chooses a goal type from the taxonomy, a number of expectations are generated in the form of goal aspects that need to be filled in. Also, constraints between aspects allow some aspects to be filled in based on the choices made for other aspects. Our requirement assistant considers a goal to be incomplete until every mandatory goal aspect has been given a value.

Example 4.2 *Consider the goal cliche presented in Figure 4.3. Now suppose the response description and the quantities of interest have already been filled in, but not the analysis method. Since the analysis method is a mandatory goal aspect, and no default value is available for this goal aspect, the requirement assistant detects an incompleteness in the goal cliche and prompts the user to remedy the incompleteness by specifying the analysis method to be used. If a default value had been available for the missing goal aspect, the requirement assistant could have filled in the missing aspect automatically.*

4.4.3 Contradiction detection

Because our requirement assistant is not capable of making all possible deductions from a set of requirements, not all contradictions that might be present in the users requirement set can be detected. However, our requirement assistant can detect or even prevent some inconsistencies, both within a single goal cliche and between different goals, by making use of predefined *constraints* among goal aspects.

Possible inconsistencies within a single goal cliche can be prevented by examining the possible interactions between different goal aspects. As discussed before, the possible choices for a specific goal aspect are sometimes dependent on choices made for other goal aspects. These dependencies can be represented in the form of constraints, and these constraints can be used to automatically prevent inconsistencies by restricting the possible choices for a goal aspect.

Example 4.3 *Again, consider the goal cliche presented in Figure 4.3 and some of the constraints defined among its goal aspects depicted in Figure 4.4. Now suppose the delay time has been selected as response description. Since the delay time is only defined for a step input, the requirement assistant can automatically fill in the corresponding goal aspect, thereby preventing inconsistencies between the consequences of the response description 'delay time' and the type of input to be used.*

Inconsistencies can not only exist within a specific goal cliche, but can also result from different goals. To detect these inconsistencies, global constraints must be specified to describe when different requirements are in conflict. In the object-oriented implementation of our requirement assistant each requirement has its own methods for detecting inconsistencies, regardless of the goal from which the requirement was originally derived. For example, a requirement on maximum model order knows that it can be in conflict with a requirement on minimum model order if the maximum value is smaller than the minimum value, since the model order can not at the same time be larger than 3 and smaller than 2. The requirement assistant automatically tests these constraints, and indicates which requirements are in conflict. The user should now decide which of the conflicting requirements to modify. The requirement assistant supports this process by keeping track of which requirements originate from which goal cliche, and by providing alternative choices for decisions made during the specification of each goal cliche.

4.5 Example session

In this section we will follow an engineer in the specification and formalization of his needs and demands into a set of explicit, testable requirements. The simple railroad car example system discussed in chapter 3 will be used to illustrate our approach. The schematic in Figure 4.5 shows the system just as contact occurs.

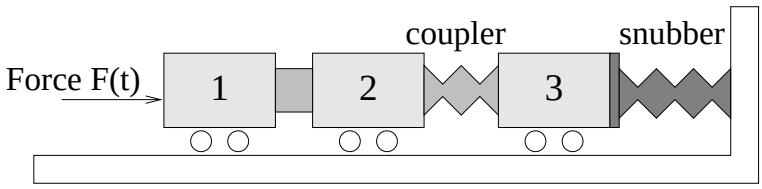


Figure 4.5: Example physical system: a string of railroad cars impacting on a snubber (reprint of Figure 3.1, chapter 3).

An example of an informal modeling problem statement for the system in Figure 4.5 might be: 'I want to know how long it will take for car1 to become stationary, and I want to know if this complies with design criteria posed by the management (say, maximum settling time $\tau_s = 5$ seconds). An additional requirement could be that the resulting model is to be used in a time-critical environment, so computation of this feature must be faster than real time.

Step 1: Goal selection Since the primary modeling goal consist of *predicting* the response of the system, and since the fact that the *time* it takes for car1 to become stationary is essential and presupposes numerical results, the engineer selects the goal 'Predict Response Time' from the goal taxonomy depicted in Figure 4.2. The requirement assistant now retrieves the goal cliche for this goal, and presents this cliche to the engineer. Of the available selection menus, only the ones for the goal aspects 'response description' and 'methods available' are shown in Figure 4.6 (a copy of Figure 4.3, reprinted here for the convenience of the reader).

Step 2: Response description. The engineer starts the goal specification by choosing which response description best captures the response time behavior for his current purpose. The possible choices for this response description are restricted by the modeling goal: only response descriptions suitable for describing the response time behavior of the system can be selected. In this example, the fact that the time for car1 to become stationary is important to the engineer can be translated into the fact that the *settling time*, a general engineering measure for damping speed, is the most suitable. In fact, this statement is a form of *abstraction*: the modeler 'forgets' to mention the fact that the settling time measurement is defined only if the response is of an under-damped oscillatory nature.

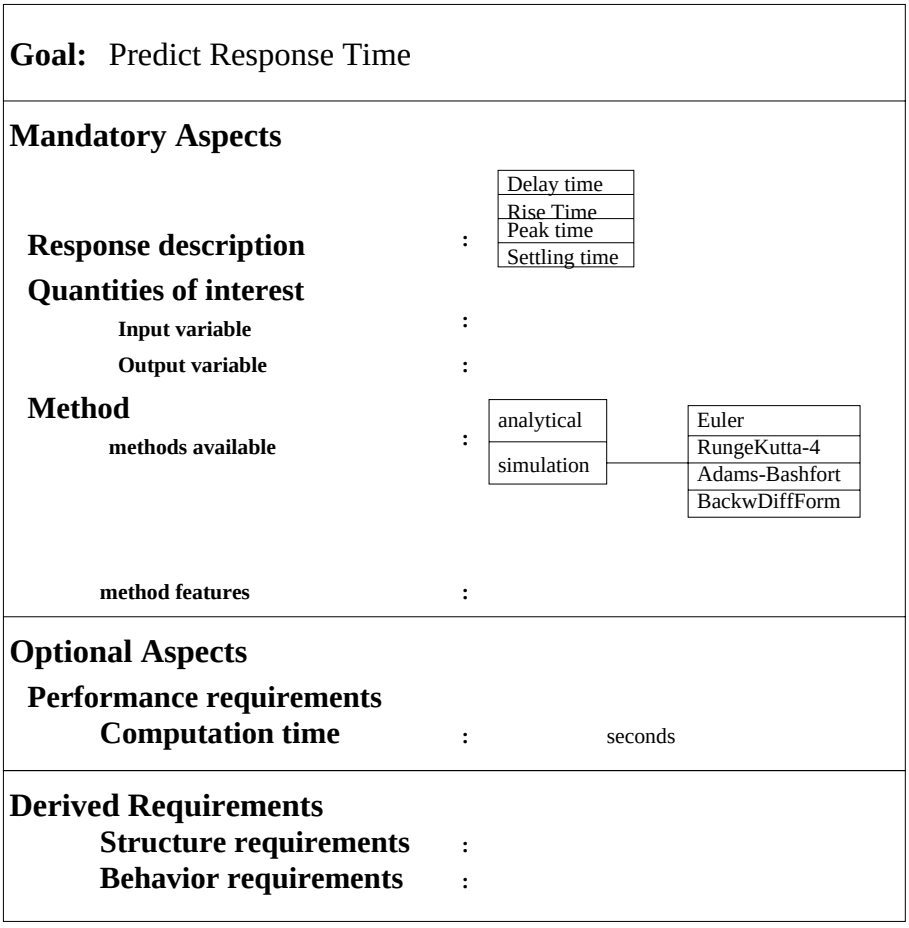


Figure 4.6: Goal cliche for the modeling goal 'Predict Response Time'.

Therefore, our requirement assistant automatically adds this as an additional requirement to the (derived) behavioral requirements in the goal cliche.

Step 3: Quantities of Interest. Now the quantities of interest need to be specified. In the case of numerical prediction, one output variable and one input variable must be specified. For the input variable both the name and the type of input needs to be specified. Our requirement assistant provides the engineer with a selection menu containing all possible dynamic output-variables in the current model representation (not shown in Figure 4.6), from which the modeler selects the single output corresponding to the position of car1. Then, the requirement assistant provides the engineer with a selection of all system inputs, by examining all sources present in the bond graph representation. Since in this example the only possible input variable is the variable F(t), this variable can also be selected by default. The *type of input* is restricted by the fact that the output must be described in terms of the settling time, since this measurement is only used for describing the reaction of a system to a *step input*. This allows the requirement assistant to automatically fill in the type of input with a step input, thereby preventing possible inconsistencies in the goal cliche.

Now the modeler tries to accept the instantiated goal cliche. However, the requirements assistant detects an *incompleteness* in the goal cliche, and indicates to the modeler that the mandatory goal aspect describing the analysis method has not yet been filled in.

Step 4: Method. Now a method needs to be specified to derive the settling time from the model representation. For this analysis multiple methods are available, and the requirement assistant lists a number of them. The engineer decides to try a simple simulation method, called the 4th order RungeKutta method. Again, this statement is a form of abstraction: although the RungeKutta method is low in complexity, it poses a number of additional requirements on the model structures to which it can be applied. In order to simulate the physical system with the RungeKutta method, the model must be in so-called explicit state-space form. This is the typical form of differential equation models we find in elementary textbooks, but in real-life cases there is no guarantee that the modeling process will yield this form. An explicit state-space form implies that the model has to meet several form criteria. It must be complete and self-contained: no conflicts may occur in assigning causality, or the input-output order, to equations. There should be no static or rigid couplings: these are also referred to as zero-order paths, and are undesirable because they give rise to implicit equations. This means that all equations should be either algebraic or solvable by integration; equations where derivatives have to be calculated are not

allowed. If any of these structural criteria is not satisfied, the system cannot be solved by standard integration methods such as the RungeKutta method. Our requirement assistant automatically derives these consequences, and adds them as additional derived structure requirements to the goal cliche.

Now that the analysis method has been selected as the RungeKutta method, the method features for these method can be filled in. In the case of standard integration methods such as RungeKutta, necessary method features are the *step-size* with which simulation is to be performed, the *start time* on which simulation should start and the *stop time* on which simulation should be terminated. These features are all essential in running the actual experiment to determine the response. In this example, the engineer decides to run the experiment for a time-period of $2 * T_s = 10$ seconds, starting at time zero, to evaluate the system response in terms of the design criterion, and he chooses a step-size of 0.1. The requirement assistant performs some syntactical *consistency* checks (e.g. checking if the stop time does not occur before the start time) on these values, and then accepts the given values.

Step 5: Performance requirements. Performance requirements put restrictions on the performance aspects of a model, like its complexity and the computation time necessary to produce results. The engineer decides to make the statement in the informal problem description that 'computation should be faster than real time' more specific by stating it as a performance requirement : the computation time should be less than $\tau_s = 5$ seconds.

Step 6: Derived requirements One behavioral requirement, stating that the system behavior should be of an under-damped oscillatory nature, has already been added by the requirement assistant in step 3. Oscillatory behavior can only occur if the system is at least of order 2, and under-damped behavior can only occur if some form of damping is incorporated in the model. The requirement assistant automatically adds these structural requirements to the set of derived requirements.

Step 8: Accept Now the modeler decides to accept the filled-in goal cliche, presented in figure 4.7, and thereby to include all its explicit requirements in the complete requirement set. The requirement assistant checks if all mandatory aspects are filled in, and adds the explicit requirements to the complete requirement set. If more goals are already included in the requirement set, the requirement assistant checks the consistency of the complete set of explicit requirements. Since in this example only a single goal has been defined, this test can be omitted. The set of explicit, testable requirements is now transferred to the automated model revision system, where it can be used to assess and adapt the model

to be reused.

Goal: Predict Response Time

Mandatory Aspects

Response description : Settling Time

Quantities of interest

Input variable : F(t) (step input)

Output variable : position(car1)

Method

method selected : RungeKutta-4 Method

method features : start-time: 0 seconds
stop-time: 10 seconds
step-size : 0.1 seconds

Optional Aspects

Performance requirements : MaximumComputationTime(5 seconds)

Derived Requirements

Settling Time (yes, position(car1))

QualitativeBehaviour(OscillatoryDamped, position(car1))
SystemContainsDamping(yes, global)
MinimumModelOrder(2, global)

Method = (simulation, RungeKutta4)

SystemSelfContained(yes, global)
BondgraphComplete(yes, global)
ZeroOrderPathsAllowed(no, global)

Figure 4.7: Filled goal cliche for the modeling goal 'Predict Response Time'.

4.6 Related work

In this section we will describe other approaches related to requirements engineering, both from our own field of physical modeling as well as from the fields of software engineering and engineering design.

4.6.1 Approaches in physical modeling

Although requirements specification is not yet a big issue in (automated) physical modeling, there are a number of modeling systems that discuss some of the issues developed in our requirement assistant. However, most of these take a far more restricted view on the role of requirement specification in engineering modeling.

In the Compositional Modeling (CM) approach (Falkenhainer & Forbus, 1991) the requirements consists of a *case model*, describing components and initial conditions of the system to be modeled, and a *modeling query*. The latter has some similarities with what we call modeling goals, although the range of possible queries seem to be more limited than the range of possible modeling goals in our work. *Query analysis* in Compositional Modeling seems only to consist of extracting relevant quantities of interest from the modeling query. Furthermore, queries are to be posed to the system in a strict, formalized format, and no support is provided for the process of clarification and formalization and its problems of incompleteness, abstractions and inconsistencies.

Support for the task of *goal specification* in modeling is also incorporated in IDEALZ (Shephard & Wentorf, 1994) and the Integrated Simulation Consultant ISE (Fishwick, 1991). ISE is essentially an information retrieval system that incorporates an embedded expert system consultant EXC. ISE allows engineers to explore which alternative modeling and analysis methods are available in modeling, and EXC provides engineers with expert advice on which method to use for a certain type of problem if they need it. EXC uses rules like 'IF (subject = gears) AND (there is an interest in modeling only gears and connecting rods) THEN use GEARSIM program' to link problem aspects to available modeling and simulation methods. IDEALZ (Shephard & Wentorf, 1994) is a system for analysis idealization control in the context of finite-element analysis. It comprises a goal manager to coordinate analysis goals during a design process. With this goal manager, users can select a set of modeling goals, and edit some of the goal attributes. The goal manager then retrieves analysis strategies by matching the analysis goals and its attributes to a library of analysis strategies. Although IDEALZ and ISE both provide support for the task of goal specification, they mainly focus on selection of the analysis method and do not address the problems in dealing with incomplete, inconsistent and abstract requirements.

Other modeling systems do not focus on the specification of modeling goals in general, but are restricted to the specification of a single aspect of model response. For example, the system described in (Schut & Bredeweg, 1993b) provides support for the specification of discrepancies between expected and actual model behavior. Support is provided by the

definition of a taxonomy of possible discrepancies between observed and simulated qualitative behavior, and by enabling the modeler to specify these discrepancies in different levels of detail, ranging from very sparse descriptions like the number of states missing to detailed information on what the expected behavior should have been. Approaches like these are usually restricted to reasoning about a single type of modeling goal and a single analysis method, and therefore they omit the goal-dependent reasoning incorporated in our requirement assistant.

4.6.2 Approaches in software engineering

Although requirements engineering has been a major subject in software engineering for a long time, there are not yet many 'requirement assistants' for software engineering providing the type of automated support described in this chapter. Support for specification of software engineering requirements is usually provided in a much less knowledge-intensive form. Knowledge-intensive approaches to requirement specification, like our requirement assistant, can be used mainly in domains with well-defined concepts and requirements. Situations where the same kind of software systems have been designed many times before in the same domain and with the same types of requirements are most suitable for this type of support. A frequently used example of such a situation is the design of library information systems.

The software engineering approach which most closely resembles our own requirement assistant is the requirements apprentice RA developed in the Programmer's Apprentice project (Rich & Waters, 1988). The overall goal of the Programmer's Apprentice project is the creation of an intelligent assistant for all aspects of software development. The requirement apprentice RA (Reubenstein & Waters, 1991) assists a human analyst in the creation and modification of a set of software requirements, with an emphasis on the transition between informal and formal requirements specifications. RA uses a library of reusable requirement cliches to codify domain knowledge on structure and content of requirements. These cliches are similar to our goal cliches. RA provides support for handling abstractions, incompleteness and contradiction in informal requirements, just as we do, but it also tackles problems of word ambiguity (by considering context), poor ordering of requirements and inaccuracy of informal requirements.

The computer-aided requirements engineering (CARE) environment, developed in the NATURE project (Nature Team, 1996) provides the most extensive support for requirements engineering. It comprises tools for the management and integration of different view-

points, provides process guidance by giving ‘way of work’ advice, and provides facilities for tracing, critiquing and validating a set of requirements.

Other approaches to providing support for the specification of software requirements provide intelligent editors for drawing ER-models and other diagrams, analyze existing forms (e.g. sales order forms) in the context of data modeling and data-base design (Mannino *et al.*, 1986) or try to extract a (preliminary) ER model by examining a natural language description of the application domain (Kersten, 1986).

4.6.3 Approaches in engineering design

Although several standard works on engineering design methods ((Cross, 1989; Pahl & Beitz, 1996)) emphasize the importance of requirement elaboration and specification, and indicate methods for executing this task, there are not many engineering design systems providing extensive *automated support* for this task. The reason for this lack of automated support seems to be that requirement elaboration and specification is often seen as an art, and not considered suitable for (partial) automation.

The framework DESIRE (Brazier *et al.*, 1995) provides support for the design, specification and implementation of complex systems. In its generic task model of design (Brazier *et al.*, 1994), reasoning about sets of requirements is an important part of the design task. Specific strategies can be employed to manipulate sets of requirements. The generic task model of design has been specialized and instantiated for different design tasks, among which the task of elevator configuration design (Brazier *et al.*, 1996b). In this concrete task, the completeness and consistency of a set of requirements is checked automatically and suitable extensions to the requirement set are suggested.

There is also some interest in automated support for requirements elaboration and specification from the field of case-based reasoning. A good example is the *index elaboration* process described in (Maher & Balacandran, 1994). Index elaboration is a process that transforms the given specifications of a new problem into a more appropriate set of specifications. If an extensive case base is available, index elaboration can be supported by *incremental retrieval*: by examining the set of cases retrieved for an initial incomplete specification, the specification can be extended with other relevant specifications or restricted to a subset of critical specifications. This new set of specifications can then again be used to retrieve more relevant cases from the case-base. The main difference between our approach and the approach described in (Maher & Balacandran, 1994) is that for their

approach an extensive case-base of possible designs is assumed to available, and can be used for index elaboration.

4.7 Conclusions

Dealing with requirements is in our view a crucial part of modeling and design tasks, which up to now has received relatively little attention in automated modeling. In modeling and design systems for engineering applications, requirements support is often nonexistent or very limited, because requirements are being viewed as exogenous and fixed beforehand. In reality, requirements are often changed in the course of the process just as the designs and models are. Ultimately, intelligent systems must be able to handle requirements engineering on the same footing as the construction of models and designs. Our approach to requirement management presents a step into this direction.

In this chapter we have described the task of formulating and formalizing modeling requirements in physical modeling, and the problems associated with it: requirements are often incomplete, stated in abstract terms or inconsistent. We showed that the early stages of requirement specification can be partly automated by using a knowledge-based approach in which expert knowledge on structure and content of requirement is modeled and used to support requirement specification.

Modeling goals form a central concept in structuring modeling requirements. We have classified a number of important types of modeling goals in physical modeling, and provided a taxonomy of modeling goals in the context of the analysis of dynamic, lumped systems. *Goal cliches* describe central concepts in each goal, as well as constraints between these concepts. Our approach resulted in a requirement assistant, which performs semi-automated requirement management and thereby supports the human modeler in the specification of requirements in engineering modeling. Our requirement assistant uses the library of goal cliches to tackle problems like incompleteness, abstraction and inconsistency in requirements.

In our current approach, we have focused on goals in engineering modeling. However, the techniques presented in this chapter can also be extended to include more explicit design goals. Just as modeling goals can be linked to goal aspects and analysis methods, so could explicit design goals be linked to corresponding modeling goals. The general idea of guiding specification in physical modeling and design by means of a taxonomy of generic goals together with semi-automated requirement management based on a library of goal

cliches is, in our opinion, generic enough to be applied to other areas of engineering design in which frequently used, semi-formalized goals and requirements can be distinguished.

The next chapter will discuss how the explicit, testable requirements resulting from the requirement specification process discussed in this chapter can be used to automatically assess and adapt physical models.

Chapter 5

Assessment and adaptation in model revision

There are types of model revision that can do without the availability of a large library of model fragments. In this chapter we focus on 'constructional' solutions to modeling problems, in which the structure and content can be changed to solve specific modeling and simulation problems without having to resort to a library of model fragments.

In automated model revision we have to resolve a number of issues. The first issue is to distinguish handles for basic revision operations in the model and requirement representation used. These can function as basic building blocks from which larger revisions can be constructed in a compositional manner. The second issue is to automatically find out which *revisions are most appropriate, in case of discrepancies between the current model and the posed requirements. The key point here is to tie the engineer's modeling expertise and the requirements knowledge together in a convenient way. Repair plans provide a link between model-requirement discrepancies and generic solutions. Each repair plan is composed out of a set of basic revision operations.*

Our structured representation of both model and requirements offers a wide variety of aspects to be adapted. This, together with our use of multiple methods for assessment and adaptation allows us to identify and solve a wide range of modeling problems.

5.1 Introduction

In the previous chapter we described how the human modeler can be supported in specifying a set of requirements suitable for automatic processing. In this chapter we will discuss how these explicit requirements are used for assessment and adaptation of a physical model.

Most automated modeling approaches focus on model revision based on (intelligent) selection from a library of models (Addanki *et al.*, 1991) or model fragments (Nayak *et al.*, 1992). The most important limitation of these approaches is that they depend on a complete and consistent library of models or model fragments. We think this assumption is too restricted in real engineering modeling situations: in most applications some models and model fragments might be available, but usually the library will be far from complete. In contrast, we will show that there are general model structure considerations that generate automatic revisions, but do not depend on the availability of a large library of model fragments.

A key point in model revision is how to tie together the engineer's modeling expertise and the requirements knowledge in a convenient way. Goal (Goel, 1996) has first introduced the notion of a family of *model-revision* plans, containing generic but executable modification tactics, based on acquisition of expert engineering knowledge. In our approach to model revision a modified version of this concept, called *repair plans*, is used to suggest and carry out repairs on the model or on the analysis method used. Each repair plan is composed out of a set of basic model revision operations.

In the general context of modeling and model analysis, not only the model structure and content can be adapted in order to solve modeling problems, but also the method used for model analysis. Often, multiple methods are available for performing the same model analysis, each with its own different restrictions on model structure and content. For example, for small (second-order, linear) models both algebraic analysis as well as many different numerical simulation methods can be used to predict model response in time, while for larger models only numerical simulation methods are suitable. Most automated modeling approaches focus solely on adaptation in terms of model structure and content, and are not capable of changing the analysis method used for deriving results from this model.

In most automated modeling approaches, only a limited number of requirements can be specified, assessed and remedied. In our approach, different methods of assessment and

adaptation are used to satisfy a wide range of modeling requirements. Adaptations are generated at run-time, and an extensive, complete library of model fragments is not essential. Our model representation offers a large number of aspects to be adapted, and allows for both qualitative and quantitative assessment and adaptation. Furthermore, not only the model itself can be adapted but also the analysis method used to derive results from the model.

In this chapter we describe our approach to assessment and adaptation in model revision. In section 5.2 we discuss the role assessment and adaptation play in the model revision process. In section 5.3 we describe a typology of basic modifications, showing which handles our model representation and requirement representation provide for assessment and adaptation. In section 5.4 we show how more complex representations of assessment and adaptation knowledge, called *repair plans*, can be composed from these basic structural modifications. When these instruments for assessment and adaptation have been discussed, section 5.5 and 5.6 discuss the tasks of assessment and adaptation in more detail. In section 5.7 we present an illustrative example of assessment and adaptation in our model revision system 007. Section 5.8 compares our approach to assessment and adaptation with related approaches in (semi-)automated modeling, and section 5.9 presents our conclusions.

5.2 Model revision

As indicated in chapter 2, model revision can be decomposed into three steps: *requirement management*, *assessment* and *adaptation*. Requirement management has already been discussed in chapter 4. One of the issues discussed there has been the fact that often multiple methods are available for achieving a specific goal, and that each method poses it own additional requirements on the structure and behavior of the model to be analyzed. This means that in case of discrepancies between a model and the prerequisites for a specific analysis method, not only the model can be adapted but also the method used to perform model analysis.

In our approach to model revision, assessment consists of testing the current model against a set of explicit requirements. This assessment results in a set of one or more model-requirement *discrepancies*. *Revision knowledge* in the form of *repair plans* is used to link these discrepancies to useful structural or parametric adaptations of either the current model or the analysis method used to analyze the model. Since the analysis method

used is part of the requirements specification, adaptation of the analysis method requires an additional phase of requirements management in which consistency and completeness of the new set of requirements is checked. In case of inconsistency or incompleteness, the human modeler is notified. Figure 5.1 represents this decomposition of the model revision task.

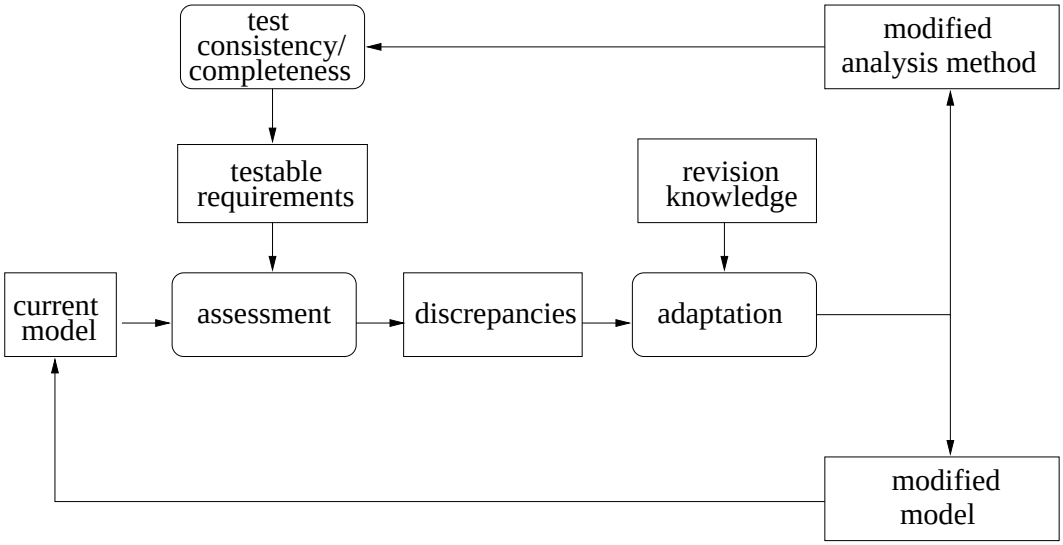


Figure 5.1: Assessment and adaptation in model revision.

5.3 Basic Revision Operations

In this section we present a typology of basic structural modifications, in which we describe how our model representation (chapter 3) and our representation of requirements (chapter 4) provide handles for model assessment and adaptation in the form of basic revision operations. From these basic revision operations larger revisions can be constructed in a compositional manner.

5.3.1 Physical model representation

Our representation for physical models (Top & Akkermans, 1994), also described in chapter 3, consists of three levels: functional components, physical processes and mathematical relations. Each of these levels provides its own hooks for basic model revision operations. The example first presented in chapter 3, a string of railroad cars impacting on a snubber, will again be used as a running example in this section to illustrate the different levels in our model representation, and the handles they provide for model revision.

Component level

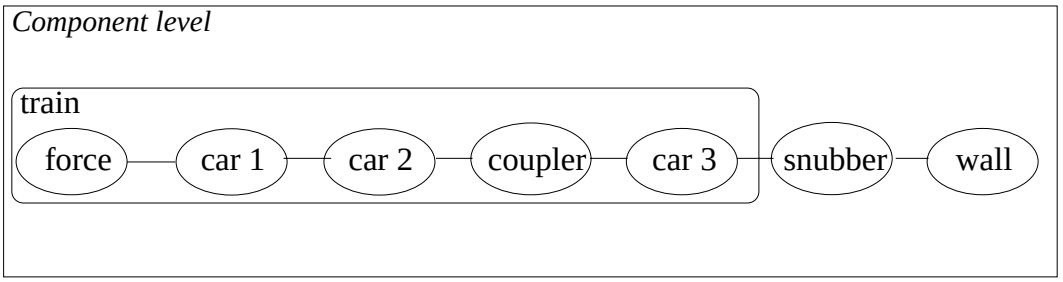


Figure 5.2: Component level (reprint of Figure 3.2, chapter 3).

Figure 5.2 shows the component level for the railroad cars example used throughout this thesis. For more details on this level, its representation and for an explanation of the example used see chapter 3. The component level provides a number of handles for the assessment and adaptation of physical models:

- The well-formedness of the component level can be easily checked, e.g. by checking if there are no unconnected ports.
- The compositional structure of the representation makes it easy to add, delete or replace components in the current physical model.
- *Is-a* and *part-of* hierarchies are useful in guiding the replacement of components or subsystems by other components or subsystems. The fact that car1 is-a car could be used to replace car1 by a car with a different decomposition in which, for example, wheels are modeled as separate components.
- The links to alternative processes, generating the dynamic behavior of the components, can be employed to select different physical mechanisms occurring in a component. For example, car1 is, according to Figure 5.3, now purely acting as an inertial mass, but we could opt for additionally including the physical process of air friction.

Adaptations at the component level typically require a library of model fragments. Since in our current approach to model revision we focus on synthetic adaptations that can be applied without having to resort to an extensive library, adaptations at the component level are not directly included in our current approach to model revision. Instead, we focus on synthetic adaptations at the process and mathematical levels.

Process level

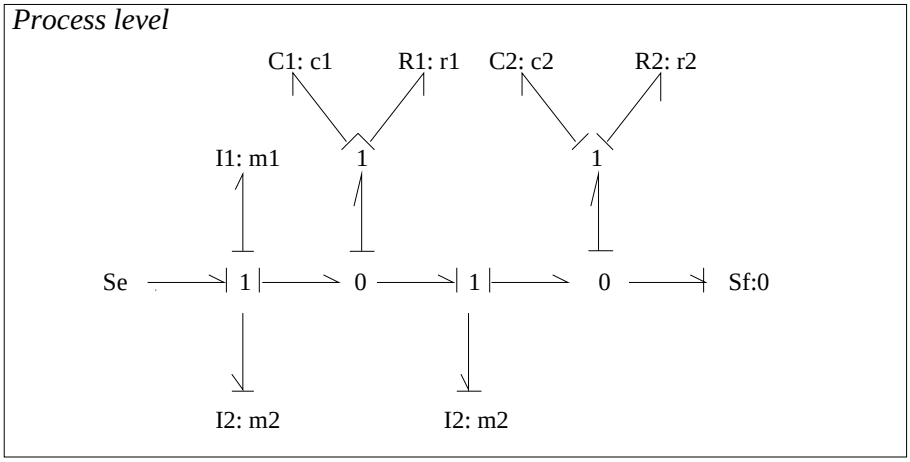


Figure 5.3: Process level (reprint of Figure 3.3, chapter 3).

The process level of our model representation represents the physical processes in a system in the form of a bond graph (Karnopp *et al.*, 1990). An impression of bond graphs as a physics knowledge representation, and an explanation of the example used, can be found in chapter 3. Figure 5.3 shows a process level description of our railroad cars example. The process level provides a number of handles of its own for the assessment and adaptation of physical models:

- Different forms of causal analysis can easily be performed in the bond graph representation in order to assess model structure.
- The compositional structure of the bond graph representation makes it easy to add, delete or replace physical mechanisms in the current model. This feature can be used to carry out *structural* adaptations.
- The typing of the nodes gives additional information about the physical semantics, which is employed to guide modifications in the graph structure. This aspect of typing is also important for obtaining genericity in the repair plans, since it allows us to describe modifications in terms of type of nodes (e.g. ‘add a resistance process’) instead of in terms of actual processes (e.g., ‘add resistance between wheels and ground’).
- The links to alternative mathematical relations ‘implementing’ the physical mechanisms can be used to select a different mathematical description (e.g., linear versus non-linear) for a physical mechanism in order to modify model structure and behavior. This typically requires a library of mathematical descriptions.

Mathematical level

The mathematical level specifies the set of equations necessary for symbolic analysis and numerical simulation. The mathematical level consists of variables, parameters, constants and relations between them. It provides the following handles for the assessment and adaptation of physical models:

- Numerical simulation methods and other mathematical analysis methods can be applied to the mathematical level to assess qualitative and quantitative model behavior.
- The value of parameters and the initial values of state variables can be changed to modify model response. This corresponds to *parametric* adaptation.
- The functional shape of parameters can be changed, for example from static to time-dependent and vice versa, to modify model structure and response. This typically requires a library of parameter-relations.
- The form of the constitutive relations can be changed, say, from non-linear to linear to modify model structure and response.

5.3.2 Requirements

As discussed in chapter 4, modeling requirements can be represented in the form of a parent-child hierarchy. In this section, we will show which hooks this representation provides for automated assessment and adaptation. For more information on requirement representation, as well as on the specific example used in this section, the reader is referred to the previous chapter.

Our hierarchical representation of requirements, an example of which is depicted in Figure 5.4, provide the following handles for assessment and adaptation:

- All testable requirements can be automatically assessed against the current model. The emphasized requirements in Figure 5.4 provide some examples.
- Requirements are structured in a parent-child hierarchy. This means that, given a discrepancy, or violated requirement, all parents can be traced. This feature supports tracing and focusing in the requirement set. In Figure 5.4 the parent-child hierarchy is indicated by indentation, e.g. the requirements *SystemContainsDamping* and *MinimumModelOrder* were both derived from the requirement *QualitativeBehavior*.

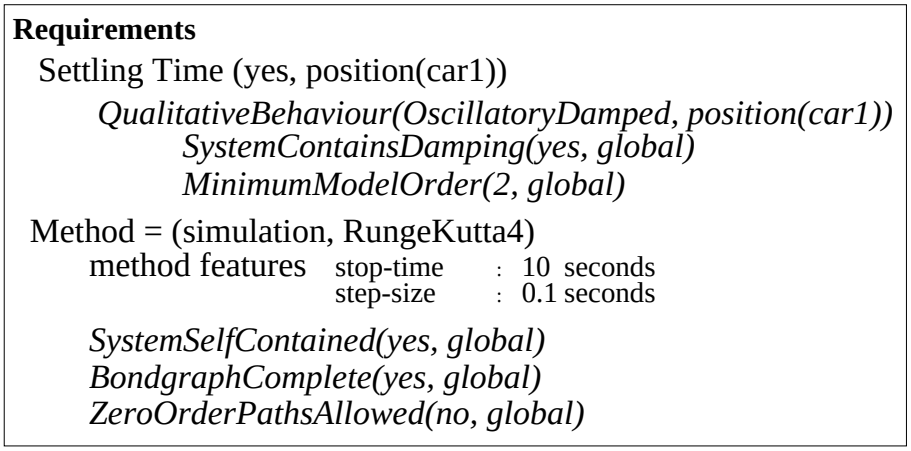


Figure 5.4: Hierarchical representation of requirements.

- Methods may pose restrictions on model structure or model content. The (absence of) these restrictions can be used to automatically suggest a more suitable method for a given analysis goal. As an example, in Figure 5.4 the structure requirements SystemSelfContained, BondgraphComplete and ZeroOrderPathsAllowed were derived from the use of the 4th order RungeKutta method. If there is a model-requirement discrepancy, a different analysis method which does not necessitate this requirement can be selected to perform the same analysis of computing the settling time.

- The values of method features for a specific analysis method, e.g. the step size, start time and stop time for the numerical 4th order RungeKutta simulation method (Figure 5.4), can be changed during adaptation.

As shown in this section, our structured representation of models and requirements allows for a large number of possible basic revision operations. Both synthetic and library-based adaptations are possible. In our current approach to model revision we focus on synthetic adaptations, but library-based adaptations could easily be incorporated providing suitable libraries are available. From the basic revision operations presented in this section larger revisions, called *repair plans*, can be constructed in a compositional manner, as will be shown in the next section.

5.4 Structure of repair plans

In order to automatically generate useful modifications of physical models, three types of knowledge should be specified (Goel, 1996): (1) which differences between the model

and the requirement set can be automatically recognized; (2) which elementary modifications can be performed on a physical model; and (3) how can differences between the model and the requirement set be linked to suitable (combinations of) elementary modifications to solve certain types of modeling problems. In the previous section we discussed which basic revision operations can be performed on the physical model and on the set of model requirements. In this section we will discuss how these elementary modifications are combined to form *repair plans*. Repair plans link differences between model and requirements (*discrepancies*) to generic, but executable, repair tactics. The repair plan depicted in Figure 5.5 will be used as a running example.

Repair plan Perform Time-Scale Reduction (slow)
"reduce the model to its most relevant, i.e. slowest, part"

Violation List:

ModelOrder(TooHigh)

Focus conditions:

CausalLoop(X)
LoopOrder(X, 1)
LoopGain(X, minimal)
"find the loop representing the largest contribution to damping in the current model"

CausalLoop(Y)
LoopOrder(Y, 2)
LoopGain(Y, minimal)
"find the loop representing the largest contribution to oscillation in the current model"

Abstract Actions:

Process-Delete(C, $C \ni X \wedge C \ni Y$)
"delete all compliance processes not in the slowest part of the model"
Process-Delete(I, $I \ni X \wedge I \ni Y$)
"delete all inertance processes not in the slowest part of the model"
Process-Delete(R, $R \ni X \wedge R \ni Y$)
"delete all resistance processes not in the slowest part of the model"
Process-Delete(Junctions, Simplify)
"delete all unnecessary junctions"

Figure 5.5: Example of a repair plan. This repair plan solves the problem of the order of the model being too high by performing a form of model reduction based on separation of different time-scales in the model.

5.4.1 Discrepancies

Discrepancies are defined as differences between the model under construction and a modeling requirement. Note that not only the fact that a requirement is violated is important in determining what is wrong with a model, but also the *nature* of the difference between the model and the requirement put upon it. Moreover, a single requirement might result in a number of different discrepancies. In fact, discrepancies formalize the concept of different *types of modeling problems* which might occur while designing engineering models.

Requirements, as discussed in the previous chapter, consist of an aspect of system structure or system behavior, an expected value for this aspect, and an (optional) scope for which the requirement is defined. A *discrepancy* describes the 'difference' between the current model and a modeling requirement. Therefore, discrepancies consist of an aspect of system structure or system behavior, a scope and an expression describing the difference between the expected model aspect value and the corresponding aspect value derived from the model. This difference can be qualitative or qualitative in nature.

Example 5.1 *The discrepancy OrderTooHigh(2, global) denotes that the global* aspect *model order of the actual model is too large (the* qualitative difference*). The quantitative difference between the expected value and the model value for this aspect is 2. In the repair plan depicted in Figure 5.5 a generalized version of this discrepancy indicates that the repair plan 'perform time-scale reduction' is suitable for the type of modeling problem concerned with the order of the current model being too large, regardless of the exact quantitative difference between the expected and actual model order.*

5.4.2 Repair plans

Repair plans provide a link between model-requirement discrepancies and generic solutions for these discrepancies. Our approach relies on a collection of repair plans, where each repair plan is suitable for solving one or more *discrepancies*, and in addition specifies a sequence of *elementary actions*, generic enough to be applied to many different models and requirements.

Repair plans are indexed by the type(s) of discrepancies they can help to solve. For example, the *'perform-time-scale reduction (slow)'* plan, shown in Figure 5.5 is useful for solving the problems of model order being too high. Some discrepancies might be solved by more than one repair plan. Each plan represents a single *repair tactic* used for solving

one or more types of modeling problems. A repair tactic generally consists of two important, and interrelated, steps: to determine which part of the model (or the requirement set) is to be repaired, and to determine which modifications should be applied to this part. By eliciting expert knowledge, we discovered that these two steps are often mutually dependent. The type of modification depends on the structure of the part to be modified, or rather the combination of the two steps depends on the discrepancies to be solved. Therefore we have chosen to provide instructions for both these steps within a single repair plan.

A repair plan thus incorporates two heuristic types of knowledge: knowledge on how to find the part of the model (or the part of the requirement set) to be modified for a specific discrepancy and knowledge about how to modify this part. Which part needs to be modified is represented by means of *focus conditions*. The way in which the obtained focus has to be modified is represented by a *sequence of elementary actions*. For an example of a complete repair plan see Figure 5.5. This repair plan will be used as an illustration in the following examples.

Focus conditions

Focus conditions contain instructions on how to find the part(s) of the model (or the part(s) of the requirement set) to be modified for a specific discrepancy. Focus conditions can be applied in many different physical models (or requirement sets) due to their general nature. Focus conditions in a repair plan are used to find a *focus*, i.e. a set of one or more model elements or requirements to be modified. Focus conditions are described in terms of general (model or requirements) structures. In this way focus conditions can be described independently of the specific model involved. At the moment, all focusing in physical models occurs at the process level. Focus conditions are described in terms of causal constructs: causal loops and or paths, their causal order and their path (or loop) value. Analysis of these causal constructs is performed by standard causal algorithms from control engineering.

Example 5.2 *In the example repair plan shown in Figure 5.5, the focus conditions are described in terms of two causal loops, one of causal order 2 and one of causal order 1. Both loops are supposed to have a minimal loop gain. Causal analysis on the level of physical processes produces the actual focus in the current model, in terms of a combination of physical processes.*

Focus conditions on requirement sets provide instructions on how to navigate in the set of requirements in order to find a requirement suitable for adaptation. Navigation through the set of requirements is supported by the hierarchical parent-child structure in requirements. Focus conditions are described in terms of hierarchical operations and in terms of typing of requirements. Focus conditions describe how to 'walk through' the requirement set in order to find a requirement to be adapted.

Focus conditions:

Ancestor(X,ZeroOrderPathsAllowed(no))
Type(X, Method)

"find the method requirement X from which the requirement 'ZeroOrderPathsAllowed(no) was derived"

Figure 5.6: Example of a set of focus conditions defined on the set of requirements. This set of focus conditions finds the method accountable for the structure requirement stating that no zero-order paths are allowed.

Example 5.3 *Figure 5.6 presents an example of focus conditions on a requirement set. These conditions provide instructions on navigating the requirement set in order to find a requirement suitable for modification. The focus condition consist of instructions in terms of the requirement hierarchy (*Ancestor(X, ZeroOrderPathsAllowed(no)*) and in terms of the* type *of requirement suitable for adaptation (*Type(X, Method*). These instructions are used to find the method responsible for the structure requirement 'ZeroOrderPathsAllowed(no, global) by guiding a search through the requirement hierarchy. Applying these instructions to the requirements structure presented in Figure 5.4 results in the fourth order RungeKutta simulation method as a focus for modification.*

Actions

The manner in which the obtained focus found is to be modified is represented by a sequence of elementary actions. Elementary actions are basic revision operations, as described in section 5.3, specified in an abstract manner to be applicable to many different models and requirement sets.

Typical abstract actions on the physical model are the addition or deletion of a physical process of a specific type, or the (re)computation of the parameter for a process of a specific type. Abstract actions are automatically executed by instantiating the generic elements in the instructions with specific model elements in the focus.

Example 5.4 *In the example repair plan depicted in Figure 5.5 the complex repair action of retaining only the slowest part of the current model is operationalized by combining a number of basic structural modifications: deletion of all compliances not in the slowest part of the model, deletion of all inertances not in the slowest part of the model, deletion of all resistances not in the slowest part of the model and deletion of all unnecessary junctions.*

Abstract Actions:

Select-AlternativeMethod(Y)

not(hasChild(Y,ZeroOrderPathsAllowed(no))

AddToGoal(Y)

AddToGoal(children(Y))

"select an alternative method that can handle zero-order paths and add it to the requirement set, together with its children (if any)"

RemoveRequirement(X)

RemoveRequirements(children(X))

"remove the focus requirement as well as all requirements derived from this requirement"

Figure 5.7: Example of a set of abstract actions defined on the requirements representation. These actions are used to select an alternative analysis method for prediction of the settling time which can handle the presence of zero-order paths.

Typical abstract actions on the requirement set are the selection of an alternative analysis method used to satisfy some goal under some specific conditions, or the (re)computation of some of the method features of this analysis method, like the step-size for simulation methods. Figure 5.7 presents a typical example of a set of actions defined on the requirements representation: it provides instructions for selecting an alternative analysis method for computation of the settling time which can handle the presence of zero order paths in the model.

5.5 Assessment

During the requirement specification phase described in the previous chapter, a set of testable requirements has been derived. Assessment starts with a set of one or more testable requirements. Each requirement has its own assessment method. This method can be applied during assessment to test the requirements against the current model and to deter-

mine (qualitative or quantitative) differences between the expected and the model value for a specific aspect. Deriving the model value for a specific aspect is a form of goal-directed reasoning, in which only the model aspects are derived which are necessary to test the current requirements. One requirement can produce multiple model-requirement discrepancies. The resulting discrepancies (e.g., the fact that the settling time derived from the simulated model response is larger than the required settling time) can now be used to determine which repair plans might be useful to modify the physical model in order to satisfy the posed requirements. Available methods for assessment in our approach to model revision include different forms of causal analysis, numerical simulation and other mathematical analysis techniques. Assessment in model revision is depicted in Figure 5.8.

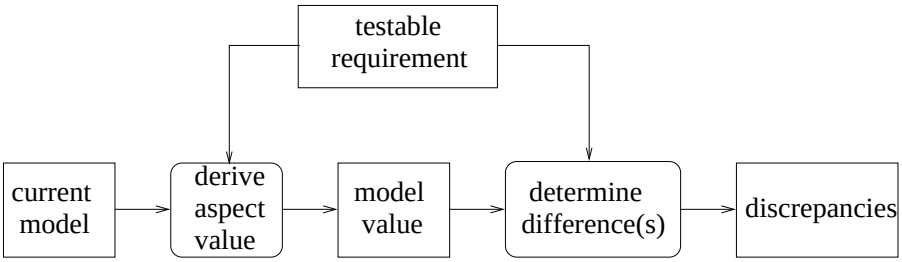


Figure 5.8: Data flow in the assessment task.

Example 5.5 *Suppose that we have a requirement stating that one of the outputs of a model should be tuned to a specific measured data set. Then the model is simulated to obtain its numerical response. Subsequently, different features of this simulated response, like its settling time, rise time, maximum overshoot etc, are compared to related features of the measured data set. This might result in a number of discrepancies, for example, the discrepancy MaximumOvershoot(e1, TooLarge:5 %) stating that the maximum overshoot of variable e1 is larger than the maximum overshoot derived from the measured data set with a difference of 5 percent.*

One of the most significant features of our approach to assessment is that an extensive set of modeling requirements can be automatically assessed. This set includes requirements on different aspects of model structure as well as requirements on (qualitative or quantitative) system behavior and response. This provides the human modeler with a large range of model requirements in which demands can be expressed, thereby allowing him a larger influence on the modeling process. To assess this wide range of model requirements we use a number of different assessment methods, including causal analysis, numerical simulation and mathematical analysis. All assessment techniques used are implementations of accepted techniques in physical systems engineering and control engineering.

5.6 Adaptation

Adaptation starts when one or more discrepancies have been detected in the assessment phase. The method of using repair plans decomposes the adaptation task into three subtasks: finding all suitable repair plans, executing these repair plans and selecting the best result. The set of discrepancies is used to find suitable repair plans for solving the current problems. All repair plans that may help to solve at least one of the current discrepancies are selected from the repository of possible repair plans. The selected repair plans are then executed, yielding a number of possible modifications of the model or the requirement set. Possible modifications are ranked based on the number of discrepancies they have helped to solve, but selection of the 'best' alternative is usually performed by the human modeler. This task decomposition of adaptation is presented in Figure 5.9.

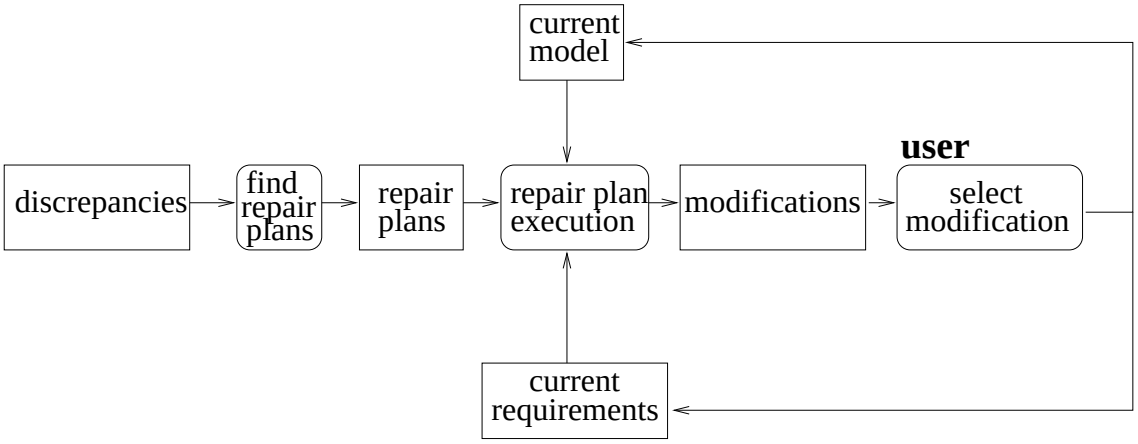


Figure 5.9: Data flow in the adaptation task.

Each repair plan represents a *repair tactic* for solving one or more model-requirement discrepancies, and contains two types of general instructions: instructions on how to find the part(s) to be modified and instructions on how to modify these part(s) (see also section 5.4). In order to execute a selected repair plan, our approach first tries to find foci by applying the general focus conditions in the repair plan. When the part(s) to be adapted have been found, the sequence of elementary actions in the selected repair plan is applied to the current model in order to repair (some of) the model-requirement discrepancies. This results in one or more modifications. Each modification yields either a modified model structure or a modified sets of requirements. Selection of the best alternative is left to the user.

One of the most important features of our approach to adaptation is the fact that we have shown that there are types of model revision that can do without the availability of a large

library of models or model fragments. In most of our repair tactics, the adaptation knowledge typically refers to general engineering and mathematical considerations about model structure, and its relation to suitable analysis methods. Another important feature of our approach is that, contrary to most automated modeling approaches, not only the physical model itself can be adapted, but also the requirements indirectly posed on the model by the analysis method used. This is possible because the method used to perform specific forms of analysis on the model is part of the requirement set. In this way we can guide the human modeler not only in constructing a proper model but also in analyzing this model with the proper analysis method.

5.7 Assessment and adaptation: an illustrative example

In this section we will illustrate our approach to assessment and adaptation in model revision. Figure 5.10 presents a possible set of hierarchical, testable requirements for model revision (in fact, this is exactly the set of requirements derived at the end of the example session presented in the previous chapter, in Figure 4.8).

Requirements

Settling Time (yes, position(car1))

QualitativeBehaviour(OscillatoryDamped, position(car1))

SystemContainsDamping(yes, global)

MinimumModelOrder(2, global)

Method = (simulation, RungeKutta4)

method features stop-time : 10 seconds
step-size : 0.1 seconds

SystemSelfContained(yes, global)

BondgraphComplete(yes, global)

ZeroOrderPathsAllowed(no, global)

Figure 5.10: Set of requirements on the railroad cars system.

The physical model that we use in this example session is the same as the one used in the previous chapters and consists of a string of railroad cars impacting on a snubber. An initial model for this system is shown in Figure 5.11.

Step 1 007 assesses the initial model of Figure 5.11 with respect to the set of testable requirements (depicted in Figure 5.10 in italic script), using a number of different assessment methods: Self-containment (Iwasaki & Simon, 1986; Iwasaki, 1988) of a system can be assessed by causality assignment (Karnopp *et al.*, 1990) in the bond graph: no conflicts

may occur in assigning causality. The presence of zero-order paths (Dijk & Breedveld, 1991) can also be determined by causal analysis. The presence of damping and the minimum model order of the system can be assessed by respectively simple examination (to see whether an R-element representing damping is present) and causal analysis (Rosenberg & Andry, 1979; Brown, 1972) to determine the model order.

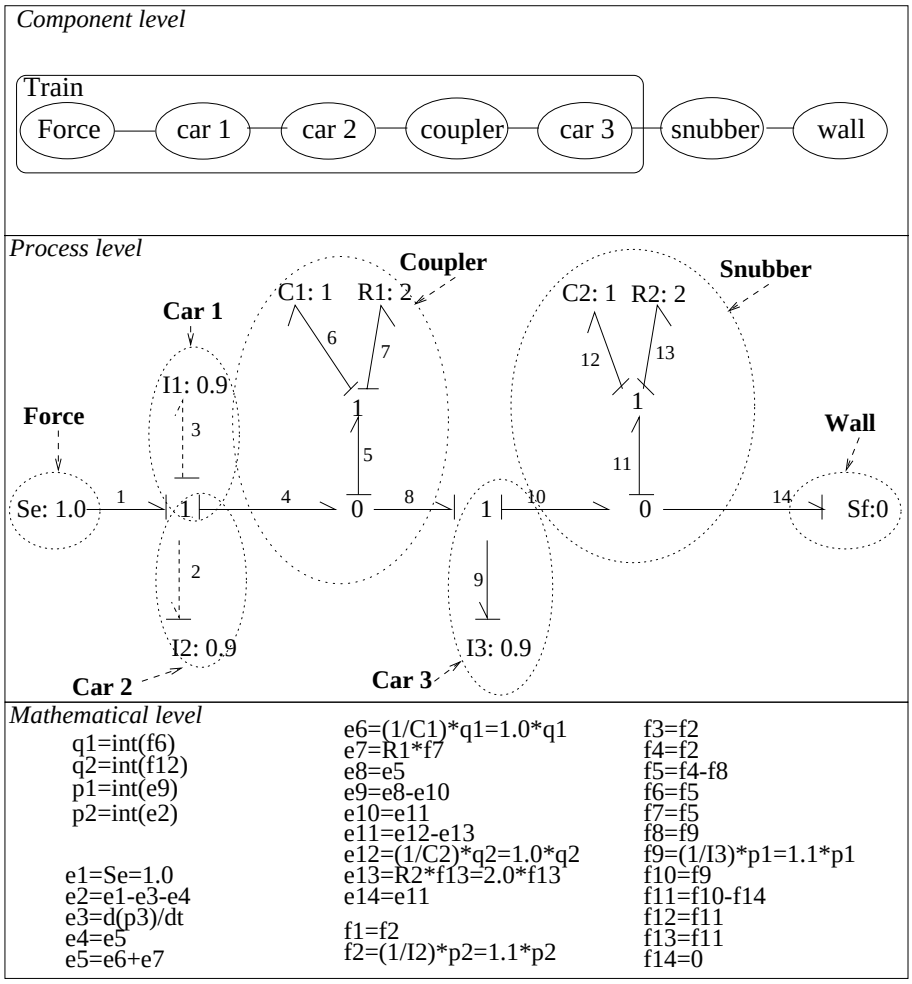


Figure 5.11: Example physical system: a string of railroad cars impacting on a snubber.

The requirements on self-containment, model order and model damping turn out to be satisfied. However, the requirement stating that there should be no zero order paths present in the model is violated, since the storage element I1 in Figure 5.11 is in derivative causality. Derivative causality in this example is due to a zero-order loop between storage elements, in this case between two inertances: I2 with integral causality and I1 with derivative causality. Due to this derivative causality, simulation with an explicit state-space simulation method like RungeKutta can not be performed. As a result, the settling time can not be determined. The resulting set of discrepancies consists of: ZeroOrderPathsPre-

sent(2I), RungeKuttaSimulation(can not be performed) and SettlingTime(can not be determined).

Step 2 Now 007 starts looking for applicable repair plans. All repair plans that may help to solve at least one of the current discrepancies are selected from the repository of repair plans. The first repair plan in Figure 5.12 represents a modification of the simulation method used for simulation, while the other four repair plans represent modifications of the current physical model itself. The first suggested repair plan in Figure 5.12 represents a modification of the method used to derive the settling time of the model, and therefore a modification of the current requirement set. The last four repair plans are implementations of theories described in (Dijk & Breedveld, 1991), and their aim is to solve the problem of zero-order paths if the numerical solver can not handle them. This can however be done in several ways: either by adding additional physical processes within the loop, as in the 'StiffCompliance' and 'ParasiticElement' approaches, by deleting one of the physical processes from the loop, as in the 'Delete Smallest I element' approach, or by replacing the two physical processes with a single, combined physical process, as in the 'Transfer dependent inertance' approach. All five repair plans are presented to the user.

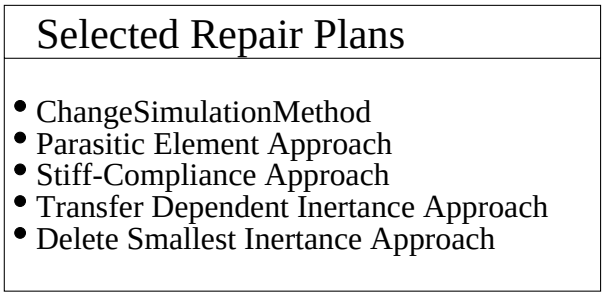


Figure 5.12: Repair suggestions produced by 007*. Each of these repairs may help to solve one or more of the current discrepancies.*

Step 3 The modeler now makes a choice between these repair plans, and thereby starts the automatic execution of the repair plan selected. Suppose the repair plan 'Transfer dependent inertance' is chosen. This repair plan is shown in Figure 5.13.

Step 4 Now 007 starts the automatic execution of the repair plan 'Transfer dependent inertance'. To execute this repair plan 007 first tries to find a *focus*. In this case it looks for a linear zero-order causal loop between two inertances in the physical model, following the instructions in the repair plan. This requires some causal analysis to determine which causal paths are present in the system. As a result, the loop between the two inertances I1 and I2, indicated by the dashed bonds in Figure 5.11, is found as a focus.

Repair plan Transfer-Dependent-Inertance-Approach

"model two dependent inertances (I) as a single independent inertance (I) (based on (van Dijk en Breedveld [1991])"

Violation List:

ZeroOrderPathsPresent(Class:2I)

"this repair plan might help to solve problems of zero-order paths between two inertances"

Focus conditions:

CausalLoop(X)

LoopOrder(X,0)

LoopType(X,2I)

Linear(X)

"find a linear zero-order causal loop between two inertances"

Abstract Actions:

Process-Delete(I_deriv)

"delete the inertance with derivative causality (I_deriv) on the process level"

Process-Replace(I_int, I_new)

"replace the inertance with integral causality (I_int) with a new inertance (I_new) on the process level"

Process-Parameter-Compute(I_new, I_int+I_deriv*factor)

"compute the value for the new I, using the values of both old Is and a factor computed from the loop connecting them"

Compose(parent(I_int), parent(I_deriv), compNew)

"replace the two cars by a single car"

Figure 5.13: Example of a repair plan. This repair plan solves the problem of a zero-order loop between two inertances by modeling the two inertances as a single independent inertance.

Step 5 Now 007 executes the elementary actions in the repair plan with respect to this focus. The two masses I1 and I2 are modeled as a single mass $I_{new} = I1 + I2$ on the process level, in order to remove the static coupling between these two masses. This also means that the functional components car1 and car2 are mapped to a single physical process I_{new}. The result of performing this repair plan is shown in Figure 5.14. The choice of another repair plan may result in, for example, keeping the two cars as separate masses, but replacing the rigid coupling with a very fast dynamic coupling.

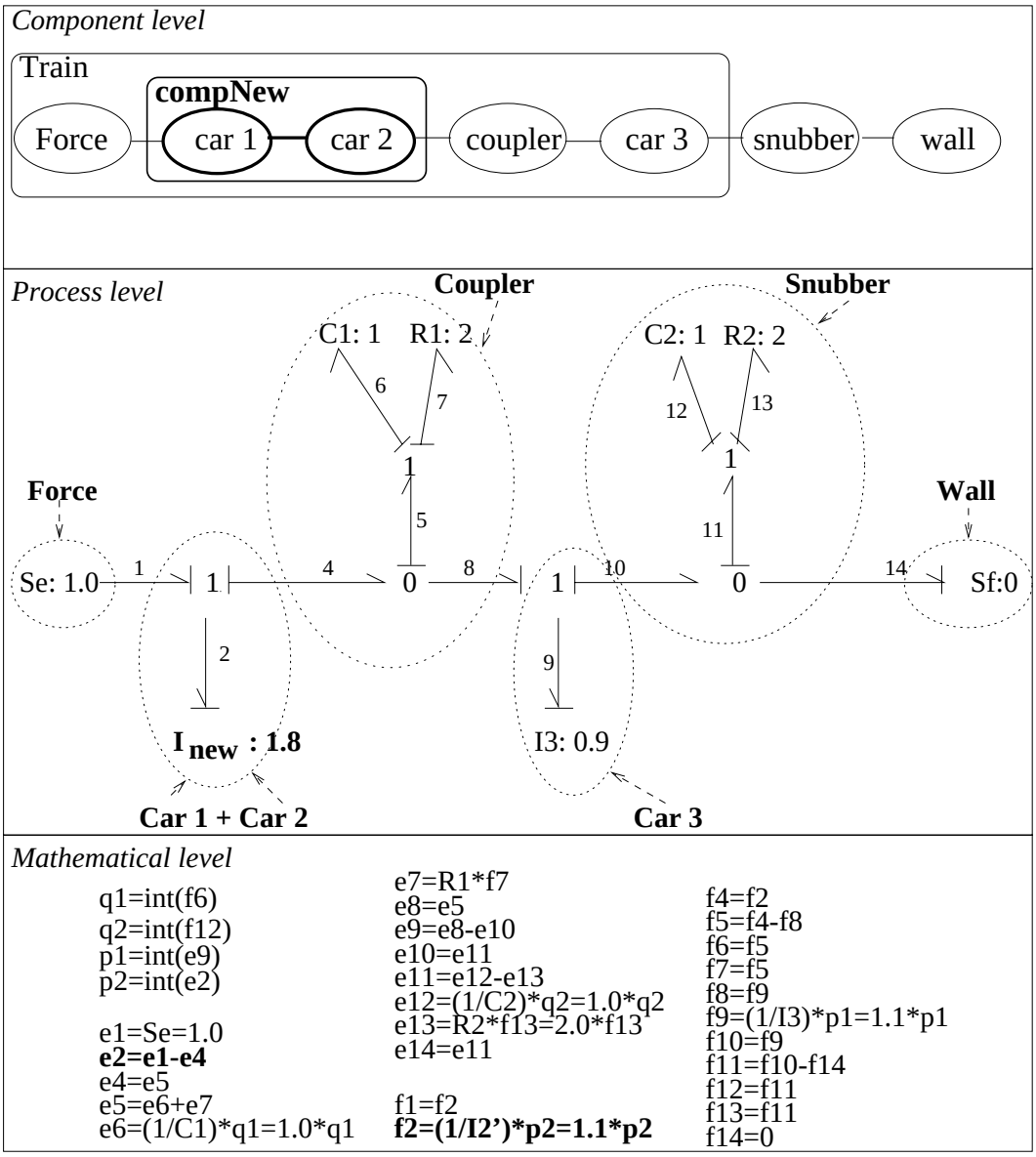


Figure 5.14: Modified model of the railroad car system, after execution of the repair plan 'Transfer dependent storage element'.

Step 6 The 007 system again performs an assessment, and finds that numerical simulation can now be performed. This means that the settling time can now be automatically determined by examining the simulation results.

Step 7 However, the modeler decides that he does not like the fact that the two cars are no longer available separately. Therefore, he retracts the modification and the model is again as it was in Figure 5.11 with the possible repairs represented in Figure 5.12. The modeler now makes a different choice between these repair plans, and decides to choose the repair plan 'ChangeSimulationMethod'. Instead of adapting the physical model itself,

this repair plan tries to adapt the method used for simulation. This repair plan is depicted in Figure 5.15.

Repair plan Change-Simulation-Method

"change the method used for simulation to one that can handle the presence of zero-order paths"

Violation List:

Runge-KuttaSimulation(CanNotBePerformed)

EulerSimulation(CanNotBePerformed)

ZeroOrderPathsPresent(all classes)

"this repair plan might help to solve problems of (state-space) simulation not being possible due to zero-order paths"

Focus conditions:

Ancestor(X,ZeroOrderPathsAllowed(no))

Type(X, Method)

"find the method requirement X from which the requirement 'ZeroOrderPathsAllowed(no) was derived"

Abstract Actions:

Select-AlternativeMethod(Y)

Not(hasChild(Y,ZeroOrderPathsAllowed(no))

AddToGoal(Y)

AddToGoal(children(Y))

"select an alternative method that can handle zero-order paths and add it to the requirement set, together with its children (if any)"

RemoveRequirement(X)

RemoveRequirements(children(X))

"remove the original focus requirement as well as all requirements derived from this requirement"

Figure 5.15: Another example of a repair plan. This repair plan solves the problems of simulation not being possible due to zero-order loops by selecting an alternative simulation method.

Step 8 Now 007 starts the automatic execution of the repair plan 'ChangeMethod'. To execute this repair plan 007 first tries to find a *focus*. In this case it uses the hierarchical representation of requirements, depicted in Figure 5.10, to find the analysis method responsible for the requirement 'ZeroOrderPathsAllowed(no)'. Finding this method requires tracing all ancestors of 'ZeroOrderPathsAllowed(no)' and selecting from these a requirement representing the method used to simulate the model. In the current requirement set, 007 finds the requirement 'RungeKutta4' as the sole focus.

Step 9 Now 007 executes the actions in the repair plan with respect to this focus. Alternative methods for prediction of the settling time are e.g. the analytical method, and

the simulation methods Euler, Adams-Bashfort and Backward Differentiation Formula (BDF). From these 007 selects both the analytical method and the Backwards Differentiation Formula (BDF) method, since they are the only ones that do not require the absence of zero order paths in the model. This means that two modifications are executed. In both, the original RungeKutta method and the requirements it posed on model structure are deleted from the requirement set, and substituted with either the BDF method or the analytical method, each with their own (default) method features. Also, the restrictions posed on model structure and behavior imposed by the new analysis method are added to both the new requirement sets. Both requirement sets are presented to the human modeler. He chooses the BDF simulation method. Now the consistency and completeness of the new sets of requirements is checked. Since no inconsistency or incompleteness is detected, Figure 5.16 shows the final result of these manipulations on the requirement set, with default values substituted for the BDF methods features of absolute and relative integration error.

Requirements
Settling Time (yes, position(car1))
QualitativeBehaviour(OscillatoryDamped, position(car1))
SystemContainsDamping(yes, global)
MinimumModelOrder(2, global)
Method = (simulation, BackwDiffForm)
method features
stop-time : 10 seconds
absolute integration error : 0.01
relative integration error : 0.001
SystemSelfContained(yes, global)
BondgraphComplete(yes, global)

Figure 5.16: Modified set of requirements, after execution of the repair plan 'Change Simulation Method'.

Step 10 The 007 system again performs an assessment with the new set of requirements. Since simulation can now be performed, the settling time can now be determined from the simulated results and there are no more discrepancies. The modeler now accepts the new (changed) requirements and the new (unchanged) model, and the assessment and adaptation session is at an end.

This example session has illustrated a number of the possibilities of the 007 system in assessment and adaptation of physical models. In chapter 6 another example of a (parametric) repair plan used for model identification will be discussed in detail.

5.8 Discussion and related research

There are a large number of modeling systems employing some form of assessment and adaptation in the model construction process. In this section we will concentrate our comparison on the use of assessment and adaptation in various automated modeling approaches. We will point out some important differences and similarities between these approaches and our own approach. For a more thorough description of general aspects of these modeling approaches, the reader is referred to chapter 2.

Many automated modeling approaches focus on (intelligent) selection from a library of reusable models (the graphs-of-models approach, (Addanki *et al.*, 1991)) or model fragments (compositional modeling, e.g. (Nayak & Joskowicz, 1996; Smith *et al.*, 1996)). They do so by employing physically oriented assumptions. A contribution of the present work is that certain classes of automated revisions can be carried out *directly* within a physical model, without the need for an extensive library of model fragments: a set of basic well-defined model elements is the only necessary support for this form of repair. Similar approaches in the bond graph paradigm are described in (Wilson & Stein, 1995) and (Amsterdam, 1992).

In our approach we have noticed that these 'constructional' repairs are typically based on generic model form or structure considerations, rather than physics-oriented assumptions. Of course, we do not at all imply that the latter are less important. Our use of repair plans allows us to solve a number of concrete modeling problems without having to resort to a complete library of models or model fragments. Current repair plans do not yet reason explicitly with physically oriented modeling assumptions. However, we think that the current theory can be extended to include some form of reasoning about modeling assumptions. This would require a new family of repair plans, linking model-requirement discrepancies to differences in basic modeling assumptions (e.g. friction included/excluded), instead of directly to changes in model structure and content (like 'add R-element'). The purpose of this exercise would be to make repairs more 'intelligent', and able to solve a larger range of modeling problems.

A model revision approach closely related to our work is the KA system (Goel, 1996). Differences lie in the content as well as the representation of models and repair plans. Goel's approach is more in the spirit of qualitative and functional reasoning as studied in Artificial Intelligence, whereas our approach originates from a knowledge-based view on more conventional engineering practices. At a more detailed level, this does lead to

differences in task decomposition and applicable problem-solving methods. Also very close to the 007 system is the modeling assistant described in (Finn, 1993). Here, the main difference lies in the extent of automation provided in adapting physical models. Finn's modeling assistant suggests useful model revisions by using problem traces, but leaves the actual execution of the revisions to the modeler, while 007 both suggests and automatically executes model revisions.

The nature of the repair plans generally depends on the domain. This aspect of the domain dependency of repair plans naturally leads to the need for thorough expert knowledge acquisition. In our approach, the adaptation knowledge typically refers to general engineering and mathematical considerations about physical model form or structure, and we have outlined how it can be formulated in a generic and reusable way in the form of repair plans. Model revisions shown in this chapter mainly involved simplification of the model being redesigned. However, model revision steps can also introduce additional detail not present in the initial model, such as the inclusion of secondary processes (energy dissipation, for example). In fact, the 007 system comprises repair plans for simplification, approximation, expansion as well as parametric adaptation of simulation models. Partly as a result of the chosen representation, which exploits the analogies between different fields in physics, our repair plans turn out to be generic across domains, in the sense that they are (re)usable for multidisciplinary physical systems. Application areas we are dealing with are mechatronic and thermodynamic systems (Borst *et al.*, 1997).

Assessment and adaptation is usually based on a set of (implicit or explicit) requirements which should be satisfied. Typical requirements in model revision are differences in device structure (Goel, 1996), internal model consistency (Smith *et al.*, 1996), model simplicity (Nayak & Joskowicz, 1996; Ling *et al.*, 1993) and model behavior (either qualitative (Amsterdam, 1992; Smith *et al.*, 1996; Nayak & Joskowicz, 1996) or quantitative (Addanki *et al.*, 1991; Wilson & Stein, 1995)). Most automated modeling approaches handle only a small number of these requirements explicitly. 007 comprises repair plans for solving some concrete modeling problems relating to model structure and (quantitative or qualitative) model behavior. The model revision tasks carried out range from parametric to structural redesign.

To our knowledge, none of the approaches in automated modeling extend the adaptation process to include adaptation of the analysis method and its features, although this is standard practice in engineering disciplines. Other important differences between our approach and most other automated modeling approaches lie in the range of possible modeling re-

quirements which can be automatically repaired, and in our use of both qualitative and quantitative methods for assessment and adaptation.

5.9 Conclusions

In this chapter we have presented a task model for assessment and adaptation in model revision. The objective of assessment in model revision is to check whether the current model satisfies the set of testable requirements derived in the previous chapter, and furthermore to determine in which way the model differs from the requirements posed upon it. The objective of adaptation in model revision is to determine which revisions of the model are most appropriate, given the differences between the current model and the posed requirements. These are the main contributions of our approach to assessment and adaptation:

- We have shown that there are types of automatic model revision that can do without the availability of a large library of models or model fragments. All that is required is a small set of basic (bond graph) elements. The types of model revision possible with this approach range from structural to parametric adaptations.
- We have outlined how adaptation knowledge based on general engineering and mathematical considerations about model structure can be represented in a generic and reusable way in *repair plans*. Since diagnosis and repair tasks in model revision are often strongly interdependent, and their combination depends on the modeling problems to be solved, in our approach both diagnosis and repair actions are integrated in a single repair plan.
- Adaptations do not necessarily have to be restricted to the physical model itself. Due to our explicit representation of requirements, not only the physical model itself but also the analysis method used to derive results from the model can be adapted.
- The techniques described in this chapter can be used to solve different types of problems. In this chapter we have shown that certain problems of model structure, often caused by the constraints posed on the model by the use of a specific analysis method, can be solved by means of repair plans. In the next chapter we will show that repair plans can also be used to solve problems of quantitative model behavior. The range of problems to be solved could be extended by including some form of reasoning with modeling assumptions in repair plans. However, this would require a new fam-

ily of repair plans, linking model-requirement discrepancies directly to differences in basic modeling assumptions.

- We have shown that a structured representation of physical models, like the bond graph representation, facilitates automated model revision. On the other hand, our approach does not depend on this specific representation. Many engineering domains have their own diagrammatic notations (for example, block diagrams in control engineering) which can also be exploited. It is true, however, that for automated model revision network flow and similar diagrammatic notations are generally easier to handle than plain sets of mathematical constraint equations.

We have shown how these features result in an semi-automated model revision system 007 with clearly-defined interaction points with the human modeler, and presented an example of its operation. The next chapter will present a parametric repair tactic, showing that repair tactics are not restricted to structural adaptation but can also be used to solve complex numerical problems.

Chapter 6

Parametric identification of physical models

Parametric identification involves determining model parameter values given experimental data on system response. Transient system response can be described by means of qualitative engineering abstractions, forming an abstract description of system response. These abstractions can either be (automatically) derived from experimental data and used for parameter identification, or they can be defined as required design values and used in the detailed design phase of (controlled) dynamic systems.

In this chapter we show that techniques from traditional engineering disciplines and more knowledge-based techniques can be combined to perform parametric system identification. We describe an algorithm for parametric identification based on an abstract description of transient system response. The technique presented in this chapter is another example of a (parametric) repair tactic in model revision, and shows that the use of repair tactics in model revision is not restricted to solving non-numerical problems by structural model adaptation, as presented in the previous chapter, but can also be employed in solving complex numerical problems.

6.1 Introduction

Modeling and simulation is intensively used in the design and analysis of physical systems. The models constructed during this process are often based on physical insight, and parameters in these models correspond to physical system parameters. Often, these

models will be partially-known (Gawthrop *et al.*, 1992), in the sense that both the system structure and some of the parameter values are known from the original design or from first principles, but other parameter values can only be accurately identified from experimental test data. For example, consider a tank of known dimensions containing a fluid of known density but whose outflow has an unknown discharge coefficient. In these situations, prior structural and parameter information should be incorporated in parameter identification, to focus attention on the unknown system parameters. Traditional (black-box) identification techniques (see e.g. (Ljung, 1987)) are often unable to incorporate prior parametric information, and parameters in these black-box models usually do not correspond to physically meaningful system parameters. Our interest is in performing parameter identification of partially-known physical models from an abstract description of transient system behavior. The basic idea to achieve this is to employ *engineering abstractions* of transient system behavior.

In many engineering domains, data on the transient behavior of a dynamic system is captured in the form of *transient performance specifications*, like the time it takes a system to 'settle' after a sudden change in its input values. These specifications form an abstract description of the response of a dynamic system to a specific input, and yet provide enough information to be useful in the analysis and (detailed) design of a dynamic system. (Palm, 1983) lists a number of these transient performance specifications, e.g. the time necessary for the response to reach its maximum value (called peak time) or the time necessary for the response to stay within some specific limits (called settling time).

In this chapter we describe an algorithm for *model parameter identification* which uses these transient performance specifications to estimate a single unknown physical parameter of a physical (simulation) model. The algorithm employs quantitative, qualitative and knowledge-based techniques; it allows parametric adaptation of numerical physical models, suitable for conventional engineering applications, based on an abstract description of system behavior. First, model reduction based on separation of *time-scales* is used to estimate general model characteristics like its natural frequency ω_n and its damping ratio ξ as a function of (some of the) model parameter values. However, the analytical relation between the unknown model parameter and the value of transient performance specifications of output behavior is in general unknown. Therefore, *approximate functions* which relate transient performance characteristics directly to the general model characteristics natural frequency ω_n and damping ratio ξ are used to compute a new value for the unknown parameter. This results in a first, roughly estimated parameter value. Since results with this one-shot method are rather inaccurate, a standard *iteration method* is used to it-

eratively adapt the value of the unknown parameter until the model behavior matches the experimental data.

In section 6.2 we will first introduce the concept of behavioral abstractions used in engineering, and describe the specific transient performance specifications used in our parametric identification approach. In this section, we also introduce the approximate functions which relate transient performance specifications on output response directly to general model characteristics like natural frequency and damping ratio. In section 6.3 we will introduce the subject of system identification. We will also show how our approach fits in the general scheme of system identification. Then, in sections 6.4 to 6.7, we describe our approach to model parameter identification, and illustrate it with a simulated example. In section 6.8 we present test results on simulated model data which indicate how well the method performs in different situations. In section 6.9 we describe the limitations and possible extensions of our approach, and compare it with some other (quantitative, qualitative and knowledge-based) system identification approaches. Finally, in section 6.10, we present our conclusions.

6.2 Engineering abstractions of transient response

There are many different ways to describe the response of a physical system. What kind of description is preferable depends on the context and purpose for which the description is to be used. For example in the early phases of design highly abstract (e.g. qualitative or causal) descriptions might be helpful, where in later, more detailed, design phases quantitative descriptions provide more suitable information. The transient response of a system in response to a rapid change in the input (also called the step response) is an important aspect in system identification. This transient response can be described by a time series, but it can also be characterized in a more abstract manner by well-defined abstract *transient performance specifications* derived from this time series (depicted in Figure 6.1), like the time necessary for the response to reach its maximum value (peak time) or the time necessary for the response to stay within some specific limits (settling time). These characteristics are well-defined and used in many engineering domains to analyze and (re)design the performance of dynamic systems. Together they form an engineering abstraction of transient system response.

(Palm, 1983) distinguishes the following transient performance specifications to describe the transient response of physical systems to a step-response : rise time, peak time, de-

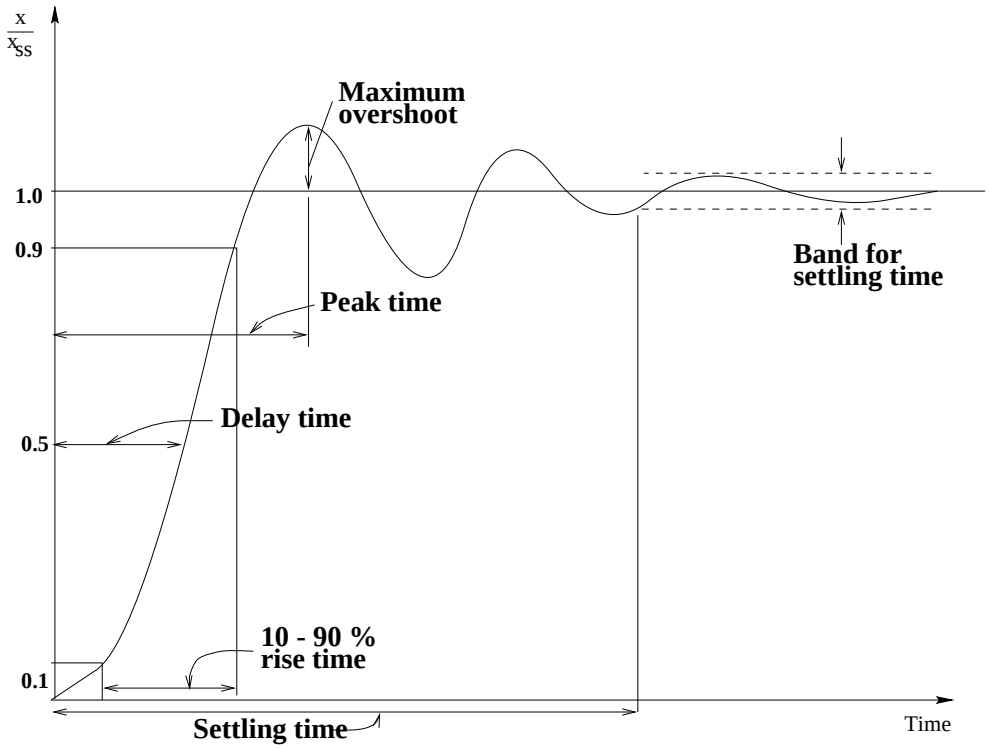


Figure 6.1: Transient response specifications based on step response [Palm, 1983].

lay time, settling time and maximum overshoot (all depicted in Figure 6.1). These performance specifications are relatively easy to obtain from an experimentally determined or simulated step-response time series, but in general difficult to determine analytically from the set of differential and algebraic equations, especially for higher-order models. However, for small (second-order) models approximate formulas, relating the transient performance characteristics to system characteristics like damping ratio (ξ) and natural frequency (ω_n), can be deduced.

- Peak time τ_p: the time at which the maximum deviation of the response above its final value (called maximum overshoot) occurs. For a second order system the peak time can be described as a function of the damping ratio ξ and the natural frequency ω_n by the following analytical function: $\tau_p = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$, $0 \leq \xi < 1$.

- Rise time τ_r: the time required for the output to rise from 10 % to 90 % of its final value. Even for a second-order system the rise-time can not be expressed as a proper function of damping ratio ξ and natural frequency ω_n, but (Palm, 1983) shows that τ_r can be approximated by the following straight-line approximation: $\tau_r \approx \frac{0.8+2.5\xi}{\omega_n}$, $0 \leq \xi \leq 1$.

- Delay time τ_d: the time required for the response to reach 50 % of its final value. Even for a second order system the delay time can not be expressed as a proper function of damping ratio ξ and natural frequency ω_n, but (Palm, 1983) shows that τ_d can be approximated by the following straight-line approximation: $\tau_d \approx \frac{1+0.7\xi}{\omega_n}$, $0 \leq \xi \leq 1$.

- Settling time τ_s: the time required for the oscillations to stay within 5 % of the final steady-state value. For a second order system the settling time can be found from the following approximate expression: $\tau_s \approx \frac{3}{\xi\omega_n}$, $0 < \xi \leq 1$.

- Percentage Maximum Overshoot: the maximum deviation of the output (x_{max}) above its steady-state value (x_{ss}) as a percentage of this steady-state value. For a second order system the percentage maximum overshoot can be described as a function of the damping ratio ξ by the following analytical function: $pMaxO = \frac{x_{max}-x_{ss}}{x_{max}}100 = 100e^{\frac{-\pi\xi}{\sqrt{1-\xi^2}}}$

The value of each of these transient response specifications can automatically be determined from an (experimental or simulated) time series and used for parameter identification. However, they can also be specified independently as intended design values and used for the purpose of detailed design of a dynamic system. In later sections, we will show that general system characteristics like natural frequency and damping ratio, mentioned in the approximate functions described above, can be related to specific physical parameters in the simulation model by making use of model reduction based on separation of time-scales. This allows us to use the approximate functions to (re)compute the value of an unknown model parameter given the value of one of the transient-performance specifications.

6.3 System identification

System identification deals with the problem of building and complementing mathematical models of dynamical systems by using experimental data (Ljung, 1987). System identification is used extensively in the design of regulators used to control and regulate system response. For example, in adaptive and optimal control a recursively identified (parametric) model of the system to be controlled is used directly for on-line tuning of system regulators (see e.g. (Kosut *et al.*, 1992)). System identification methods can be characterized by four characteristics (Astrom & Eykhoff, 1971):

- Class of models used. Models used in identification can be either *white box*, *grey box* or *black box models*. *Black box* models are not based on any physical insight. Parameters in these type of models usually do not correspond to physical system parameters. Well-known examples of classes of black box models are transfer functions, ARX and ARMAX models. *White box* models are based on physical insight of what is happening in the system. Parameters in these models correspond to physical system parameters. *Grey box* models, also called partially-known models (Gawthrop *et al.*, 1992), are models in which some but not all of the physical phenomena are known, i.e. some system parameters can be obtained directly from physical insight but others will have to be identified from experimental test data. This information can be used to restrict parametric identification to parameters which are either unknown or can only be determined with a very large margin of error. Our approach is an example of parametric identification of grey box models.

- Class of input signals used. Experimental data used in identification usually consists of the response of a dynamic system to a specific class of input signals. Input signals can either be periodical (e.g., sinusoidal input) or non-periodical (e.g. step input or impulse input). In our approach a step-input is used to excite the system.

- Criterion used. Identification needs a criterion to determine how well model output conforms to the experimental data given. When using time-response data, this experimental data is often given in the form of a time series. An example of a (simple) criterion used in system identification is simply to take the sum of the differences between model output $\hat{y}$ and experimental test data y ($\sum_1^N |\hat{y}_n - y_n|$). In our approach, we will not use time series, but an engineering abstraction of transient system response, described in terms of values for one or more of the transient performance specifications discussed in the previous section. This means that our optimization criterion will not be based on comparing time series $\hat{y}$ and y directly, but on comparing the experimental measured ($\tau_{x,exp}$) and model ($\tau_{x,model}$) value of the transient performance specification τ_x derived from the original time series. Ergo, our optimization criterion is $|\tau_{x,exp} - \tau_{x,model}|$.

- Computational aspects. Methods used for identification can be either direct, one-shot methods or iterative methods. In our approach we use a direct, one-shot method to compute a rough estimation of the unknown model parameter and a standard iterative method, called the modified Newton method (Hamming, 1962), to iteratively estimate more precise model parameter values.

6.4 Model parameter identification

In parametric identification of partially-known systems, the structure of the equations and parameters in the model is considered to be fixed, but the values of some of the parameters are unknown. The purpose of our approach to parametric identification is to determine new values for a single unknown parameter of a simulation model, so that the response of this simulation model matches the experimental response of the associated system. Both experimental and model responses are expressed in terms of the transient-response specifications discussed in the previous section. Figure 6.2 presents the data-flow in our approach to parametric identification. The method proceeds in a number of steps:

1. The method starts with a kind of *data preprocessing*, in which experimental data and model data are compared, resulting in an abstract description of their differences. This abstract description is formulated in terms of the transient response specifications discussed in the previous section. If the abstract descriptions of the experimental data and model response do not match, identification will have to take place.

2. Then, model *reduction* based on separation of *time-scales* in the bond graph representation is used to select the part of the model most relevant to its transient response. From this slowest part of the model, *general characteristics* of the model like its most significant natural frequency ω_n and damping ratio ξ can be approximated. The approximate characteristics ω_n^* and ξ^* can be determined symbolically as functions of the parameters $P_1 \ldots P_n$ in the time-scale reduced model.

3. In order for identification to succeed, the unknown parameter P_i should be one of the parameters $P_1 \ldots P_n$ in the time-scale reduced model. If this is the case, *rough tuning* uses approximate formulas to derive a first value for the single unknown model parameter P_i. These approximate formulas relate transient performance specifications τ_x to the general model characteristics ω_n and ξ. They take into account only the influence of the slowest time-scale and not the additional influences of faster time-scales. Due to the approximate nature of the formulas used, *rough tuning* will often produce a new value for P_i which does not completely remove the discrepancy between simulated model and experimental system response.

4. Therefore, *fine tuning* may be necessary. In this phase, standard iteration methods are used to iteratively compute a better value for the unknown parameter P_i.

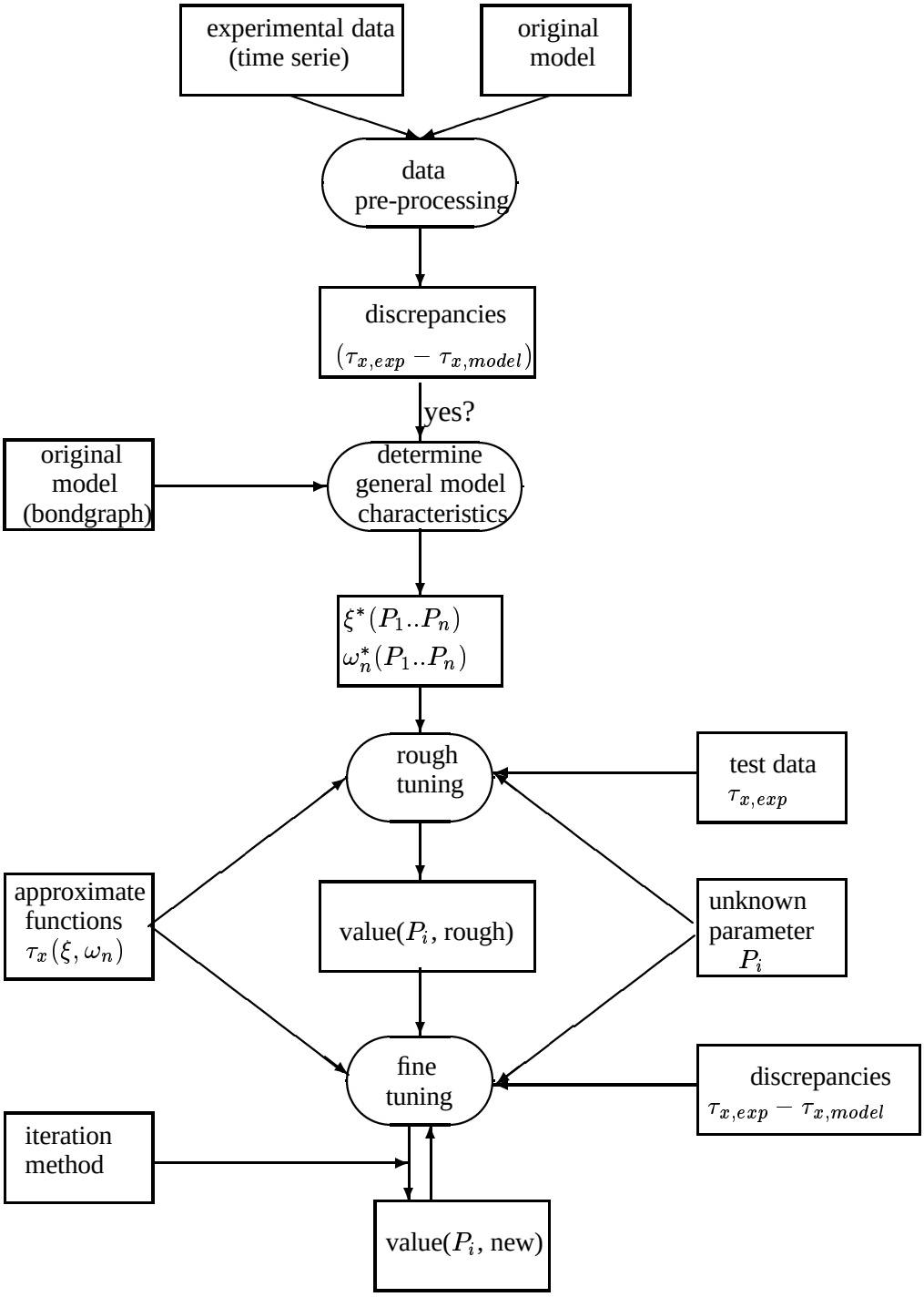


Figure 6.2: Data-flow for the parametric identification task.

6.5 Data preprocessing

In order to start parametric identification, we need to know whether the output of the current model differs from the experimental data. If this is not the case, the current model is already sufficient and parametric identification does not need to take place. This task can in our approach also be called data preprocessing, since it transforms the raw data, in the form of a simulated or experimental time series, to an abstract description of this data in terms of the transient performance specifications. This data preprocessing task is depicted in Figure 6.4. In data preprocessing the current model is simulated with a standard numerical simulation method to obtain its numerical response in the form of a time series. From this time series, an abstract description $\tau_{x,model}$ in terms of the transient performance specifications peak time, rise time, delay time, settling time, and PercentageMaximumOvershoot can be automatically obtained. The same procedure is employed on the time series data obtained from an experiment with the actual system, also yielding an abstract description $\tau_{x,exp}$ in terms of the same transient performance specifications. If these two abstract descriptions match, the model agrees (qualitatively) with the experimental data, and no parametric identification is necessary. If there are discrepancies between these two abstract descriptions $\tau_{x,exp}$ derived from experimental data and the abstract descriptions parametric identification continues.

Example 6.1 *Consider a string of railroad cars impacting on a snubber, a variation on the example used throughout this thesis (figure 6.3). Values for the masses and spring constants are assumed to be known, as well as the resistance of the snubber, but the resistance of the coupler can not easily be determined and will have to be identified from experimental data on system response. Suppose the position of car 1 is the (single) output used for identification.*

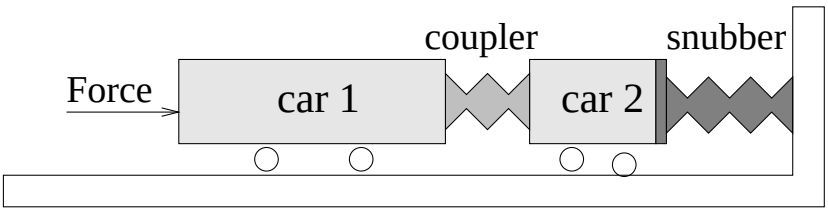


Figure 6.3: Variation on the railroad car system: two railroad cars impacting on a snubber.

The model is simulated with a standard numerical simulation method to obtain the simulated model response of the selected output variable $position(car1)$*. From this simulated response, in the form of a time series, the following abstract description is derived*

automatically: $\tau_{p,model} = 11.3, \tau_{r,model} = 4.4, \tau_{d,momodel} = 4.2, \tau_{s,model} = 37.0$ *and* $pMaxO_{model} = 43.5\%$*. Suppose that experimental data is obtained for this system, also in the form of a time series. Again the following abstract description is derived automatically:* $\tau_{p,exp} = 22.0, \tau_{r,exp} = 8.4, \tau_{d,exp} = 5.3, \tau_{s,exp} = 12.4$ *and* $pMaxO_{exp} = 0.11\%$*. It is obvious that the two specifications do not match, and a set of discrepancies can be obtained which describes the difference between the two abstract descriptions. For example, comparing the peak times of the two abstract descriptions results in the discrepancy* $\tau_{p,exp} - \tau_{p,model} = 22.0 - 11.3 = 10.7$

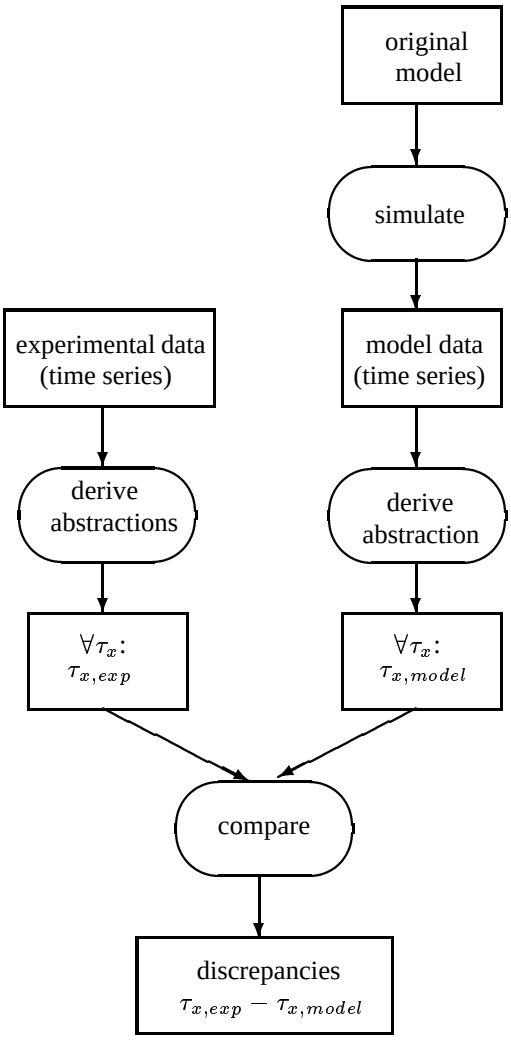


Figure 6.4: Data-flow for the data preprocessing task.

6.6 Determine general model characteristics

In large systems, there are many different contributions to oscillatory and damping aspects of system behavior: the larger the system, the more eigenfrequencies and damping influences are involved in determining system behavior. Many application examples can be presented in which the behavior of the system is influenced by both fast and slow processes, e.g. a robotic system with a slow mechanical part and a fast electronic part or chemical processes with different reaction speeds. When a system has this multi-time scale property it is often useful to separate these dynamics, to simplify the analysis or to eliminate calculation errors. In the field of AI, reasoning about time-scales is employed both in qualitative simulation (Kuipers, 1987) and in numerical analysis (Iwasaki & Bhandari, 1988). In our approach we focus on an engineering approach to model reduction based on separation of time-scales which makes use of the bond graph representation (Sueur & Dauphin-Tanguy, 1991). This approach is based on the singular perturbation theory (Kokotovic *et al.*, 1976). Both will be explained in this section (subsections 6.6.1 and 6.6.2). Separation of time-scales can be used to separate the slow part of the model from the faster part. In this section we discuss how general model characteristics like its eigenfrequency ω_n and damping ratio ξ, can be approximated by considering only the most relevant part of the model (subsection 6.6.3). Figure 6.5 represents the data flow in this subtask, which will be explained in the rest of this section.

6.6.1 Used tools: Singular perturbation method

Suppose that an $(n+m)$th order system has n *slow dominant modes* and m *fast parasitic* modes, that is n of its eigenvalues λ are $O(1)$ and the remaining m are $O(\frac{1}{\epsilon})$, where ϵ is a small positive scalar ($\epsilon \ll 1$). Typically, ϵ represents small time constants, masses, inertances etc. Then it is possible to describe the system as a singularly perturbed system (Kokotovic *et al.*, 1976)

$$\begin{aligned} \dot{X}_1 &= A_{11}X_1 + A_{12}X_2 + B_1U \\ \epsilon\dot{X}_2 &= A^*_{21}X1 + A^*_{22}X2 + B^*_2U \\ 0 < \epsilon &\ll 1 \end{aligned} \tag{6.1}$$

where $\dot{X}_1$ denotes the n slow variables and $\dot{X}_2$ denotes the m fast variables.

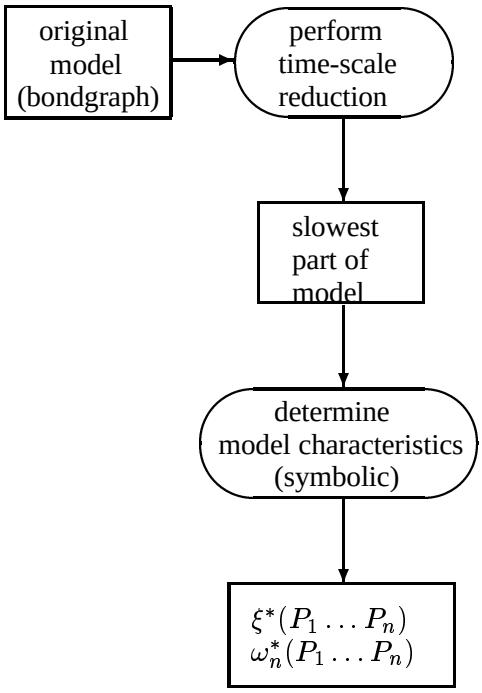


Figure 6.5: Data-flow for the determine-general-model-characteristics task. $P_1 \ldots P_n$ are parameters in the time-scale reduced model.

Assuming that A_{22} is invertible, the explicit slow reduced model (6.2) is obtained from (6.1) by formally setting $\epsilon = 0$. X_{1s} and X_{2s} denote the slow parts of variables X_1 and X_2 respectively.

$$\begin{aligned} \dot{X}_{1s} &= (A_{11} - A_{12}A_{22}^{-1}A_{21})X_{1s} + (B_1 - A_{12}A_{22}^{-1}B_2)U_s \\ X_{2s} &= -A_{22}^{-1}A_{21}X_{1s} - A_{22}^{-1}B_2U_s. \end{aligned} \tag{6.2}$$

Again assuming that A_{22} is invertible, he fast reduced model (6.3) is obtained from (6.1) by introducing a fast time-scale. X_{1f} and X_{2f} denote the fast parts of variables X_1 and X_2.

$$\begin{aligned} \dot{X}_{1f} &= 0. \\ \epsilon\dot{X}_{2f} &= A_{22}X_{2f} + B_2U_f \end{aligned} \tag{6.3}$$

From equations (6.2) and (6.3) we can get an approximation of variables X_1 and X_2: It is known from (Kokotovic *et al.*, 1976) that the contribution of the fast modes to X_1 is only

$O(\epsilon)$, and hence X_1 can be approximated by its slow behavior X_{1s}. X_2 is formed of a fast transient (X_{2f}) and a quasi steady-state (X_{2s}).

$$\begin{aligned} X_1 &\approx X_{1s}, \\ X_{1s}(t_0) &= X_1(t_0) \\ X_2 &\approx X_{2f} + X_{2s}, \\ X_{2f}(t_0) &= X_2(t_0) - X_{2s}(t_0) = X_2(t_0) + A_{22}^{-}1A_{21}X1(t_0). \end{aligned} \tag{6.4}$$

However, if A_{22} is not invertible, the time scales can not be expressed explicitly. The problem now consists of finding a meaningful choice for the slow state variables X_1 and fast state variables X_2, such that A_{22} is invertible. (Dauphin-Tanguy *et al.*, 1985) uses model-reduction based on time-scale in the bond graph representation to solve this problem. This technique is based on the observation that the global eigenvalues λ of a system can be approximated locally by considering causal loops in the bond graph representation.

6.6.2 Used Tools: Time-scale reduction in bond graphs

From the bond graph representation, discussed in chapter 3, a state-space model can be easily obtained by standard techniques (Karnopp *et al.*, 1990), by considering the energy variables p (impulse) and q (charge) associated with storage elements in integral causality as state variables, resulting for linear systems in

$$\begin{aligned} X' &= AX + BU \\ X &= \begin{bmatrix} p \\ q \end{bmatrix} \end{aligned} \tag{6.5}$$

When a system is represented as a bond graph, it is possible to approximate the eigenvalues by calculation of the *causal loop gains* (Rosenberg & Andry, 1979). Separation of slow and fast modes can be performed by comparing the gains of these loops in the bond graph model: fast modes correspond to large loop gains and slower modes to smaller loop gains.

In bond graphs causal loops can be determined by a process of consistent causality assignment. In general, causal loops are of order zero, one or two. Causal loops are zero-order

loops if either no integrating (i.e. storage) element is present in the causal loop (algebraic loops), or if there exists a causal loop between a storage element with derivative causality and a storage element with integral causality (Dijk & Breedveld, 1991). First-order loops involve a storage element and a resistance element, while second-order loops occur between two storage elements with integral causality. Zero-order loops describe fixed couplings in the system, first-order loops are associated with damping and second-order loops are associated with oscillatory behavior. Each loop has its own unique *loop gain*, which consists of a combination of the values of the elements involved in the loop and describes the numerical contribution of this loop to system behavior (Rosenberg & Andry, 1979). The transfer function for a linear bond graph can be determined by considering causal paths and causal loops, as well as their interactions (Brown, 1972).

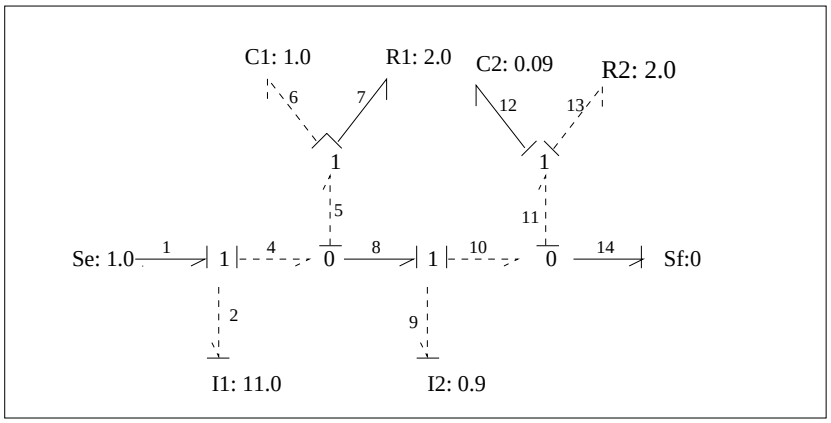


Figure 6.6: Bond graph model of the example system. The dashed bonds represent two of the causal loops in this model.

Example 6.2 *Consider the bond graph representation of our example system. In this representation a number of causal loops can be distinguished. Two of these are presented in Figure 6.6 as dashed bonds. The causal loop on the left is a second order loop: it occurs between two storage elements (C1 and I1) both of which are in integral causality. Second order loops influence the oscillating behavior of a system. The causal loop on the right is a first order loop: it involves a storage element (I2) and a resistance element (R2). First order loops influence damping in the system.*

6.6.3 Approximation of model characteristics ξ and ω_n

When the system is represented as a bond graph, causal loops can be used to approximate both the natural frequencies (complex eigenvalues) and damping influences (real eigenvalues) of a system. Fast behavior modes correspond to large loop gains and slower modes

to smaller loop gains. The transient performance specifications defined earlier (section 6.2) are mainly influenced by the slow behavioral mode(s) of system behavior, since faster modes tend to die out fast and have a relatively small impact on the form of the transient response. Thus, the smaller the loop gain of a causal loop the more influence it has on the transient performance specifications. Therefore, we approximate model characteristics like the total damping influence ξ and the natural frequency ω_n by considering the values of the smallest loop gains of first and second order loops respectively. Together these loops represent the slowest mode of system behavior.

Example 6.3 *In the bond graph of our example system of railroad cars (figure 6.6) there are 6 loops. The corresponding approximations of the complex eigenvalues, contributing to the oscillating behavior of the model, are given by the following loops of second order:*

(1) slowest part:

$\frac{1}{C1*I1} = 0.090909$ *(along bonds 2, 4, 5 and 6)*

(2) faster parts:

$\frac{1}{C1*I2} = 1.111111$ *(along bonds 5, 6, 8 and 9),*

$\frac{1}{I2*C2} = 12.345679$ *(along bonds 9, 10, 11 and 12)*

The loop whose value approximates the most significant contribution to the slow oscillating behavior of the system, i.e. models the lowest frequency, corresponds to the second order loop with the smallest loop gain and is depicted in Figure 6.6 by means of dashed bonds. The most important natural frequency ω_n^* of the system can be approximated by considering the loop gain of this loop. In the current example, the approximated natural frequency $\omega_n^* = \sqrt{\frac{1}{C1*I1}} = 0.3015$.

The contributions to damping in the system (real eigenvalues) can be approximated by considering the following first order loops:

(1) slowest part:

$\frac{R1}{I1} = 0.181818$ *(along bonds 2, 4, 5 and 7)*

(2) faster parts:

$\frac{R1}{I2} = 2.222222$ *(along bonds 5, 7, 8 and 9),*

$\frac{R2}{I2} = 2.222222$ *(along bonds 9, 10, 11 and 13)*

The loop whose value most heavily influences the damping contribution to slow system behavior corresponds to the first order loop with the smallest loop gain. The total damping ratio ξ^* of the system can be approximated by considering the loop gain of this loop and the natural frequency ω_n^*. In the current example the approximated total damping ratio $\xi^* = \frac{R1}{I1} * \frac{1}{2*\omega_n^*} = 0.30151.$

6.7 Parameter tuning

In the previous section we approximated the most significant damping ratio ξ^* and natural frequency ω_n^* of the model by examining causal loops in the bond graph. In this section we will compute a new value for a single unknown parameter P_i in this model. For this we use the approximate formulas defined in section 6.2. These formulas relate the transient performance specifications to the damping ratio ξ and frequency ω_n. For the moment, we only consider cases in which a single model parameter P_i is unknown, in which there is a single output variable Y used for identification, and in which a single value defines transient system response. The last condition means that two data sets match if they agree on a single performance specification τ_x.

Before we start computing a value for the unknown parameter P_i, we need to consider whether the unknown parameter P_i is *suitable* for identifying the output variable Y. In order to be able to compute a value for the unknown parameter P_i, two conditions need to be satisfied. First, it is essential that P_i is a parameter in the slowest part of the model. If P_i is not part of this slowest part of the model it is not possible to use the approximate formulas discussed in section 6.2. This is due to the fact that the general model characteristics ξ^* and ω_n^* can only be approximated from the slowest part of system behavior. Another condition is that P_i and the output variable Y are *causally connected*, i.e.. there is at least one causal path from P_i to Y. If P_i and Y are not causally connected, changing the value of P_i does not influence the behavior of Y. These two conditions are a specific example of the more general issue of *identifiability*: the question whether the parameters of a parameterized set of candidate models can be uniquely (globally or locally) determined from data (Ljung & Glad, 1994a; Walter, 1982; Ljung & Glad, 1994b).

If both conditions are satisfied, finding a suitable value for the unknown parameter P_i generally proceeds in two steps: First, the approximate function for the transient performance specification is used directly to compute an initial value for P_i. Since the functions used are only approximations, this initial value will usually not solve the problem completely.

Therefore, this stage is referred to as *rough tuning*. To increase the accuracy of parameter estimates, fine tuning is necessary. In *fine tuning* a standard iteration method is used to iteratively (re)compute the value of P_i. Fine tuning continues until the simulated model response $\tau_{x,model}$ matches the experimental measured data $\tau_{x,exp}$.

6.7.1 Rough tuning

Assuming that the approximate function for the performance specification τ_x is invertible with respect to P_i, a first educated guess can be obtained for the value of P_i. Using the fact that all other parameter values other than P_i are known we can express any transient performance specification τ_x as a function of P_i by substituting $\xi^*(P_1 \ldots P_n)$ and $\omega_n^*(P_1 \ldots P_n)$ into the formula for $\tau_x(\omega_n, \xi)$. This results in an expression for τ_x in which the unknown parameter P_i is the only variable. If this function is invertible with respect to P_i, inversion yields an approximate expression for the value of the unknown parameter as a function of the selected transient performance specification, $P_i(\tau_x)$. By substituting the experimental value $\tau_{x,exp}$ for τ_x, we obtain a first, rough guess for the value of the unknown parameter P_i.

Example 6.4 *In earlier examples we obtained symbolic expressions for the (approximated) general model characteristics ξ^* and ω_n^*. Now suppose that we select the transient performance specification* peak time τ_p *to abstractly describe both experimental and model data. The experimental value for the peak time $\tau_{p,exp} = 22.0$ while the value of the peak time derived from simulated model response $\tau_{p,model} = 11.3$. In order to find a first rough value for the parameter R1 we need to invert the approximate function relating the value of R1 to a value for the performance specification τ_p: $\tau_p = \frac{\pi}{\omega_n\sqrt{1-\xi^2}} \approx \frac{\pi}{\omega_n^*\sqrt{1-\xi^{*2}}} \approx \frac{\pi}{\frac{1}{C1*I1}\sqrt{1-\frac{R1}{I1}^2}} \approx \frac{\pi}{0.3015\sqrt{1-\frac{R1}{11.0}}}$. This function is invertible with respect to R1. Inversion results in the following expression: $R1 \approx \sqrt{(1-(\frac{\pi\sqrt{11}}{\tau_p})^2)*44} \approx \sqrt{(1-(\frac{\pi\sqrt{11}}{22.0})^2)*44} \approx 5.84298$. After simulating the model with the new value for R1 ($R1 = 5.84298$), the peak time derived from the simulated response $\tau_{p,model} = 19.9$. Although this is closer to the experimental value $\tau_{p,exp} = 22.0$ than the original value ($\tau_{p,model} = 11.3$), it still does not match the experimental value exactly. Therefore, we will need to perform some additional fine tuning.*

6.7.2 Fine tuning

In the previous example, rough tuning did not completely solve the discrepancies between measured and model behavior. In this section we discuss how we can use approximate functions (section 6.2) together with a standard iterative method (modified Newton (Hamming, 1962)) to compute increasingly better values for the unknown parameter P_i.

The Newton iteration method is one of the numerical methods available for computing the real zero of a function. The idea behind the Newton method is the "analytical substitution" of the local tangent line for the function itself and then the use of the zero of this line as the next approximation to the zero of the function ($x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$) (see also Figure 6.7). Although it is easy to use, the Newton method has a number of important faults: it can cycle and never converge, and it may oscillate wildly in the case of local minima. These faults can be partially compensated for by a simple device: if the new value is not closer to zero than the previously computed value, do not accept the step, but instead go back and halve the step. Repeat this until the new computed value is better than the old one. In fact, this procedure combines the Newton method with the bisection method, thus trying to combine the advantages of both. This combination is called a 'modified Newton' method (Hamming, 1962).

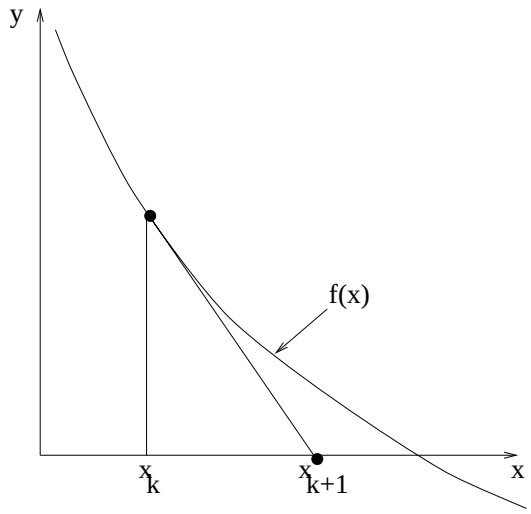


Figure 6.7: The Newton iteration method.

The aim of our approach to parametric identification is to find a value for the unknown model parameter P_i so that the transient response of the model matches the experimental transient response, in which both responses are described by a single transient performance specification τ_x. Using the fact that all other parameter values are known we can

write any τ_x as a function of P_i by substituting ξ^* and ω_n^* ,approximated by time-scale reduction in the bond graph model, into the approximate formula for τ_x. Since the resulting function $\tau_x(P_i)$ is relatively simple, we can now automatically obtain the derivative $\tau_x'(P_i)$ with any mathematical package. Now we can use this derivative to iteratively compute the new value of P_i by using the (modified) Newton iteration method: $P_{i,new} = P_{i,old} + \frac{\tau_x(P_{i,old}) - \tau_{x,exp}}{\tau_x'(P_{i,old})}$. The new model response $\tau_x(P_i, new)$ can now be obtained by simulation. Fine tuning continues until the model response matches the experimental value.

Example 6.5 *The current value of the model peak time $\tau_{p,model}$, after rough tuning, is 19.9 for $R1 = 5.84298$. Since the value of the model peak time $\tau_{p,model}$ differs from the value of the experimental peak time $\tau_{p,exp} = 22.0$, fine tuning will have to take place. To determine the amount by which the parameter R1 must be changed in each iteration we use the Newton iteration method with the formula for peak time $\tau_p(R1) = \frac{\pi}{\omega_n\sqrt{1-\xi^2}} \approx \frac{\pi}{\omega_n^*\sqrt{1-\xi^{*2}}} \approx \frac{\pi}{0.3015\sqrt{1-\frac{R1}{6.63}}}$. The derivative of this approximate function can automatically be obtained with any mathematical package like Mathematica: $\tau_p'(R1) = \frac{0.075*\Pi*R1}{(1-0.022*R1^2)^{3/2}}$. Using this formula with the (modified) Newton iteration method causes the actual peak time of the model to converge in 17 iterations to the exact value $\tau_{p,exp} = 22.0$.*

6.8 Experimental results

The parametric identification method described in this chapter has been tested on different examples, ranging in order from a simple second order system (a system with only one railroad car) to a more complicated 10th order system (a system with 5 railroad cars). All the examples were modifications of the example system shown used throughout this thesis, differing in the number of railroad cars and in the starting values for system parameters. The position of car 1 is taken as the output used for identification, and R1 is supposed to be the only unknown parameter. Tests were carried out with simulated data. For each test, 10 test values were selected uniformly from the range $0.1 < \xi^* < 0.9$, which means that nearly undamped ($\xi^* < 0.1$) and nearly over damped ($\xi^* > 0.9$) cases were excluded. From these 10 tests a mean value was computed. In order to compare the effects of using different performance specifications, results were normalized.

Our approach to parametric identification based on transient performance specifications obviously will work best if the approximate function closely resembles the actual model response. When additional time-scales are introduced, the approximate function will less closely resemble actual model response and convergence of the results will slow down,

especially if fast and slow time-scales are relatively close. Therefore, we first tested the effect of using different performance specifications for identification on a small second order system with only one railroad car. Since in this system no additional time-scales are present, and therefore no additional disturbances, it will gives us a good insight on the performance of using the different performance specifications. The results are shown in Figure 6.8.

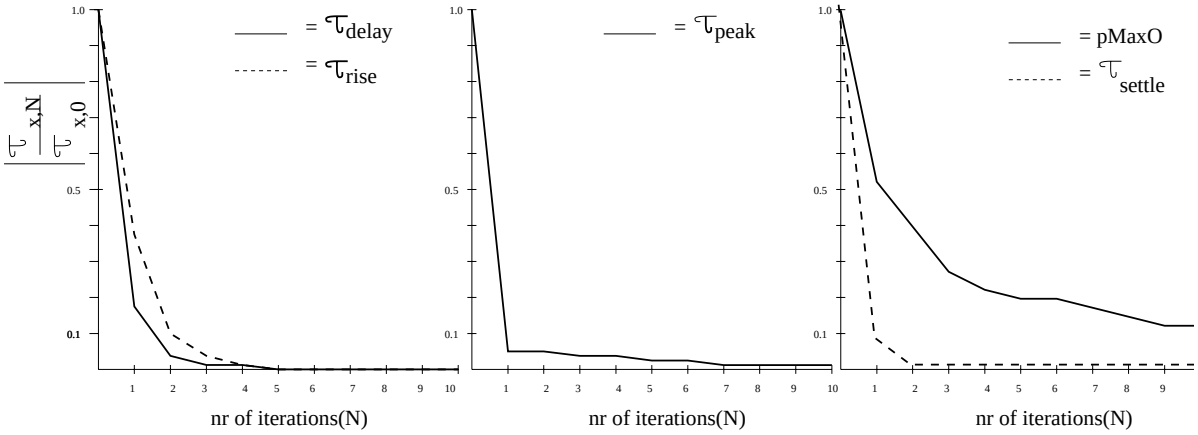


Figure 6.8: Test-results obtained for a small second order model for different performance specifications.

Rough tuning uses the approximate formula for the transient performance specification τ_x directly to compute an initial value for the unknown parameter R1. However, rough tuning can only be performed when the approximate formula is invertible with respect to the single unknown parameter. In the current set of tests, all approximate formulas were invertible as a function of the unknown parameter R1, except for the percentage maximum overshoot $pMaxO$. Therefore, results for $pMaxO$ were obtained with fine tuning only. Invertibility of each function depends both on the model structure and on the unknown parameter P_i. For example, in the current model the approximate function for the performance specification peak time τ_p is invertible for the parameter R1 but not for parameters C1 and I1.

The result of rough tuning is strongly dependent on how well the approximate formulas represent (simulated) model response: the better the approximation, the better the initial value for the selected parameter. Of the performance characteristics discussed (τ_d, τ_r, τ_s and τ_p), the formulas for the settling time τ_s and peak time τ_p provide the best approximations of simulated system response. This means that the (simulated) response for the initial value computed for R1 by rough tuning is already very close to the measured value. The approximate formulas for the performance characteristics τ_d and τ_r are less good ap-

proximations of the simulated model response, and therefore the response simulated for the initial value computed for R1 by one of these formulas is often further from the experimental response.

Fine tuning uses the approximate formulas iteratively to compute new values for the selected parameter. The speed of convergence strongly depends on the sensitivity of each performance specification to changes in parameter values. Delay τ_d and rise time τ_r are not very sensitive to differences in ξ and ω_n, while the settling τ_s and peak time τ_p are very sensitive to differences in ξ and ω_n. Therefore, it is much easier to find a value of the selected model parameter R1 such that the results match the measured delay (rise) time exactly than it is to find a value of the selected model parameter R1 such that the results match the settling (peak) time exactly. The result is that fine tuning of the settling time and peak time take a long time to converge to the exact measured value, even though the initial estimate might be close.

As discussed before, the introduction of additional time-scales will influence the accuracy of the approximate formulas negatively, and therefor the rate of convergence of identification will deteriorate, especially if time-scales are close together. To see how our approach to parametric identification scales up to larger models with time-scales closely influencing each other, we also tested the approach on a larger, more complex model tenth order model with 5 railroad cars. In this model additional time-scales close to the slowest time-scale were introduced.

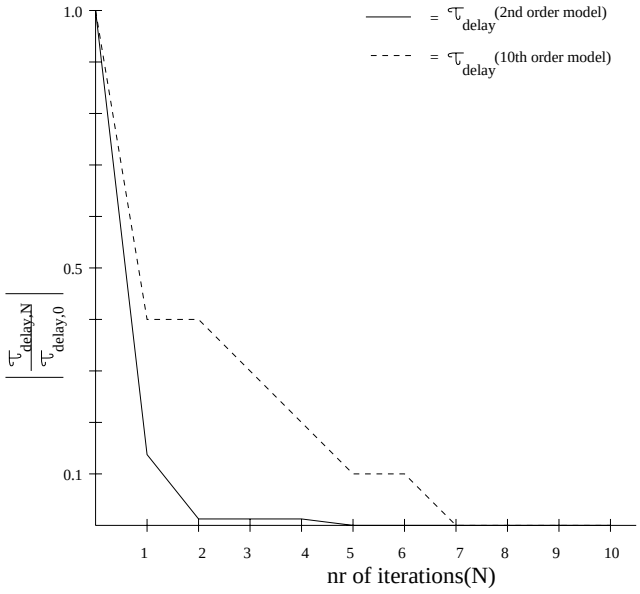


Figure 6.9: Test-results obtained for the delay time τ_{delay} of a small, second order and a larger, 10th order model.

Figure 6.9 shows a selection of the test-results for two models: a small second order model and a larger, more complex tenth order model where additional time-scales close to the desired time-scales are introduced. The delay time τ_d is used to describe transient response. We see that the number of iterations increases for larger models. However, even for a 10th order model with additional time-scales close to the most significant time-scale, model parameter identification converges to the correct experimental value in only a small number of iterations.

6.9 Discussion and related work

The transient performance specifications used in our approach to parametric identification have a few specific characteristics: they are explicit abstractions of time, they can automatically be abstracted from numerical time series data and they are widely used in engineering disciplines. In engineering applications these specifications are mainly used in detailed, parametric design or control phases instead of in earlier, more general design phases. This is also the main difference between our engineering abstractions and general notions like make, maintain, prevent and control (Franke, 1991) or notions like substance difference, substance property difference and substance location difference (Goel, 1991). Descriptions in terms of these general notions are useful in early, general phases of the design process, while functional time related engineering abstractions like ours are used in later, more detailed phases of the design or control process where time related and performance aspects play an important role. In fact, both descriptions describe different abstractions of the same behavior, and can be used complimentary.

Most of the different elements in our parametric identification approach are not new in themselves; our contribution lies precisely in using a combination of quantitative, qualitative and knowledge-based techniques to support identification from an abstract description of system behavior. Other identification approaches have discussed identification of partially-known physical models, and some of these have also used the bond graph representation as a suitable formalism to describe these models (Gawthrop *et al.*, 1992; Nagy & Ljung, 1992). The combination of time-scale reduction using bond graphs and well-defined performance specifications in our approach allows us to identify partially-known physical models using sparse, abstracted knowledge on transient system response.

Techniques for time-scale abstraction have been investigated both in AI and in traditional engineering. For example, time-scale reduction using the bond graph representation has

been investigated for both frequency (Margolis & Young, 1977) and state-space models (Sueur & Dauphin-Tanguy, 1991). For a survey of other model reduction approaches in engineering see (Fossard *et al.*, 1987). In the field of Artificial Intelligence, time-scale abstraction in both qualitative (Kuipers, 1987) and quantitative (Iwasaki, 1988) models has been investigated.

Using abstract descriptions of experimental data in system identification and controller tuning is not limited to step-response identification for state-space models. Other approaches have discussed the use of abstract features to perform identification of transfer functions (Kraus & Myron, 1984; Rake, 1980). The well-known controller tuning rules by Ziegler and Nichols (Ziegler & Nichols, 1942) are a typical example of how similar abstract features can be used for tuning PID-controllers in the frequency domain.

The scope of our identification approach is in its current form more limited than most traditional identification methods used in control engineering today. At present, a whole range of identification methods exist, among which methods capable of handling models with 'difficult' characteristics like nonlinearity, time dependency and noisy input-output data. Our method can in its current form handle none of these irregularities. In terms of traditional identification methods our method is a batch system that provides single-input-single-output (SISO) identification of linear, state space models showing oscillatory behavior. Currently our approach is based on linear systems, but in principle (weakly) nonlinear systems could be handled by performing automatic linearization, taking the current model parameter values as the working point, prior to performing time scale reduction. Strongly nonlinear systems tend to show non oscillatory, chaotic behavior, and neither our method nor traditional identification methods are of much use. Also, our choice to modify only one selected parameter restricts the scope of results attainable. This scope could be enlarged by extending the number of unknown parameters to be adapted, and by using a multidimensional iterative method (e.g. the steepest gradient method) instead of the one-dimensional Newton method to (re)compute parametric values. Our approach can also be extended to multiple dimensions, i.e. using more than one performance specification for identification, for example by means of Pareto optimality.

A number of expert systems for identification have been constructed (Gentil *et al.*, 1990; Haest *et al.*, 1990). These systems automatically derive characteristic aspects of the model (like model order) and of the data (like noise factor). These characteristic aspects are used to direct the identification process, e.g. by suggesting the proper identification method to be used..

AI approaches to identification typically use qualitative (Say & Kuru, 1996) or fuzzy (Xu & Lu, 1987) test data to identify qualitative or fuzzy models. However, these models provide insufficient detail for conventional detailed design of (controlled) dynamic systems. On the other hand, we show that a mix of qualitative and knowledge-based AI techniques with traditional numerical techniques is capable of identifying numerical state-space models, suitable for conventional detailed design and control engineering, from an incomplete, abstract description of transient system response.

The approach discussed in this chapter can not only be used for parametric identification but also for detailed design of dynamic systems. Up to now, we have described the approach in terms of parametric identification of a system which already exists: the performance specifications used in identification are then obtained from experimental data on tests run with the real system. However, the performance specifications discussed can also be used to describe the *desired* transient response of a dynamic system. If a parametric model of the system is available, the technique discussed in this chapter can be used to estimate values for the systems parameters such that the response of the system complies with the required transient response described in terms of one or more performance specifications. In the context of detailed design, usually a combination of performance specifications is used to describe required system response, and the possible values of system parameters are usually constrained within a specific range. This means that in the context of detailed design it is essential to extend the algorithm to multi-dimensional, multi-variable optimization.

6.10 Conclusions

In this chapter we presented an approach to parametric identification which combines numerical and knowledge-based techniques to estimate parameter values in a partially-known model by using engineering abstractions of transient system response. These abstractions are well-known and used in many engineering domains to analyze and (re)design transient system response. Since the exact relation between model parameters and performance specifications defined on model output is in general unknown, approximate functions are used to link performance specifications to general model characteristics like natural frequency and damping ratio. In turn, these general model characteristics can be estimated by causal analysis and time-scale reduction in the physical (bond graph) model. To increase the accuracy of parameter estimation, a standard numerical iteration method (modified Newton iteration) is used to iteratively (re)compute parameter values.

The main contributions of this work are:

- Our model parameter identification approach combines techniques from engineering with more knowledge-based techniques to perform a complex engineering task: parametric system identification.
- Our approach performs parametric identification based on an abstract description of system response, instead of time series data. This allows us to use the approach both for parametric identification and for detailed design.
- Our approach focuses on identification of partially-known physical systems. In contrast to many other identification approaches, this allows us to integrate prior parameter information in the identification process, and to restrict identification to the estimation of unknown or insufficiently known parameters. The focus on physical systems also means that parametric adaptations are defined in terms of physically meaningful parameters. This is especially important if the techniques presented are to be used for *detailed design* instead of just for identification.
- Model parameter identification provides a good example of a parametric repair tactic in model revision; this approach shows that repair tactics are not restricted to (qualitative) structural adaptations on the physical model description, but can also be used to solve complex numerical engineering problems like matching model response to experimental data.

Chapter 7

Specification of modeling assumptions

A consistent library of model fragments complete with respect to the problems to be solved is essential in many automated modeling approaches. However, most of these approaches do not address the problem of constructing such a library in the first place. Realistic domain libraries will often be incomplete, and additional model fragments may have to be constructed during modeling and model revision.

Modeling assumptions support the modeling decisions made when constructing a specific model fragment; they form the design rationale *for the construction of a model fragment. Specifying and keeping track of these modeling assumptions during the modeling process is a complex and tedious task. In this chapter we show how specification of model fragments can be supported by organizing the modeling assumptions which underlying the construction of a new model fragment. Cliches, first developed in chapter 4 for supporting the specification of requirements, are shown to be helpful in supporting specification of model fragments and their underlying modeling assumptions.*

7.1 Introduction

Many automated modeling approaches reduce the model construction task to the problem of selecting the appropriate model fragments from a predefined library. An underlying assumption of these approaches is that a model fragment library is available in which all necessary fragments are incorporated. However, most of these approaches do not address the problem of constructing such a library in the first place. In realistic domains libraries

will often be incomplete, and additional model fragments may have to be constructed during modeling and model revision.

Earlier in this thesis (Chapter 5) we have shown that model revision can be performed by a 'constructional' approach, in which new models are automatically constructed from existing models by means of repair plans representing generic actions on models. However, this approach can not handle all problems liable to occur in modeling a physical system. For problems which can not be solved in this constructional manner, missing model fragments should be constructed by the human modeler himself to extend the library.

The idea of constructing a library of sharable and reusable model fragments is not new to automated modeling. Libraries of sharable and reusable models and model fragments have been constructed in different engineering domains. Experiences with this type of libraries have clearly pointed out the need for handles for knowledge management. This includes both database administration aspects for version management and accessibility restriction, as well as more knowledge-based background information on how and why a model fragment has been constructed. Modeling assumptions play an important role in extending the re-usability of model fragments in a library, since they can be used to assess the applicability of a model fragment in a new context. Furthermore, modeling assumptions support the modeling decisions made in the construction of a model fragment, and can be considered as its *design rationale*.

Making explicit the modeling assumptions made during the specification of a model(fragment) is a complex and tedious task. Specification of modeling assumptions by human modelers is often incomplete, imprecise and sometimes even inconsistent. Therefore, support in specifying and keeping track of modeling assumptions is important. In most engineering libraries, modeling assumptions are simply described in textual format, if at all. In order for a support system to be able to actually *reason* with and about modeling assumptions, it is essential that modeling assumptions should be structured and formalized.

Organization and structuring of information is essential in constructing and maintaining reusable and sharable libraries. In this chapter we discuss how techniques for specification support, first discussed in chapter 4 in the context of support for specification of modeling requirements, can also be used to support a human modeler in specifying modeling assumptions and associated model fragments. Specification of modeling assumptions extends the sharability and re usability of a library of model fragments. In section 7.2 we describe the library organization of a specific library, the OLMECO library (Breunese, 1996; Top *et al.*, 1995b). Models and model fragments in this library follow the differ-

entiation between ontological viewpoints discussed in chapter 3. Experiences with these and similar libraries "have clearly pointed out the need for handles for knowledge management information in sharing and reuse" (Borst *et al.*, 1997). Modeling assumptions form an important part of the model management information. Structuring and organization of the information available in modeling assumptions is essential in reasoning with and about modeling assumptions. Assumption-based reasoning is not only useful in automated modeling systems, but can also be used to support a human modeler in his specification of modeling assumptions and associated model fragments (section 7.3). We show that *cliches*, a technique first discussed in chapter 4 for the specification of modeling requirements, can also be used to organize and structure information for specification of modeling assumptions (section 7.4). Although the theories discussed in this chapter may be somewhat preliminary, they point to a direction in which the complexity and versatility of meta-model information can be made more manageable and better understood.

7.2 Library organization

The model representation described in chapter 3 provides the basis for structuring a library of models and model fragments. These ideas have been further developed in the OLMECO library for models of mechatronic components (Breunese, 1996; Top *et al.*, 1995b).

The basic structure of the OLMECO library is shown in Figure 7.1. The library follows the differentiation between ontological viewpoints, as discussed in chapter 3. The *refined-by* relations indicate three different points at which the user can make separate modeling choices: systems can be decomposed in different ways, functions of which device components are the carriers can be realized by different physical processes, and physical processes can be specified by different mathematical relations. These multiple links make it possible to provide a modeler with *reasonable alternatives* for model fragments in the model, which can be used for model revision. The *kind-of* relationship in Figure 7.1 provides the possibility for storing a component taxonomy in the library based on functional similarities between model fragments. This taxonomy can be used by the modeler to quickly access the functional components he wants to use.

Experiences of modeling experts in building the OLMECO library have clearly pointed to the need for handles for model management information. Figure 7.2 shows a schema for this kind of information in the OLMECO library. The management attributes version,

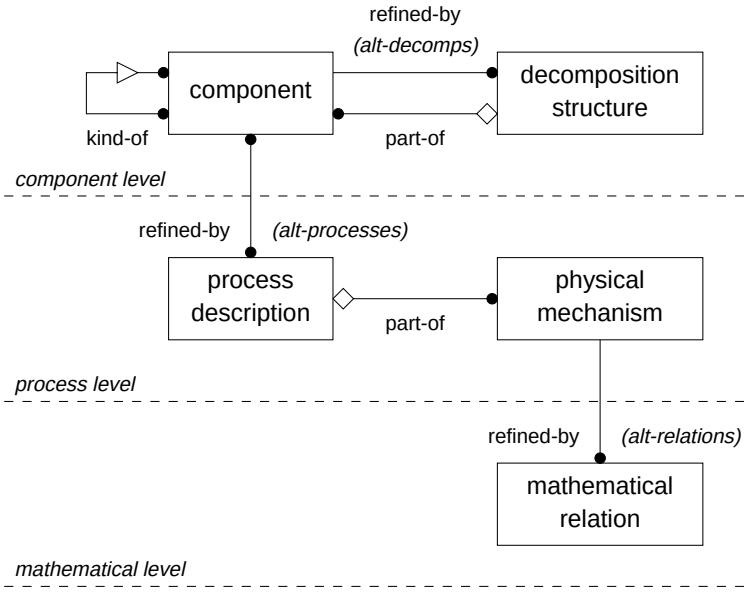


Figure 7.1: The general schema of the OLMECO library. Rectangular boxed denote entity types or classes. Solid line connections stand for relations. Open and solid balls indicate the cardinality of the relation to be zero or one (optional association) and zero or more (one-to-many association, respectively. The triangle symbolizes the is-a or kind-of relationship (generalization/specialization), while the small diamond is employed for the part-of relationship (aggregation) (figure copied from (Top et al, 1994)).

responsibility and publicness, represent database administration aspects. *User advice* represents miscellaneous comments for using the model (such as hints for simulation algorithms or step sizes in tricky cases). *Validation information* contains any information that explains how the model has been or can be validated: literature references, measurement data, etc. Currently, all model management information in the OLMECO library is simply given in free text format. In this chapter we show how modeling assumptions can be organized and partly formalized. Organization of modeling assumptions can be used to support the modeler in the task of specification, while formalization supports automated reasoning with and about modeling assumptions.

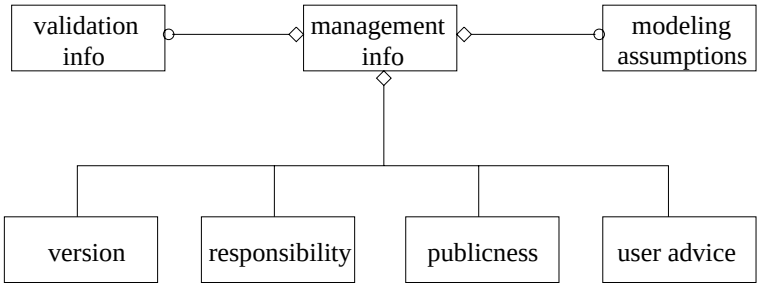


Figure 7.2: Model management information in the OLMECO library (figure copied from (Top et al, 1994)).

7.3 Structuring modeling assumptions

In the OLMECO library (Breunese, 1996; Top *et al.*, 1995b), there are three different points at which the user can make separate modeling choices: the choice between different component decompositions, the choice for different physical processes for each component and the choice for different mathematical equations for each physical process. These choices often depend on a number of different modeling assumptions. Together they provide the 'design rationale' for the choice between different alternative model fragments or for the specification of additional model fragments in case the alternatives in the library are insufficient for the task at hand. Modeling assumptions provide additional information on the *context* in which a model fragment is designed.

To provide support for specification of modeling assumptions and associated model fragments, both the model content and the model context should be structured. In the OLMECO model representation, model content has already been structured: Each model fragment is either a (composition of) functional components, a (partial) bond graph or a set of mathematical equations. For each of these types of model fragments, dedicated editors can be made available to support model entry (Top, 1993). In this section we describe how information on model *context* can be organized and structured to provide support for specification of modeling assumptions. We will also indicate how modeling decisions are linked to the actual content of model fragments. These links can be used to advice the user, and to help him in describing the content of a model fragment.

7.3.1 Model context

Modeling assumptions support the modeling decisions underlying the construction of a model fragment. They describe the context in which model fragments were constructed. Figure 7.3 shows the different elements of model context, and the relations between them.

As shown in Figure 7.3 model context embodies three elements:

- *Modeling decisions*, stating relevant approximations and abstractions used. Modeling decisions typically correspond to ignoring or including influences that are presumably (in)significant in the current model context (e.g. (in)compressible fluids). Modeling decisions should reflect only choices relevant in an application domain. For example, in the domain of fluid flow it is not necessary to state that quantum effects have not been included, since this is the case for each and every model in the

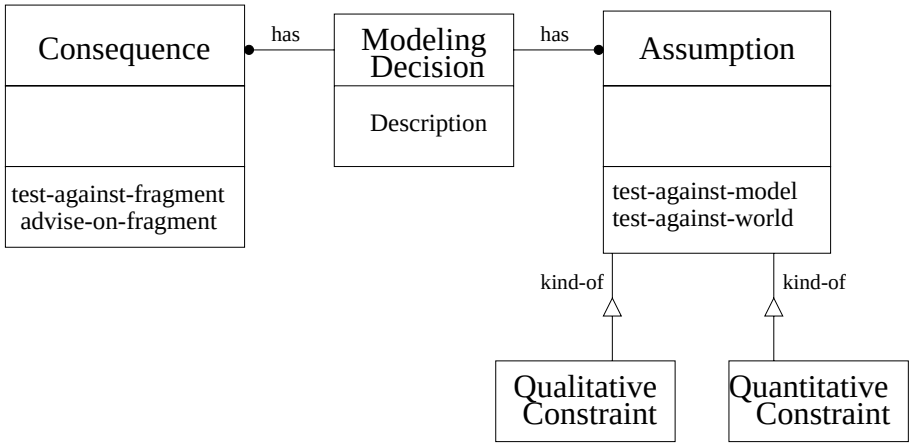


Figure 7.3: General schema of model context. Each model fragment can have zero or more modeling decisions associated with it.

fluid flow domain, and thus does not reflect a proper choice in the application domain.

- *Modeling assumptions*, expressing the conditions under which a modeling decision is valid. Modeling assumptions can be specified in terms of both qualitative and quantitative (in)equalities between quantities and constants, e.g. Reynolds number $Re < 2300$, or in terms of order of magnitude constraints, e.g. $X \gg Y$ or 'deviation small enough'. Modeling assumptions can be used in model fragment selection to decide whether a modeling assumption is valid in the current situation, and therefore whether the corresponding model fragment can be used in the current model. They can also be used dynamically at run-time to check whether the validity conditions are still valid within the current model, and thus whether the model remains valid during simulation.

- *Consequences* for model content. Modeling decisions not only define model context, but also influence model content. In general, it is not possible to generate model content automatically from the modeling decisions underlying model construction. However, it is possible to give suggestions about the content and representation of a model fragment based on the modeling assumptions on which it is based. These suggestions will in our approach be stated in terms of the model level affected by the modeling decisions (corresponding to the component, process and mathematical model fragments in the OLMECO representation), and in terms of types of model elements or interactions that should be explicitly included in the model fragment (e.g., the modeling decision 'viscous flow' implies that an element for hydraulic friction ($R_{hydraulic}$) should be included in each model fragment for which this assumption is

specified). These suggestions can be used to advice the modeler on how to construct the model fragment, or they can be checked automatically against the constructed model fragment to ensure that the content of the model fragment constructed does not violate the modeling decisions on which it is based.

Figure 7.4 shows a specific modeling decision (inviscid flow), the assumptions underlying this decision (inviscid flow is valid if fluid velocity is low ($M \ll 1$), or if the length along which convection takes place is very small) and some hints on the consequences of this modeling decision on model structure (the neglection or inclusion of viscosity takes its effect on the physical level, and indicates that hydraulic resistance $R_{hydraulic}$ should be excluded from each model fragment for which inviscid flow is assumed). Order of magnitude assumptions like $M \ll 1$ can only be tested automatically if the relation $\ll$ is formally specified. However, the threshold for this definition often depends on the desired accuracy of the modeling results, and is therefore difficult to specify up-front.

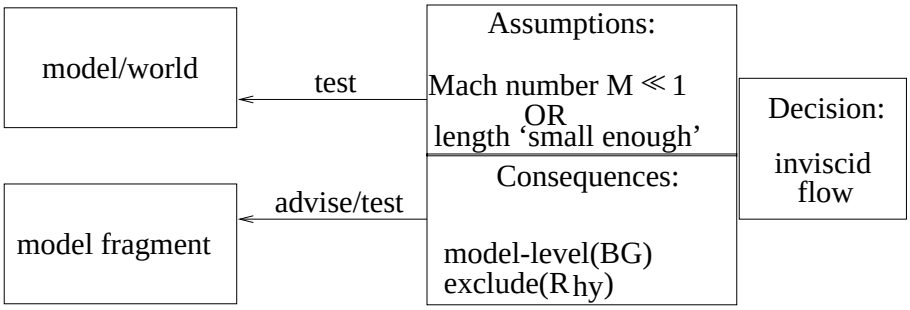


Figure 7.4: Example of a modeling decision (inviscid flow), its underlying assumptions and its consequences on model structure.

7.3.2 Structure over individual modeling decisions

In the previous section we have discussed which information lies at the basis of a single modeling decision. However, in general the construction of model fragments requires a large number of modeling decisions, some of which may be strongly interdependent. Therefore, it is important to distinguish some additional structure on top of single modeling decisions. In this chapter we will discuss this additional structure and illustrate it by looking at the thermodynamic part of the OLMECO library. This part of the library focuses on "heat generation, exchange and transport through thermal fluids (liquid or gas), where both convection and conduction of thermal energy can play a role" (Top *et al.*, 1995a).

Figure 7.5: General schema of a decision structure. Structure is provided by distinguishing relevant processes in a domain, characteristic properties for each of these processes, and relevant modeling choices for each property.

The following forms of additional structure among modeling decisions can be distinguished (also depicted in Figure 7.5):

- Modeling decisions can be split up into different relevant processes in the application domain in question. These processes need not necessarily be independent. Discovering these relevant processes is often as simple as looking at the index of a standard domain textbook. Fluid flow convection, conduction and radiation are relevant processes in the thermodynamic fluid flow domain.

- Each process can be characterized by a number of often-used properties. For example, fluid flow convection can be characterized by a *flow*, a *geometry* in or around which this flow flows, and a *surface* along which convection takes place.

- Each property has a number of *relevant modeling choices*. These modeling choices capture *dimensions* along which a modeling decision can be made, and are also called assumption classes (Falkenhainer & Forbus, 1991). The choices within a modeling choice are mutually exclusive. For example, the modeling choice 'viscosity' allows the flow to be modeled either as viscous or as inviscid, but never as both. Each decision in a modeling choice can be associated with underlying assumptions and consequences on model content, as discussed in the previous section.

  Figure 7.6 shows that modeling choices either consist of a single modeling dimension, or of a *package* of interdependent modeling options. *Default* assumptions can be identified for a class of problems. Finally, constraints define relations among (packages of) assumptions.

  – 'Packages' of modeling options. Some modeling options are often used together, and therefore can be presented as a 'package'. An example is the expression 'Bernoulli flow'. This refers to a steady, incompressible, viscous flow with non-uniform velocity. Identifying packages of modeling assumptions allows the human modeler to specify his decisions in a short, abstract manner, and leave the 'bookkeeping' to the support system.

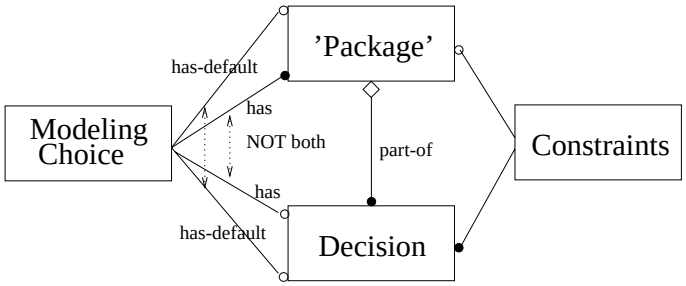


Figure 7.6: General schema of a modeling choice. Modeling choices can be either packages or single decisions. Constraints define relations among (packages of) assumptions.

– Default decisions. Many default decisions are used in standard situations. To prevent the modeler from having to specify each and every modeling assumption, it is essential to distinguish as many default assumptions as possible. The modeler then can be presented with the default decisions for a class of problems, and only has to specify non-standard modeling decisions explicitly.

– Constraints and relations among different modeling choices. Some modeling choices may conflict, and can therefore never be included together for the same model fragment. An example of such a conflict is the combination of turbulent inviscid flow: this combination is physically impossible. On the other hand, some modeling choices only become relevant when other choices have already been made. For example, it is not useful to decide on whether viscosity should be Newtonian or non-Newtonian before having decided to model flow as viscous instead of inviscid. Support for specification of model context can take the form of *detecting* conflicts between specified assumptions, or of suggesting additional modeling choices only when they become relevant. Automatically detecting conflicts helps the human modeler to keep the set of modeling assumptions consistent, while suggesting additional modeling choices only when they become relevant helps the modeler to focus only on the assumptions relevant in the current model context.

The next section discusses the operationalization of the concepts described above in the form of *process cliches*.

7.4 Cliches: Support for specification

Organization of modeling decisions in a domain can be presented in the form of *process cliches*, similar to the goal cliches described in chapter 4. These cliches are used to structure and support specification of modeling decisions. Just as in chapter 4, cliches contain relevant modeling choices, as well as predefined *constraints*. In this section we will discuss how the different concepts in assumption organization, discussed in previous sections, can be fitted into the framework of process cliches. We also show how these cliches are used to support the specification of modeling decisions and associated model fragments.

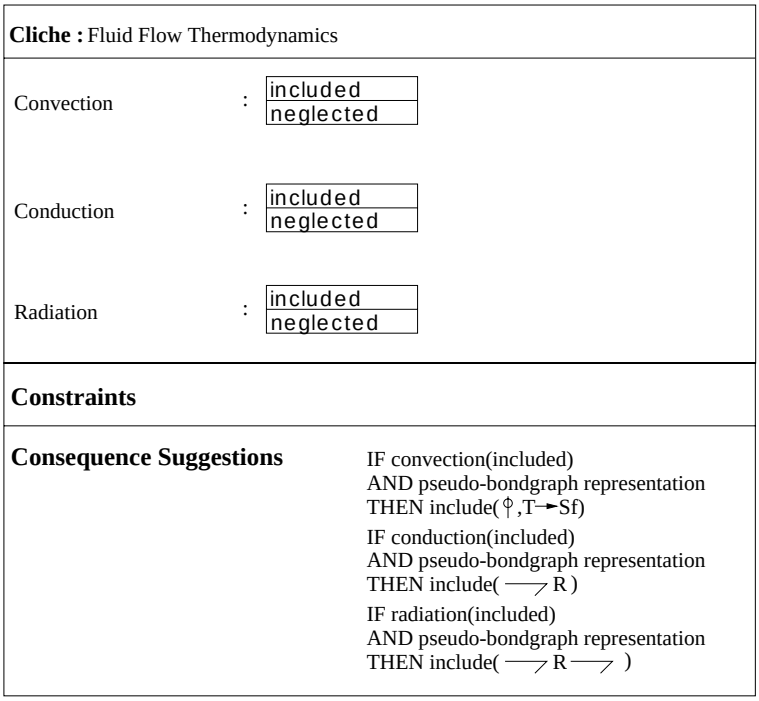


Figure 7.7: Cliche for fluid flow thermodynamics. Only some of the constraints are presented in the cliche.

First, relevant processes should be distinguished in the domain of interest. In the thermodynamic fluid flow domain these relevant processes are conduction, convection and radiation. Figure 7.7 presents the matching cliche for fluid flow convection. Each of the three relevant processes needs to be explicitly included or excluded for each model fragment. The lower part of Figure 7.7 presents some of the constraints associated with this cliche. All of these cliches describe the consequences of a modeling decision (i.e., include or exclude a basic process) in terms of model elements to be present in the associated model fragment. For example, the first consequence constraint in Figure 7.7 states that if the process of convection is explicitly included, and the model representation is in the form of a

pseudo-bond graph then at least a flow source (Sf), controlled by temperature difference T and fluid velocity ϕ should be present in the pseudo-bond graph (Top *et al.*, 1995a). The other consequence constraints in Figure 7.7 express similar consequences for conduction and radiation.

Now, a process cliche can be constructed for each relevant process. Figure 7.8 shows the process cliche for the process of fluid flow convection. Essential properties of fluid flow convection are a *flow*, a *geometry* in or around which this flow moves and a *surface* along which convection takes place.

Each property has a number of important modeling choices associated with it. Modeling choices can either be represented as *single dimensions* along which a choice must be made, or as *packages of modeling options*. Packages represent combinations of modeling choices which are often used together. The concept of Bernoulli flow is an example of such a package; it consists of four different modeling decisions: viscosity(viscous), compressibility(incompressible), uniformity(Non-uniform velocity) and steadiness of the flow(Steady). This decomposition of packages into basic assumptions is specified in the form of decomposition constraints. Figure 7.8 shows a similar *decomposition constraint* associated with the decomposition of the package 'basic fluid flow'. This decomposition of packages allows the human modeler to specify his assumptions in an abstract, familiar description, while the support system can automatically deduce the full set of basic modeling decisions involved.

Specification of modeling assumptions is an extensive and tedious task. However, in many domains *default* decisions can be recognized. To relieve the modeler of explicit specification of all modeling decisions, default decisions, either in the form of single modeling decisions or in the form of packages can be used. Default decisions can be automatically presented by the support system to the human modeler, allowing the modeler to specify only non-standard decisions explicitly.

Each package or single modeling decision can have one or more *underlying assumptions* associated with it (see also Figure 7.3). These underlying assumptions describe the conditions under which the modeling decision is valid. They can be used during specification to examine whether a particular choice fits the current situation. If assumptions are specified as proper mathematical inequalities in terms of variables from the model fragment to which they belong, they can be automatically validated by the support system. If assumptions are specified in terms of order of magnitude inequalities, automated validation depends on defining a proper threshold for order of magnitude reasoning based on the re-

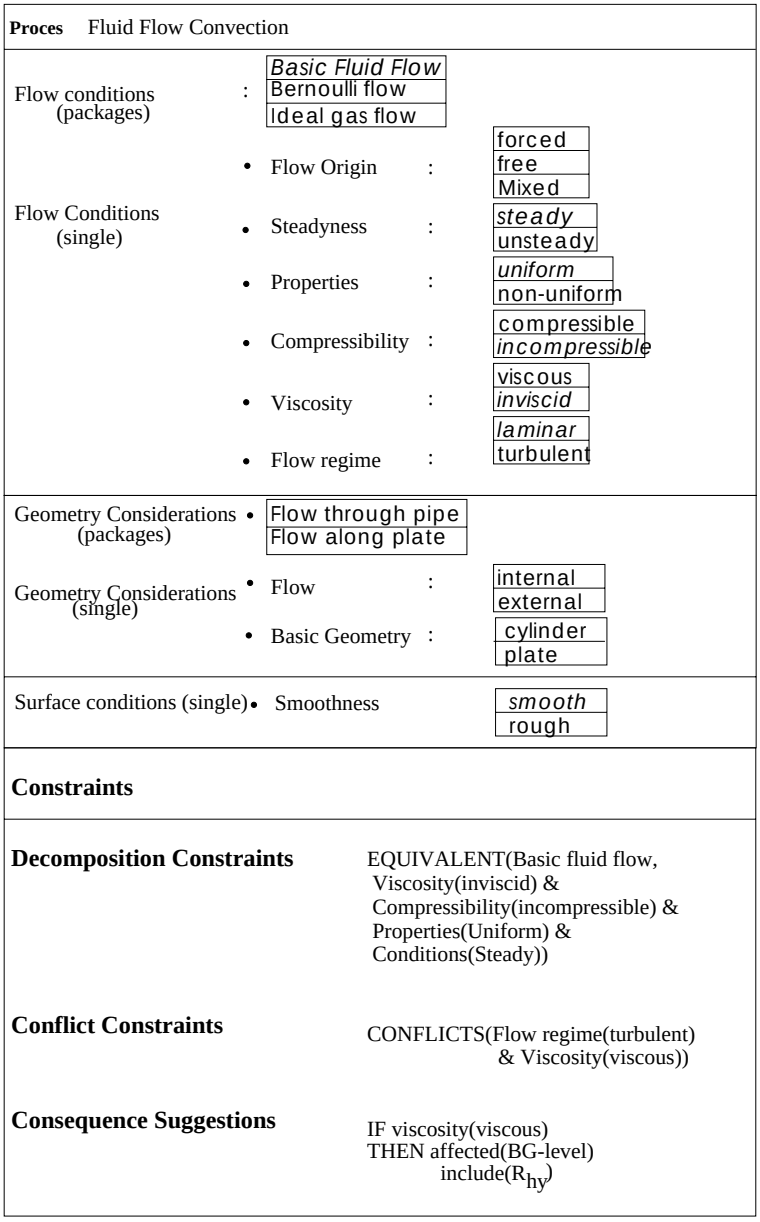


Figure 7.8: Cliche for fluid flow convection process. Default decisions for a modeling choice are presented in italics. Only some of the constraints are presented in this figure.

quired accuracy of analysis results.

Some combinations of modeling decisions are physically impossible, which means that these modeling decisions can never be included together for the same model fragment. In process cliches this information can be presented in the form of *conflict constraints*. Figure 7.8 presents an example of such a conflict constraints, indicating that an inviscid, turbulent flow is physically impossible. Conflict constraints can be used by the support system to automatically detect inconsistencies within the current set of modeling assumptions, and warn the human modeler of them. Both inconsistencies within a single process cliche as well as conflicts between modeling decisions in different process cliches can be detected.

Each modeling decision influences the structure and content of the model fragment(s) associated with it. In a process cliche, these *consequences* can be described in terms of *consequence suggestions*. Consequence suggestions are expressed in terms of the model-level(s) affected by the assumption, and in terms of specific model-elements or interactions which should be included in or excluded from the model fragment in question. Figure 7.8 presents an example of such a consequence suggestion, stating that modeling a flow as viscous influences the physical process level (described in terms of a bond graph) and requires that a hydraulic resistance process R_{hy} is included in each model fragment for which this assumption holds.

Consequence suggestions can be used in different ways to support the specification of modeling decisions and associated model fragments. First, the fact that modeling decisions influence a limited number of model levels can be used to present only the assumptions relevant to the level on which the modeler is currently acting. This supports a structured approach to model context specification, since it focuses the attention of the modeler only on the modeling decisions directly relevant to the current level of specification. Secondly, the consequences of a modeling decision can be expressed in the form of model elements to be explicitly included or excluded from the model. Consequence suggestions can be presented simply as advice to the user, to help the user in constructing a model fragment. In cases where a model fragment has already been constructed, this information can also be used to automatically test whether a model fragment violates its modeling decisions. A strongly typed and structured representation like the bond graph language makes this approach possible.

7.5 Discussion

The theories described in this chapter are still somewhat preliminary. Consequently there are still a number of open questions. The approach is clearly very knowledge-intensive: even our preliminary investigation in the domain of fluid flow heat transfer took a lot of time, and yet it is far from complete. It is also not yet clear how the theories described in this chapter will scale up to larger domains. Due to the intricate dependencies among different modeling decisions, inclusion of additional modeling decisions may introduce many additional constraints.

The domain of heat transfer has a 'strong theory', which also means that modeling decisions in this domain are relatively well formalized and quantified compared to modeling decisions in many other engineering domains. The techniques described in this chapter are most suitable for domains with strong domain theories, since they are based on existing domain theories.

A structured, generic and strongly typed representation like the bond graph representation is clearly an asset in representing consequences of modeling assumptions on the structure and content of model fragments. It allows us to represent consequences in such a format that they can be automatically checked. (Finn, 1993) uses a similar approach to specifying the consequences of modeling decisions on model content, but focuses on providing advice to human modelers rather than on automated support. Although the theories presented in this chapter have not yet been implemented, they are geared towards automated support for the assumption specification process.

A number of the structuring and representation techniques discussed in this chapter, like the distinctions between model content and modeling decisions and the concept of assumption classes, have been previously discussed in the context of automated modeling. Most automated modeling approaches in AI (Addanki *et al.*, 1991; Falkenhainer & Forbus, 1991; Biswas *et al.*, 1993) have focused on automated reasoning with and about modeling assumptions. They pay little or no attention to the specification of model fragments and associated modeling assumptions. An exception to this rule is the approach described in (Schut & Bredeweg, 1994; Schut & Bredeweg, 1996). This approach recognizes the important role of specification in model revision, but focuses on supporting specification of the content of (qualitative) model fragments rather than on specification of their underlying modeling decisions.

In our opinion support for specification of modeling decisions is essential in supporting

the construction and maintenance of large libraries in complex application domains. The theories presented in this chapter are a first step in the direction of useful semi-automated support for the specification of modeling decisions.

7.6 Conclusions

In this chapter we have shown how techniques developed for the specification of requirements (chapter 4), in the form of specification *cliches*, can also be used to support specification of modeling assumptions and associated model fragments. Thereby we hope to relieve the human modeler of some of the burden involved in specifying and keeping track of modeling assumptions.

We have shown which forms of additional structure can exist among modeling decisions in a domain, and how this structure can be represented in the form of process *cliches*. We have also shown how these cliches can be used to assist human modelers in the tedious and difficult task of specifying assumptions and associated model fragments. The techniques provided in this chapter are a first step in the direction of semi-automated support for the specification of model fragments and their underlying modeling decisions.

Chapter 8

Redesign problem solving

In the preceding chapters of this thesis we focused on model revision. In this chapter we will broaden our view, and consider the task of redesign *and its methods. We present a knowledge-level analysis of redesign to provide a better understanding of the task of redesign, the goals it aims to achieve and the different ways to achieve these goals. We will show that our approach to model revision, as discussed in the remainder of this thesis, is a special form of redesign in which simulation models are the subject of redesign. Redesign is viewed as a family of methods based on some common principles. We also distinguish a number of dimensions long which different problem-solving methods for redesign can differ. These dimensions point to relevant design decisions in constructing the problem-solving behavior of alternative methods for redesign. By examining the problem-solving behavior of a number of existing redesign systems and approaches, we came up with a collection of problem-solving methods for redesign and developed a task-method structure for redesign.*

In constructing a system for redesign a large number of knowledge-related choices and decisions are made. In order to describe all relevant choices in redesign problem solving, we have to extend the current notion of possible relations between tasks and methods in a PSM architecture. The realization of a task by a PSM, and the decomposition of a PSM into subtasks are the most common relations in a PSM architecture. However, we suggest to extend these relations with the notions of task refinement and method refinement. These notions represent intermediate decisions in a task-method structure, in which the competence of a task or method is refined without immediately paying attention to its operationalization in terms of subtasks. Explicit representation of this kind of inter-

mediate decisions helps to make and represent decisions in a more piecemeal fashion.

8.1 Introduction

The concept of reusable *problem-solving methods (PSMs)* is present in many current knowledge engineering frameworks, e.g. Generic Tasks (Chandrasekaran, 1988), Components of Expertise (Steels, 1990), Method-to-Task (Klinker *et al.*, 1991; Musen, 1989; Gennari *et al.*, 1994), role-limiting methods (McDermott, 1988), GTMD (O'Hara & Shadbolt, 1993), COMMONKADS (Wielinga *et al.*, 1993) and DESIRE (Kowalczyk & Treur, 1990; Brazier *et al.*, 1995). The interest in PSMs originates from the need to describe and explicate generic aspects of the problem-solving behavior of knowledge-based systems. Problem-solving methods are used in a number of ways in knowledge engineering (Fensel & Benjamins, 1996): as a guideline to acquire problem-solving knowledge from an expert, as a description of the reasoning process of the expert and the knowledge-based system, as a skeletal description of the design model of the knowledge-based system, and to enable flexible reasoning by automatically selecting methods during problem solving.

In this chapter we focus on *comparing* problem-solving methods for *redesign*, and on identifying and representing relevant choices in constructing and selecting problem-solving methods for redesign tasks. As the notion of redesign incorporates many different (sub)-methods it is best characterized as a family of problem-solving methods. We have made a knowledge-level analysis of redesign, and came up with a collection of problem-solving methods for this task. This collection was obtained in a bottom-up manner by examining the problem-solving behavior of existing redesign systems and approaches, most notably those developed in the REVISE[1] project. Within this project the redesign of technical systems (Eldonk *et al.*, 1996), simulation models (Pos *et al.*, 1997b), compositional architectures (Brazier *et al.*, 1996c) and control knowledge in knowledge-based systems (Straatman, 1995) is studied.

When developing any knowledge-based system for redesign for a specific application domain a large number of choices will have to be made. In order to structure this development process, these choices should be made explicit (Akkermans *et al.*, 1994). In order to describe all relevant choices and decisions, we need to extend the current notion of possi-

[1]The REVISE project has been funded by NWO/SION within project 612-322-316, "Evolutionary design in knowledge-based systems". Participants in the REVISE project are: the TWIST group at the University of Twente, the SWI department of the University of Amsterdam, the AI department of the Vrije Universiteit van Amsterdam and the STEVIN group at the University of Twente.

ble relations between tasks and methods. The general notion of a problem-solving method as a direct link between a task goal and the decomposition of this task into subtasks, is not in itself sufficient to describe all the relevant choices and decisions. The notions of task refinement and method refinement are introduced to represent intermediate decisions in constructing and selecting problem-solving methods. In these intermediate decisions, the competence of a task or method is refined without immediate attention being paid to the further operationalization of this task or method. This corresponds to changing the exact nature of the problem with the aim of making it easier to solve. Explicit representation of these types of intermediate decisions helps to represent decisions in a more structured fashion.

The structure of the chapter is as follows: In section 8.2, we present an extended architecture of a PSM, based on the additional notions of task refinement and method refinement. In section 8.3 we present our view on redesign, and distinguish a number of dimensions along which redesign approaches can differ. In section 8.4 we present a top-level problem-solving method for the task of redesign, developed in the REVISE project. In sections 8.5 and 8.6 we present two subtasks of redesign in more detail: requirement management and design modification. We will show which role the dimensions of redesign play in comparing alternative PSMs for these subtasks. We will also use these subtasks as an illustration of the notions of task refinement and method refinement, and their role in comparing tasks and PSMs in a task-method structure. Section 8.7 concludes the chapter, and points out some implications for knowledge engineering.

8.2 Modeling framework

A general view on tasks, methods and their mutual relations is the following: A *task* is characterized by a goal it can achieve. A task can potentially be realized by a number of problem-solving methods (PSMs). A *PSM* describes a way to solve a task: it decomposes a task into subtasks, each associated with a subgoal, and/or into primitive inferences, that directly achieve goals. Recently, focus has shifted from the description of reasoning strategies per se to the description of assumptions underlying these reasoning strategies (Akkermans *et al.*, 1994; Wielinga *et al.*, 1995; Fensel & Straatman, 1996). The idea is that PSMs provide solutions to tasks by making *assumptions* about the precise definition of their functionality, and about the available domain knowledge.

With this idea in mind, (Benjamins *et al.*, 1996) states that a problem-solving method con-

sists of three subparts (presented in Figure 8.1):

- its *functional specification*. This is a declarative description of the input/output behavior of the PSM. It describes what can be achieved by the PSM.
- its *operational specification*. This is an account of how to realize that behavior. The operational specification of a PSM decomposes a task into subtasks and/or primitive inferences, and defines an ordering over these operators.
- its *assumptions*. Problem-solving methods make assumptions on the precise definition of their functionality (*teleological assumptions*) and on the availability and properties of domain knowledge (*ontological assumptions*). *Teleological assumptions* are introduced in matching a task goal to the functional specification of a PSM if the functional specification of a PSM is more restricted than the task goal it is meant to achieve. In other words, they describe the requirements the PSM can meet. For example, in diagnosis a particular PSM might only be able to find single faults. *Ontological assumptions* are introduced when realizing the functional specification of a PSM by its operational specification, since operationalization often depends on the availability and properties of domain knowledge. Ontological assumptions describe what a PSM expects in return for the functionality it provides. An example of such a requirement is the availability of heuristics that link violated constraints to possible repair actions (repair plans).

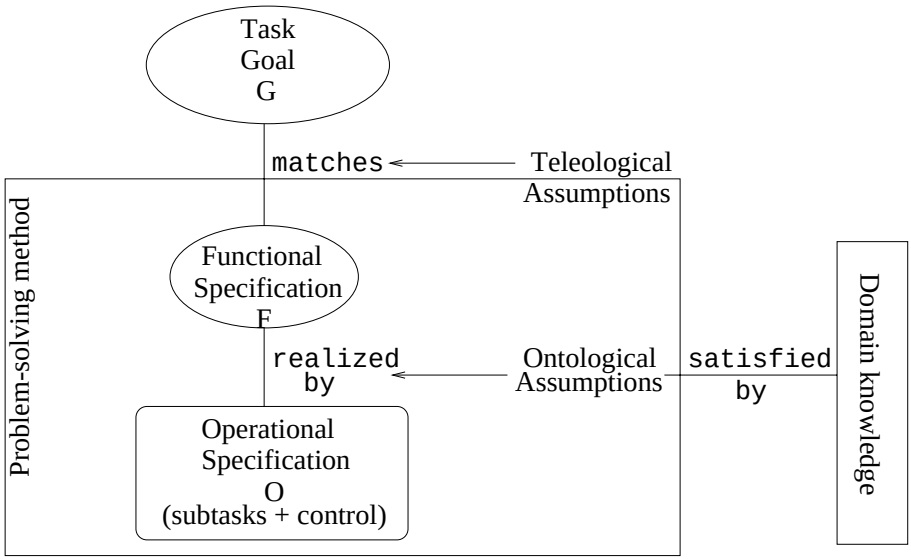


Figure 8.1: The architecture of a PSM.

We tried to use this framework to establish a task-method structure for problem-solving

behavior of different redesign systems and approaches, similar to the task-method structure for diagnosis presented in (Benjamins, 1993). However, although the PSM architecture in (Fensel & Straatman, 1996) is rich, it does not adequately cover all types of design decisions we encountered. The current framework provides a limited number of relations between tasks and methods: a task goal can be *matched* to the functional specification of a problem-solving method, and a problem-solving method can be *realized* by an operational specification which decomposes the functional specification in a number of subtasks. While trying to construct a task-method structure for redesign, we encountered two types of *intermediate* decisions that can not easily be represented in this framework; we call them *task refinement* and *method refinement*, respectively.

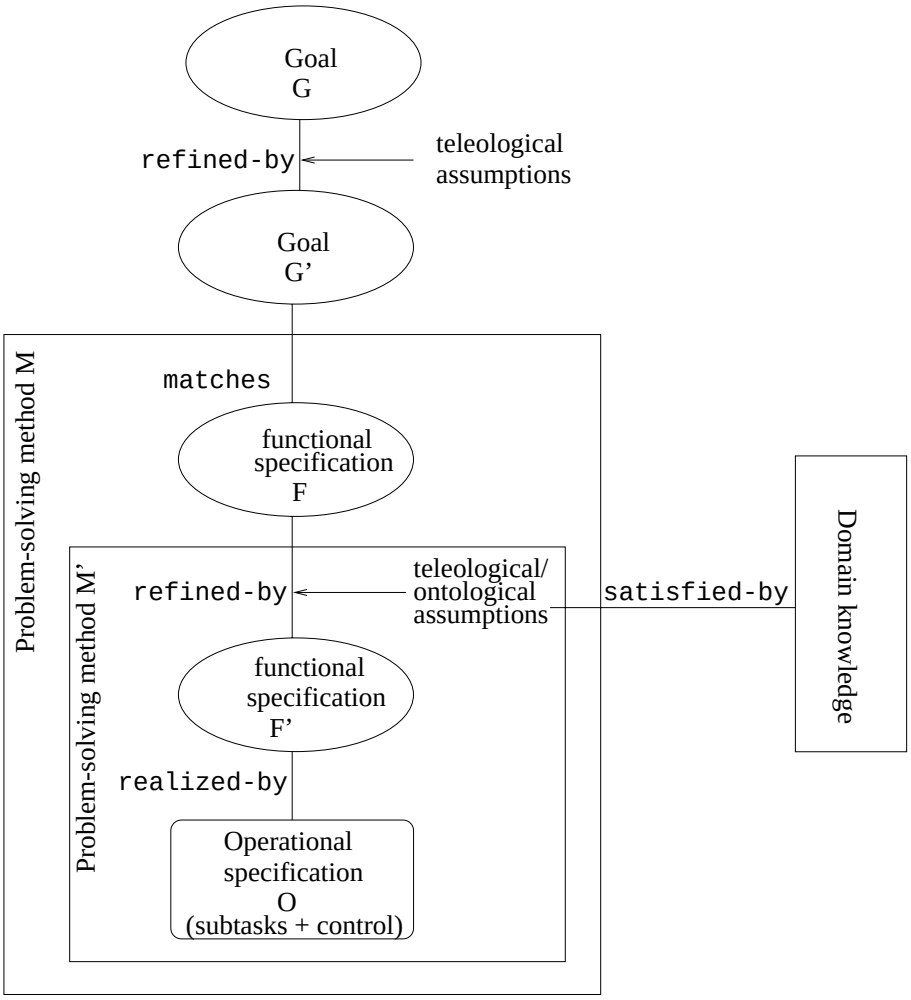


Figure 8.2: Our extended architecture for PSMs.

- *Task refinement* refines a task goal G into a more specific task goal G' but without directly making explicit *how* the task should be solved. Refinement of a task translates

a task goal into a weaker task goal by making additional teleological assumptions about the precise definition of the tasks functionality.

- *Method refinement* refines the functional specification F of a problem-solving method into a more refined functional specification F' without making explicit how exactly this functional representation should be operationalized in terms of (control over) subtasks and/or primitive inferences. Competence refinement of a problem-solving method may introduce both additional teleological assumptions on the precise definition of the functional specification of a PSM as well as additional ontological assumptions on knowledge structures that should be available in the domain knowledge. In the first case, the goal of the refined functional specification F' may be more restricted than the goal of the original specification F. In this case, teleological assumptions describe the additional restrictions on the goal to be achieved. In the second case, the more refined functional specification F' may require additional (properties of) domain knowledge not required by the original specification F. Ontological assumptions in method refinement describe which additional domain knowledge is introduced in refinement of the functional specification of a PSM.

The *competence description* (Akkermans *et al.,* 1994) of a method or goal describes its problem space and requirements on the solution it produces. Under this definition, both decisions described above can be viewed as *competence refinements*, since they refine the functional specification of a task or method without directly making explicit how this competence could be achieved. This corresponds to changing the exact nature of the problem with the aim of making it easier to solve. Both forms of refinement are only to be used as *intermediate* steps in a task-method structure: the ultimate goal of constructing a task-method structure is still to relate a task to be solved to an operational description of how this task can be solved. However, the recognition of these two *intermediate* relations allow to better distinguish and separate different reasoning steps in the construction of a task-method structure. Figure 8.2 shows the additional refinement relations, and their place in the PSM architecture. Of course, the process is recursive, i.e. goals of subtasks of a problem solving method can again either be refined to weaker goals or directly operationalized by an operational specification etc.

8.3 Redesign

Redesign is an inherent part of most design processes, but can also be seen as a family of design methods in itself. In contrast to design-from-scratch, redesign starts out with an existing design description and modifies this until it fits the current needs as good as possible. In order to perform redesign it is essential that some form of knowledge is available that allows the adaptation of existing designs. This knowledge is based on the following two principles: 1) minimally change the design, and 2) maximally exploit existing properties of the domain. An underlying assumption of the task of redesign is that the existing design description is "close enough" to fulfill the needs by only limited adaptations. However, what is considered close enough in a specific case depends on the nature of the adaptation knowledge, and on the way different requirements interact.

Redesign can play two different roles in the complete design process: First, redesign can be seen as a sub-phase of the design process. Here, design is viewed as an iterative process that uses intermediate results as a means of getting a final design description which fulfills the requirements. The task of redesign on the basis of a design created earlier produces a new temporary design description which is (hopefully) closer to the specification than the former design description. This view is the basis for the Propose-Critique-Modify family of design methods discussed in (Chandrasekaran, 1990). Secondly, redesign can be considered in the context of reuse. Here, redesign starts with a previously constructed design description, and a new set of requirements. The previously constructed design description must now be modified to fulfill the new set of requirements. This view is often taken in approaches such as case-based design (see e.g. (Kolodner, 1993; Maher *et al.*, 1995)), when the already retrieved case is adapted to suit the new requirements. Although there are very subtle differences between these two views, in both cases the important issue is to bridge the gap between a set of requirements and an existing design description. Therefore, both Iterative Redesign and Redesign for Reuse can be captured by a single spectrum of problem-solving methods for redesign.

Many systems that solve redesign problems have been described in literature (Mitchell *et al.*, 1983; Howe *et al.*, 1986; Fischer *et al.*, 1987; Marcus *et al.*, 1987; Daube & Hayes-Roth, 1989; Goel, 1991; Smyth & Keane, 1996; Eldonk *et al.*, 1996; Brazier *et al.*, 1996a), but when one takes a closer look at the different variants of the redesign task, subtle differences exist that have an impact on how the task can be performed and what kinds of knowledge are involved.

A first source of variation in redesign is the design description. There are several aspects of the design description which are important in the context of redesign. The first of these is the *fixedness* of the structure of the design description; at one end of the spectrum, the structure of the design description can be completely fixed during redesign, and only the values assigned to parameters can be altered. This leads to parametric redesign. On the other end of the spectrum we have situations where changes to the structure of the design description are not limited in any way. In between these extremes, there are cases where a skeleton structure is considered to be fixed but where the specific structure still can be filled in. Another dimension concerning the design description is the *nature of the information* presented in the design description. At one end of this spectrum the design description can purely describe the current status of the design, while on the other end the design description includes a complete plan of design steps resulting in the current design. The latter results in a form of redesign called derivational analogy (Mostow, 1989; Carbonell, 1983), while the former is the subject of redesign approaches which directly modify the current design description (e.g. KRITIK (Goel, 1991) and our own model revision system 007 (Pos *et al.*, 1997b)).

The requirements put on the design description provide a second source of variation in redesign. Again, there are several dimensions along which the requirements can be classified. The first of these is the *operationality* of requirements. Requirements are operational if their truth can be automatically derived from the design description by some inference method. The question to be considered is whether it is sufficient in an application domain to express needs and desires with operational requirements only, or whether there is a need to express non-operational requirements as well? The latter situation requires more extensive support for requirement management. Software design is a typical example in which the ability to express non-operational requirements is important in supporting the user in requirement specification. Another dimension with respect to the requirements posed on a design description is their *(local or global) nature*. Local requirements are applicable to a single component or parameter, while global requirements specify properties of the complete design. An example of a global requirement is the maximum weight of a device; this weight can not be attributed to a single component but is a function of the combined properties of all the components in the device. Requirements in our own model revision system 007 are mostly global constraints on model behavior or model structure.

Each redesign process requires some form of knowledge on which adaptations are possible/suitable/useful etc. The nature of the adaptation knowledge is the third source of variation in redesign. Again, there are several dimensions along which this adaptation

requirements	design description	adaptation knowledge
operational/ non-operational	structure fixed/free	search-based/ plan-based
local/ global	derivation/ design	specific/ generic

Table 8.1: Dimensions of redesign problems.

knowledge can be characterized. The first of these is the *knowledge intensity* of the adaptation knowledge. At one end of this spectrum are purely search-based approaches, like constraint satisfaction, while purely knowledge based approaches form the other end of the spectrum. A second dimension is the *generality* of the adaptation knowledge: how widely applicable is the adaptation knowledge. Application-specific fixes are at one end of this spectrum, while very general strategies are located on the opposite end. Our own model revision system 007 uses the notion of *repair plans* to suggest design adaptations. The knowledge captured in these repair plans is generic enough to be applied to many different physical systems, but not as generic as general strategies like 'divide-and-conquer'.

Table 8.1 summarizes the dimensions along which redesign problems can differ. A space of redesign problems can be constructed by taking the Cartesian product of the values on each of these dimensions to form a multi-dimensional problem space for redesign problems. Most of the dimensions mentioned here have been described in the context of design problems other than redesign (Wielinga & Schreiber, 1997; Bernaras, 1994). This is a result of the earlier mentioned position of redesign in the spectrum of methods for design: redesign is both a part of many other design methods, like case-based design, as well as an umbrella for many different design techniques, like parametric (re)design and configuration (re)design.

8.4 A problem-solving method for redesign

The task of redesign takes as input a set of requirements and a design description, and produces as output another design description and the current requirements. Redesign can be considered to consist of two subtasks: requirement management and artifact management. The latter is further decomposed into assessment and repair. This decomposition of the redesign task (which we will in the remainder of this chapter refer to as the "REVISE method") is motivated by work in the REVISE project (Pos *et al.*, 1996b; Brazier *et al.*, 1996c). Figure 8.3 presents the data flow for the REVISE method.

- *Requirement management* This subtask is responsible for specification, management, refinement and adaptation of requirements. Some examples of problem-solving methods for this task are discussed in section 8.5.

- *Assessment* This subtask is responsible for determining the differences between the (properties of the) current design description and the requirements. These differences drive the repair subtask.

- *Repair* Within this subtask, the design description is adapted such that it will better fit the requirements. Determining which part of the design description will be adapted (critique), and how it will be adapted (modify) are often tightly coupled subtasks in this task. Section 8.6 discusses some examples of problem-solving methods for the subtask of design modification.

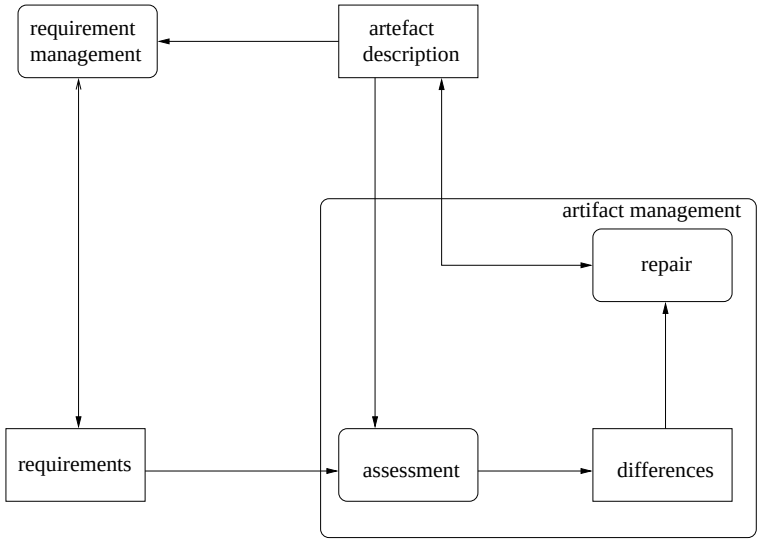


Figure 8.3: Data flow diagram of the REVISE method. This method provides an abstract, top-level decomposition of the redesign task.

(Brazier *et al.*, 1996c) presents a generic task model for redesign, which incorporates the manipulation of requirements (and their qualifications stating how 'necessarily' they have to be fulfilled), the manipulation of the design object description and the manipulation of redesign strategies. This generic model was one of the inputs for the decomposition of redesign developed in the REVISE project. The decomposition of redesign presented in the REVISE method is a subset of this generic task model. The main difference between the generic task model for redesign and the REVISE method lies in the manipulation of redesign strategies: In principle, in a redesign system different strategies can be used to organize and combine the different redesign subtasks, like requirement management and

artifact management (Brazier *et al.*, 1994; Brazier *et al.*, 1996c). However, most redesign systems do not reason *explicitly* about internal control, but assume that the order of the different redesign subtasks is fixed beforehand. Therefore, in the REVISE method for redesign no explicit reasoning is included about strategies and control over the different subtasks.

8.4.1 Other models for (re)design

If the model described above is indeed general, then it should incorporate the same elements as other task decompositions for redesign. In the remainder of this section several problem-solving methods for redesign are compared to the general model for redesign advocated above.

Brown, Chandrasekaran and Goel (Brown & Chandrasekaran, 1989; Goel & Chandrasekaran, 1989) distinguish a family of methods called *propose-verify-redesign*. Chandrasekaran (Chandrasekaran, 1990) alternatively names this family *propose-critique-modify*. The following subtasks are distinguished in this family of problem-solving methods for design: propose, verify, critique and modify. The verify-critique-modify cycle portrays the *redesign* task in this family of design methods, while the propose task falls outside our current focus since the task of redesign supposes that an initial model has already been constructed before the redesign task starts. The propose-critique-modify approach is solely focused on the artifact to be redesigned, and does not pay any attention to the task of requirement management. Contrary to this classical view on redesign, in our general model of redesign the task of requirement management plays an important role. In our point of view, explicit management of requirements is essential in non-routine redesign: addition, retraction and modification of the original requirements often forms a major part of any non-routine redesign task. Examples of such non-routine redesign tasks can be found in systems for redesign of compositional architectures (Brazier *et al.*, 1996c), software specifications (Funk & Robertson, 1994) and in our own model revision system 007 (Pos & Akkermans, 1996a; Pos *et al.*, 1997b).

(Dixon, 1986) presents a class of methods for the mechanical design process, which can best be summarized as ‘Redesign WITHIN (re)specification WITHIN decomposition’. First, a mechanical design problem is decomposed into subproblems. This continues until the size and complexity of subproblems is reduced to the point where the subproblem can be managed without further decomposition. These subproblems are then solved by a process which Dixon calls ‘the redesign model of design’: First, a problem is specified

in terms of problem parameters. An initial design procedure generates an initial design. This trial design is evaluated. If the design is not acceptable, the design is redesigned and re-evaluated. If redesign ultimately fails to produce a solution the process returns to the initial step of problem specification. Again, the task of redesign is restricted to redesign of the design description, and specification of requirements is placed outside the boundary of redesign, together with the generation of an initial design. In the general REVISE model of redesign, the role of requirement management is not restricted to the role of 'preprocessor' to the task of redesign, but is an integral part of the redesign process.

Gero (Gero, 1990) has distinguished several forms of design: *routine*, *innovative* and *original* design. Brown and Chandrasekaran (Brown & Chandrasekaran, 1989) call these *Class 3,2* and *1*-design respectively. All of these forms of design can be realized by our generic model of redesign. The differences between the forms of design are most obvious in the respect to which the requirement management task is needed.

In routine design there is often no need for management of requirements; the initial requirements can be fulfilled by assessment and repair of the design description. Innovative design usually involves small changes to requirements (without retracting any) while original design involves extensive management of requirements. Whenever the goal is to produce an original (or quite innovative) design larger changes to requirements, including requirement retraction and requirement addition, may be involved. Also, several restrictions on the redesign process may need to be relaxed, for instance the minimality of changes to the design description. Explicit representation of these restrictions facilitates the different forms of design that can be performed by a system, which advocates the need for explicit reasoning about (design) strategies.

Redesign methods can be identified that perform design-tasks that are not restricted to the form of routine design. Examples can be found in systems for redesign of compositional architectures (Brazier *et al.*, 1996c), simulation models (this thesis), and software specifications (Funk & Robertson, 1994). Our characterization of redesign, as a collection of methods for various design tasks based on the same principle, is general enough to incorporate all three forms of redesign. A possible disadvantage of our characterization can be that it provides no hard distinction between redesign and other methods for design. Rather there is a continuum of methods, being more or less redesign oriented. This again is strong support for the claim that redesign can be seen as both an inherent part of design processes and as a family of design methods in itself.

8.4.2 Model revision as a form of redesign

In this thesis we have focused on the task of model revision. In this section we will show how our approach to model revision fits in the REVISE model for redesign presented in Figure 8.3. Model revision starts with a physical model and a set of (probably implicit) requirements on this model, and produces as output a new physical model and the current set of model requirements. A first decomposition of model-revision in terms of the general framework for redesign, is presented in Figure 8.4. This Figure shows the decomposition of model revision into different subtasks, as well as the data flow between subtasks. The control flow, i.e. the ordering of subtasks, is not represented in the figure, but will be explained in text in the next paragraph.

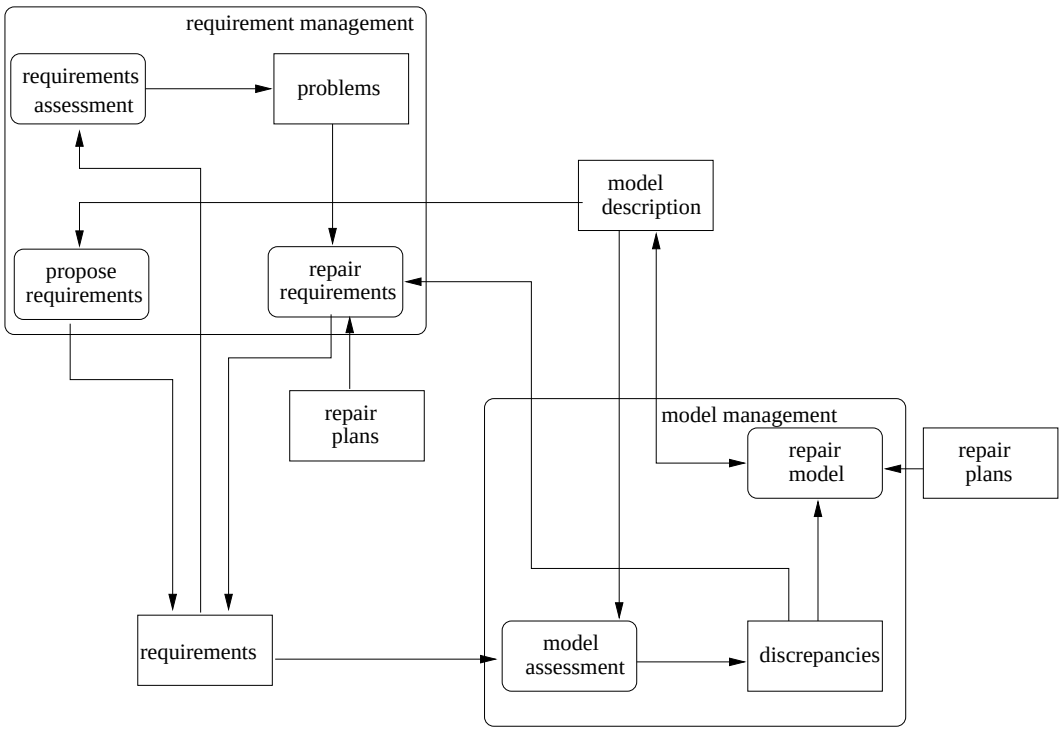


Figure 8.4: Plan-based model revision: subtasks and data flow.

In previous work, modeling has been shown to be a form of design (Top & Akkermans, 1994; Top, 1993). Therefore, it is not surprising that the task decomposition for model revision depicted in Figure 8.4 has many similarities to task decompositions for the more general task of redesign. In fact, the task description for model revision is a specialisation of the more general REVISE model for redesign presented earlier.

Model revision starts with a phase of *requirement management*, which specifies the set of requirements necessary for redesign in cooperation with the user: the user proposes

a set of one or more requirements, the system assesses these requirements with respect to consistency, completeness and operationality, and either the system or the user repairs the set of requirements until requirement assessment can detect no more problems. Our approach to requirement management has been discussed at length in chapter 4 of this thesis.

The *model assessment* task correspond to the task called *assessment* in Figure 8.3. In 007 this task consists of testing whether the current model meets the set of testable requirements, and producing *discrepancies*, indicating in which way the current model violates each specific requirement. Our approach to assessment has been discussed in more detail in chapter 5 of this thesis.

Based on the discrepancies between the current model and the requirements, either the model itself is adapted or the analysis method used to analyze the model. Since the analysis method to be used is part of the requirements, adapting the analysis method corresponds to repairing the set of requirements (by selecting a different analysis method). Both forms of adaptations use revision knowledge in the form of *repair plans* to link discrepancies to possible modifications. Our approach to adaptation has been discussed at length in chapter 5 of this thesis.

The cycle of assessment and adaptation is repeated until no more discrepancies exist. Of course, the entire process of model revision can be repeated if the modeler is not satisfied with the results, by specifying new or additional needs and demands.

8.5 Requirement management

The subtask of requirement management is responsible for specification, management, refinement and adaptation of requirements. This task has as its input the current model, and as its output it produces a correct set of requirements, suitable for assessment of the design description. Requirement management as a separate task is based on the observation that in general a design problem is often initiated by a statement of *needs and desires* (Bernaras, 1994). These, sometimes quite vague, needs and desires are to be interpreted and operationalized into a set of requirements suitable for automated assessment. This corresponds to the task of requirements engineering in software design (Wieringa, 1996). Figure 8.5 presents a partial task-method structure for the requirement management task.

We have identified two methods for this task: ASK-USER OPERATIONAL REQUIREMENTS

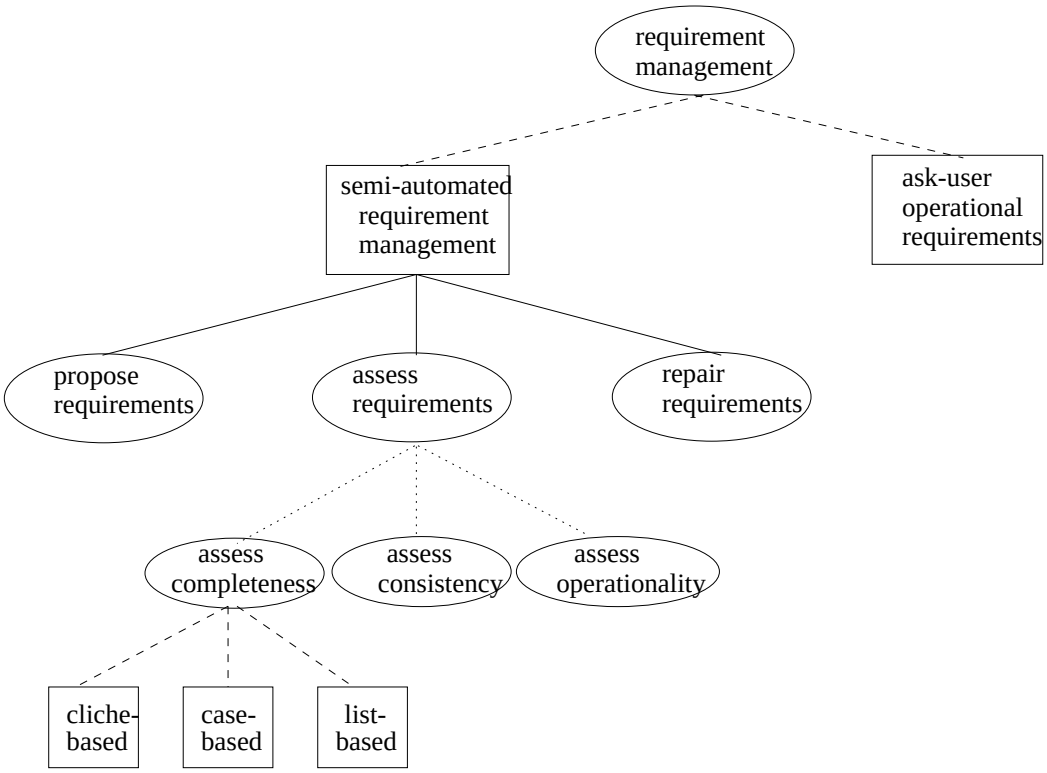


Figure 8.5: A partial task-method structure for the requirement management task. Rectangles represent methods and ellipses represent tasks. Dashed lines indicate that methods are alternatives for achieving the task goal. Solid lines decompose a method into its subtasks. Dotted lines indicate competence refinement of tasks or methods. Not all methods are decomposed into subtasks and primitive inferences.

and SEMI-AUTOMATED REQUIREMENT MANAGEMENT. In many redesign systems, the task of requirements specification is put completely in the hands of the user, and no automated support is provided. This corresponds to the ASK-USER OPERATIONAL REQUIREMENTS PSM. On the other hand, a small number of redesign systems (e.g. (Brazier *et al.*, 1996c; Pos *et al.*, 1997b; Reubenstein & Waters, 1991)) explicitly incorporate some form of semi-automated requirement management to ensure that the resulting set of requirements is correct and suitable for further processing in the assessment task.

A necessary ontological assumption for including any form of semi-automated requirement specification in a (re)design system is that knowledge on how requirements relate to each other is available in the application domain. The extent and nature of this knowledge determines which form(s) of requirement management are feasible. Requirement management is primarily useful when the set of possible requirements to be posed to the (re)design system is potentially large, when the number of requirements to be simultaneously satisfied may become large or when the need for expressing non-operational requirements arises in a (re)design system due to the complexity of the domain involved. In cases where the set of possible requirements is small and requirements are not apparently interacting, requirement management is usually not necessary.

The PSM SEMI-AUTOMATED REQUIREMENT MANAGEMENT decomposes the task of requirement management in three subtasks: PROPOSE-REQUIREMENTS, ASSESS-REQUIREMENTS and REPAIR-REQUIREMENTS. This decomposition corresponds to the family of methods called propose-verify-repair (Goel & Chandrasekaran, 1989), or propose-critique-modify (Chandrasekaran, 1990). This shows that this family of methods can not only be applied to the task of design, as already observed in section 8.3, but also to the task of requirement management. For the current example, we will focus on the ASSESS-REQUIREMENTS subtask in requirement management.

8.5.1 Requirement assessment

The assess-requirements task takes as input a set of requirements and produces a datum DA that states whether the requirements are correct or not.

task name:	requirement assessment
goal:	KNOWN(CORRECT(R)) $\vee$ KNOWN(INCORRECT(R))
input role:	R: set of requirements
output role:	DA: KNOWN(CORRECT(R)) $\vee DA$: KNOWN(INCORRECT(R))

By carefully looking at requirement management in different (re)design systems we distinguished three different *competence refinements* of this requirement-assessment task to more specialized tasks. Each of these refinements requires different teleological assumptions, providing different restrictions on the *goal* of the requirement assessment task: KNOWN(CORRECT(R)) $\vee$ KNOWN(INCORRECT(R)). The input and output roles stay the same: in each refinement the task takes as input a set of requirements R and delivers as output a statement on some aspect of the correctness or incorrectness of R.

Assess completeness

A possible way in which the competence of the assess requirement tasks can be refined is to refine the notion of 'correctness' to the more specialized notion of 'completeness'. The goal of the thus refined assess completeness task then becomes: KNOWN(COMPLETE(R)) $\vee$ KNOWN(INCOMPLETE(R)). This is an example of the introduction of additional teleological assumptions: the output of the assess requirements task is restricted from a statement about the *correctness* of the set of requirements to a more specialized statement about the *completeness* of the set of requirements.

The completeness of a set of requirements can only be assessed with respect to a specific problem or a specific set of problems. Problem-solving methods for assessing incompleteness of a requirement management in general use problem-specific knowledge to decide whether a requirement management is complete with respect to the problem(s) posed. This information can e.g. be represented in the form of cliches for different problems (Reubenstein & Waters, 1991; Maiden, 1995; Pos *et al.*, 1997b), in the form of cases (Maher & Balacandran, 1994) or in the form of a predefined list of requirements to be specified (Brazier *et al.*, 1996b). A necessary ontological assumption for each of these problem-solving methods is that knowledge on when a set of requirements is supposed to be complete is present in the domain knowledge.

Assess consistency

The second way in which the competence of the ASSESS-REQUIREMENT tasks can be refined is to refine the notion of 'correctness' to the more specialized notion of 'consistency'. The goal of the thus refined assess consistency task then becomes: KNOWN(CONSISTENT(R)) $\vee$ KNOWN(INCONSISTENT(R)).

Problem-solving methods for assessing the (in)consistency of a requirement set in general

use meta-knowledge on how requirements can interact. This knowledge can e.g. be presented in the form of deduction rules (Reubenstein & Waters, 1991; Brazier *et al.*, 1996b), or in the form of explicit constraints. Usually only prespecified inconsistencies can be detected. In our model revision system 007, predictable inconsistencies among classes of requirements are represented in the form of explicit constraints, and testing for inconsistencies is performed by checking whether these conditions hold.

Assess operationality

The second way in which the competence of the assess requirement tasks can be refined is to refine the notion of 'correctness' to the more specialized notion of 'operationality': can the truth of each requirement be directly derived from the design description? The goal of the thus refined assess operationality task then becomes: KNOWN(OPERATIONAL(R)) $\vee$ KNOWN(IN-OPERATIONAL(R)).

Problem-solving methods for assessing the operationality of a requirement management in general use meta-knowledge on which kind of requirements can or can not be automatically derived from the design description. This knowledge is often hard-coded in the definition of possible requirements. In our own model revision system 007 this knowledge is represented in an object-oriented manner: each requirement 'knows' whether it is operational or not.

All three tasks refinements described above are pure *competence refinements* of the original task goal ASSESS-REQUIREMENTS: they do not make any claims on how the task should be operationalized in terms of (control over) subtasks, but only on the way the task should be further *specified*. Due to their emphasis on specification instead of operationalization, these task refinements do not fit well in the original PSM framework. Our purpose in constructing these explicit task refinements has been to better distinguish different reasoning steps, corresponding to different design decisions, in the construction and classification of problem-solving methods. The next step would be to either refine the current task goal further to an even more specialized task, or to match sufficiently refined task goals to the functional description of one or more problem-solving methods which might be able to achieve the goal of the refined task.

Task refinements are not necessarily mutually exclusive. Since an actual system can perform multiple tasks, it can simultaneously incorporate more than one form of competence refinement for a single goal. In fact, our own model revision system 007 incorporates all

three competence refinements of the assess requirement task, i.e. it tests the set of requirements on completeness, consistency and operationality. For more details on how these tasks are operationalized see chapter 4 of this thesis.

8.6 Design modification

The task of design modification modifies part of the current design description to solve one or more discrepancies between the set of requirements and the current design description. Design modification is a subtask in the critique-modify method for the repair task. Figure 8.6 presents a partial task decomposition of this task. Design modification has as input (a part of) the design description and delivers as output a modified version of this part that is hopefully more suitable in the new situation. Three general problem-solving methods for design modification are *substitution*, *transformation* and *generation* (Smyth & Cunningham, 1993; Maher *et al.*, 1995). Substitution methods substitute a part of the old design description with a new part more suitable for the new situation. Transformation methods are used to transform an old solution into one that will (hopefully) work in the new situation. Generative methods re-enact (part of) the reasoning trace to modify the design description. FIRST (Daube & Hayes-Roth, 1989), a case-based system for redesign of mechanical systems, and COBRA (Finn *et al.*, 1992), a case-based system for redesign of heat-transfer models, provide examples of the latter approach.

Several dimensions of redesign (table 8.1) play a role as ontological assumptions in this choice between alternative problem-solving methods: whether the structure of the design is supposed to be fixed, and whether the reasoning trace or the end product of design is used for redesign. Generative methods make use of a reasoning trace, while substitution and transformation methods require only the end product of design in the form of a design description. Substitution and transformation methods mainly differ in their assumptions on system structure: substitution methods assume that the system structure is fixed, while transformation methods assume that the structure of the design is adaptable.

Substitution methods can be further specialized in direct substitution methods and substitution by search. This distinction is based on the dimension of redesign concerning the *plan-based versus search-based* nature of adaptation knowledge. This dimension defines ontological assumptions on the type of additional knowledge necessary to make the PSMs work: substitution by search requires some sort of database in which substitution elements can be found, while direct substitution methods require procedural knowledge

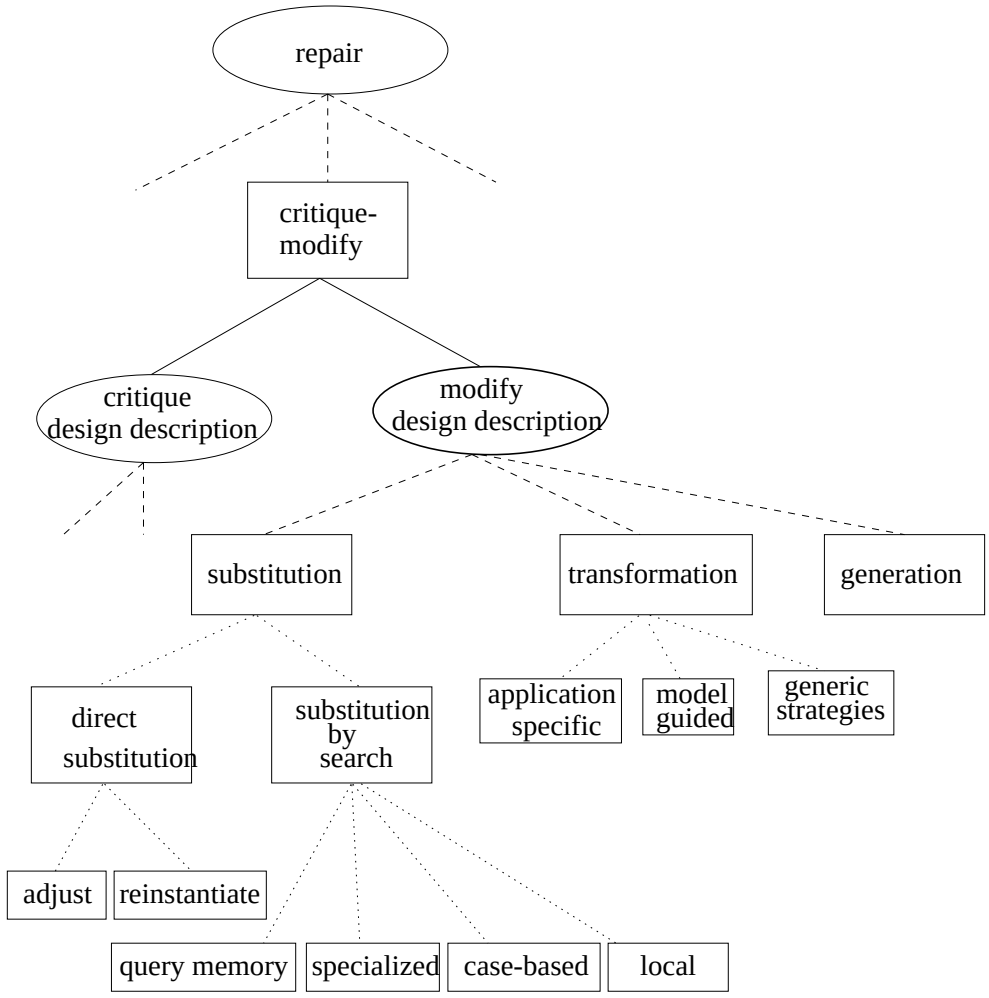


Figure 8.6: A partial task decomposition of the repair task.

on how to decide on a new value for a part of the old design description. Search-based substitution methods can be further specialized by considering the *additional knowledge* necessary for searching: query memory (Hinrichs & Kolodner, 1991) requires information on *what* to look for, in the form of a partial description of the item searched for. Local search, which can e.g. be found in PLEXUS (Alterman, 1986), requires instructions on *where* to search for alternatives. Specialized search, which can e.g. be found in the SWALE system (Schank & Leake, 1989), requires instructions on *how* to find an alternative. Case-based substitution, which can e.g. be found in the systems JULIA (Hinrichs & Kolodner, 1991), requires instructions on how to find a *similar* case which might suggest an appropriate alternative.

Transformation methods can be further specialized by considering the *general or specific* nature of the modification knowledge used for transformation: general strategies like 'divide-and-conquer', which can e.g. be found in JULIA (Hinrichs & Kolodner, 1991), form one end of this spectrum, while application-specific heuristics, which can e.g. be found in the VT-system for elevator design (Marcus *et al.*, 1987), form the other end of the spectrum. Our own model revision system 007 falls somewhere in between: similar to systems such as KRITIK (Goel, 1991) and CASEY (Koton, 1988), 007 uses repair plans to transform old design descriptions into new ones. The knowledge captured in these repair plans is generic enough to be applied to many different physical systems, but not as generic as general strategies like 'divide-and-conquer'.

All the method specializations discussed above are *competence refinements* of the functional specification of problem-solving methods: they only make claims on the functional specification of the PSM, and not on how the PSM should be operationalized in terms of subtasks and control. Due tot their emphasis on refinement instead of operationalization, they do not fit well in the original PSM framework. In the original PSM framework, the separate design decisions would either have be compressed into direct relations between the task and the bottom-level PSMs (e.g., one way to satisfy the task 'modify design description' is to directly use the bottom-level PSM 'substitution by local search'), or an intermediate operationalization would have to be constructed for each PSM in the refinement decomposition.

Method refinements represent *intermediate* decisions in the classification of problem-solving methods. Our purpose in constructing these intermediate relations has been to better distinguish different design decisions as separate steps in the classification of problem-solving methods, based on the dimension underlying each design decision. Method specializa-

tion ultimately results in bottom-level PSMs which can then be operationalized in terms of subtasks and control.

The task decomposition depicted in Figure 8.6 covers the same set of methods as the classification of adaptation methods and strategies in case-based reasoning presented in (Kolodner, 1993). This lends further support to our claim that redesign methods form a part of several more general design paradigms, of which case-based reasoning is one.

8.7 Conclusions

Redesign is an inherent part of most design processes, but can also be seen as a family of design methods in itself. In the latter point of view, redesign is a family of problem solving methods for which two questions need to be answered:"what is common to redesign?" and "which dimensions are involved in characterizing redesign?". The family of methods called redesign can be characterized by a number of common principles: redesign starts with an existing design description, and it requires some form of knowledge that allows the adaptation of designs. This knowledge is based on two underlying principles: (1) minimizing the changes being made to the design description and (2) maximizing the exploit of known properties of the design description. These characteristics can be used to differentiate this family of methods from other design methods. In order to compare different problem solving methods for the same task we distinguished a number of *dimensions* along which redesign methods can differ: the nature of the *design description*, the nature of the *requirements* and the nature of the *adaptation knowledge* are the three main sources of variation in the family of problem solving methods called redesign.

We presented a general model of redesign developed in the REVISE project. The two main subtasks in this general model are *requirement management* and *artifact management*. The task of requirement management has not often been modeled in other (re)design models, but is in our view essential in non-routine redesign. Requirement management in design corresponds to the task of requirements engineering in software design (Wieringa, 1996). Artifact management in its turn consists of assessment and repair. We have shown a number of problem-solving methods to exist for these subtasks. Each of these redesign subtasks can be achieved by one or more problem-solving methods. In this chapter we selected two subtasks , requirement management and design modification, and described informal but detailed task-structures for these subtasks. For the other redesign subtasks similar task-structures have been developed. We have also shown how our approach to

model revision fits into this general model of redesign: model revision is just another form of redesign in which simulation models are the subject to be redesigned.

In developing a knowledge-based system for a redesign task a large number of choices and decisions are made. In order to express these decisions as separate reasoning steps in a task-method structure, the current notion of possible relations between tasks and methods in a PSM architecture needs to be extended. Notions of *task refinement* and *method refinement* are introduced to represent intermediate decisions in a task-method structure, in which the *competence* of a task or method is refined without directly paying attention to its operationalization in terms of subtasks and control. Explicit representation of this kind of intermediate reasoning steps helps to make and represent decisions in a task-method structure in a more structured and piecemeal fashion.

The characterization of tasks and methods for redesign presented in this chapter is informal. We think that this type of characterization is useful in the early stages of KBS design, where problem-solving methods are constructed and selected to construct the skeleton design model for a knowledge-based system. In this early stage of KBS design, common sense descriptions of problem-solving methods and their underlying knowledge requirements (in terms of teleological and ontological assumptions made by each problem-solving method) will often be used in advance of formalisation in later stages of development. A first step in making this characterization more formal would be to formalize the dimensions for redesign, and thereby clarifying their interrelations.

In this chapter we have illustrated the extended PSM framework by describing some of the relevant choices in selecting problem-solving methods for our own model revision system 007. In the near future, we plan to use this framework to describe other redesign systems, starting with the redesign systems which are at the moment being developed by the other participants in the REVISE project. These systems focus on redesign of compositional architectures (Brazier *et al.*, 1996c) and redesign of control knowledge in knowledge based systems (Straatman, 1995), respectively.

Chapter 9

Conclusions

In this thesis we have investigated the following question:

> *How can the iterative nature of the engineering modeling process be supported?*

In this thesis we combined traditional engineering methods with knowledge-based techniques to achieve this goal. We discussed three ways in which automated support can be provided to facilitate reuse and redesign of engineering models:

- Automated support for specification of modeling requirements. Chapter 4 discussed how structuring of requirements by means of *goal cliches* can be used to guide the specification of requirements on the model to be reused.
- Automated assessment and adaptation of a model with respect to a set of modeling requirements. A general theory of assessment and adaptation by means of repair plans was outlined in chapter 5, while chapter 6 presented a specific example of a complex repair plan for identification of partially-known physical models from abstract data.
- Automated support for specification of modeling assumptions and corresponding model fragments. Proper specification of the modeling assumptions underlying model-(fragment)s facilitates their sharability and re-usability. Chapter 7 discussed how *cliches*, a knowledge-based technique first discussed in chapter 4, can also be used provide semi-automated support for the specification of modeling assumptions and corresponding model fragments.

Finally, in chapter 8, we discussed how model revision fits in the general context of *redesign*.

9.1 Contributions of this thesis

In this thesis we have shown that intelligent modeling environments are useful in supporting the inherently iterative nature of the modeling process. These modeling environments aid the user in specifying model requirements, perform assessment and adaptation of physical models and provide library support for specification and verification of reusable model entities.

In this thesis we made the following contributions to the theory of model revision:

1. We presented both a theory and an implementation for a *requirement assistant* which provides automated support for the specification of modeling requirements.
2. We presented a general theory for semi-automated *assessment and adaptation* by means of *repair plans*. We also presented a number of specific repair plans to substantiate this theory, and showed that both structural and parametric repairs can be performed. The theory has been implemented in a knowledge-based system for model revision, called 007.
3. We presented an algorithm for *parametric identification* as an example of a complex, parametric repair plan, in which numerical and knowledge-based techniques are combined to perform a complex engineering task: parametric system identification.
4. We demonstrated that automated support can be useful in the context of *specification of modeling assumptions* and associated model fragments.
5. We have presented a general view on *redesign as a family of methods*, and discussed relevant dimensions in characterizing methods for this task. We have shown that model revision is a special form of the more general task of redesign, in which a computational model is the subject to be redesigned. We have also shown that, in order to capture all relevant choices in constructing and comparing knowledge-based systems for redesign, it is necessary to incorporate the notion of *competence refinement* in a general PSM framework.

9.2 Suggestions for further research

Future research should be directed at developing modification tactics for other modeling problems. To facilitate this, the following directions should be examined:

- Extend the ranges of problems to be solved by automated assessment and adaptation by extending the theory on plan-based repair to include assumption-based instead of only construction-based techniques.
- Extend the support for requirement specification and requirement management, by extending the scope of specification from modeling goals to more general design goals.
- Further investigate the use of semi-automated support for specification of modeling assumptions and associated model fragments in different application domains.

Bibliography

ADDANKI, S., CREMONINI, R., & PENBERTHY, J. S. (1991). Graphs of models. *Artificial Intelligence*, 51(1-3):145–177.

AKKERMANS, H., WIELINGA, B., & SCHREIBER, G. (1994). Steps in constructing problem solving methods. In Gaines, B. R. & Musen, M., editors, *Proceedings of the 8th International Knowledge Acquisition Workshop (KAW'94)*, volume 2, pages 29.1–29.21, Banff, Alberta. University of Calgary, SRDG Publications.

ALTERMAN, R. (1986). An adaptive planner. In Kehler, T., Rosenschein, S., Filman, R., & Patel-Schneider, P. F., editors, *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pages 65–69, Philadelphia, PA. Morgan Kaufman Publishers, Inc.

AMSTERDAM, J. (1992). Automated modeling of physical systems. In Falkenhainer, B. & Stein, J. L., editors, *Automated modeling*, volume DSC-41, pages 31–36. ASME.

ASTROM, K. & EYKHOFF, P. (1971). System identification: A survey. *Automatica*, 7:123–162.

BENJAMINS, V. R. (1993). *Problem solving methods for diagnosis*. Ph. D. Thesis, University of Amsterdam.

BENJAMINS, V. R., FENSEL, D., & STRAATMAN, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In Wahlster, W., editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 408–412, Budapest, Hungary. John Wiley and Sons.

BERNARAS, A. (1994). Problem-oriented and task-oriented models of design in the COMMONKADS framework. In Gero, J. S. & Sudweeks, F., editors, *Artificial Intelligence in Design '94*. Dordrecht, the Netherlands, Kluwer Academic Publishers.

BISWAS, G., YU, X., & DEBELAK, K. (1993). A formal modeling scheme for continuous-valued systems: Focus on diagnosis. In Bajscy, R., editor, *Proceedings of the 13th Interational Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1474–1479, Chambery, France. Morgan Kaufman Publishers.

BOBROW, D. G., editor (1984). *Qualitative reasoning about physical systems*. Amsterdam, North-Holland.

BORST, W. N., AKKERMANS, J. M., & TOP, J. L. (1997). Engineering ontologies. *International Journal of Human-Computer Studies (in press)*, 46:365–406.

BRAZIER, F., TREUR, J., WIJNGAARDS, N., & WILLEMS, M. (1995). Formal specification of hierarchically (de)composed tasks. In Gaines, B. R. & Musen, M. A., editors, *Proceedings of the 9th Banff knowledge acquisition for knowledge-based systems workshop (KAW'95)*, volume 2, pages 15.1–15.20, Banff, Alberta. University of Calgary, SRDG Publications.

BRAZIER, F. M. T., TREUR, J., & WIJNGAARDS, N. J. E. (1996a). Interaction with experts: the role of a shared task model. In Wahlster, W., editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 241–245, Budapest, Hungary. Wiley and Sons.

BRAZIER, F. M. T., VAN LANGEN, P. H. G., RUTTKAY, Z., & TREUR, J. (1994). On formal specification of design tasks. In Gero, J. S. & Sudweeks, F., editors, *Artificial Intelligence in Design '94*, pages 535–552. Dordrecht, Kluwer Academic Publishers.

BRAZIER, F. M. T., VAN LANGEN, P. H. G., TREUR, J., & WIJNGAARDS, N. J. E. (1996c). Redesign and reuse in compositional knowledge-based systems. *Knowledge Based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):105–119.

BRAZIER, F. M. T., VAN LANGEN, P. H. G., TREUR, J., WIJNGAARDS, N. J. E., & WILLEMS, M. (1996b). Modelling an elevator design task in DESIRE: the VT example. *International Journal of Human-Computer Studies*, 46:469–520.

BREUNESE, A. P. J. (1996). *Automated support in mechatronic systems design.* Ph. D. Thesis, University of Twente, Enschede.

BREUNESE, A. P. J. & BREEDVELD, P. C. (1996). Analysis of equation submodels. *Mathematical Modelling of Systems*, 2(2):134–156.

BROWN, D. & CHANDRASEKARAN, B. (1989). *Design problem solving: knowledge structures and control strategies*. San Mateo, CA, Morgan Kaufmann.

BROWN, F. T. (1972). Direct application of the loop rule to bond graphs. *Journal of Dynamic Systems, Measurement and Control*, 94(3):253–261.

CARBONELL, J. G. (1983). Derivational analogy and its role in problem solving. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-83)*, pages 64–69, Washington, D.C. Morgan Kaufman Publishers, Inc.

CHANDRASEKARAN, B. (1988). Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *The Knowledge Engineering Review*, 3:183–210.

CHANDRASEKARAN, B. (1990). Design problem solving: a task analysis. *AI Magazine*, 11(4):59–71.

CROSS, N. (1989). *Engineering Design Methods*. Chichester, John Wiley & Sons.

DAUBE, F. & HAYES-ROTH, B. (1989). A case-based mechanical redesign system. In Shridharan, N. S., editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Michigan. Morgan Kaufman Publishers, Inc.

DAUPHIN-TANGUY, G., BORNE, P., & LEBRUN, M. (1985). Order reduction of multi-time scale systems using bond-graphs, the reciprocal system and the singular perturbation method. *Journal of the Franklin Institute*, 319(1/2):151–171.

DE KLEER, J. & BROWN, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83.

DIJK, J. & BREEDVELD, P. (1991). Simulation of system models containing zero-order causal paths. *Journal of the Franklin Institute*, 328(5/6):959–1004.

DIXON, J. R. (1986). Artificial intelligence and design: a mechanical engineering view. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pages 872–877, Philadelphia, PA. Morgan Kaufman Publishers, Inc.

ELDONK, S. J. M., ALBERTS, L., BAKKER, R., F.DIKKER, & WOGNUM, P. (1996). Redesign of technical systems. *Knowledge-Based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):93–104.

FALKENHAINER, B. & FORBUS, K. D. (1991). Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51(1-3):95–143.

FENSEL, D. & BENJAMINS, R. (1996). Assumptions in model-based diagnosis. In Gaines, B. R. & Musen, M., editors, *Proceedings of the 10th Banff Knowledge Acquisiton for Knowledge-Based Systems Workshop (KAW-96)*, pages 5.1–5.16, Banff, Canada.

FENSEL, D. & STRAATMAN, R. (1996). The essence of problem-solving methods: making assumptions for efficiency reasons. In *Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW-96)*, pages 17–32, Nottingham, England.

FINN, D. P. (1993). A physical modeling assistant for the preliminary stages of finite element analysis. *Artificial Intelligence in Engineering, Design and Manufacturing (AI EDAM)*, 7(4):275–286.

FINN, D. P., GRIMSON, J. B., & HARTY, N. M. (1992). An intelligent modelling assistant for preliminary analysis in design. In Gero, J., editor, *Artificial intelligence in Design (AID'92)*, pages 579–596. Dordrecht, Kluwer Academic Publishers.

FISCHER, G., LEMKE, A. C., & RATHKE, C. (1987). From design to redesign. In *Proceedings of the 9th International Conference on Software Engineering*, pages 369–376, Washington, D.C. IEEE Computer Society Press.

FISHWICK, P. A. (1991). Toward an integrated approach to simulation model engineering. *International Journal on General Systems*, 17:1–19.

FORBUS, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

FORBUS, K. D. & FALKENHAINER, B. (1990). Self-explanatory simulations: an integration of qualitative and quantitative knowledge. In Dieterich, T. & Swartout, W., editors, *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 380–387. AAAI Press/MIT Press.

FOSSARD, A. J., DAUPHIN-TANGUY, G., & BORNE, P. (1987). Modelling, complexity and control. In Borne, P. & Tzafestas, S. G., editors, *Applied modeling and simulation of technological systems*, pages 115–127. Amsterdam, Elsevier Science Publishers BV.

FRANKE, D. (1991). Deriving and using descriptions of purpose. *IEEE Expert*, 6(2):41–47.

FUNK, P. J. & ROBERTSON, D. (1994). Case-based support for the design of dynamic system requirements. In *Proceedings of the 2nd European Workshop on Advances in Case-Based Reasoning (EWCBR-94)*, pages 211–225, Chantilly, France.

GAWTHROP, P. J., JONES, R. W., & MACKENZIE, S. A. (1992). Identification of partially-known systems. *Automatica*, 28(4):831–836.

GENNARI, J., TU, S., ROSENFLUH, T., & MUSEN, M. (1994). Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41:399–424.

GENTIL, S., BARRAUD, A. Y., & SZAFNICKI, K. (1990). Sexi: An expert identification package. *Automatica*, 26(4):803–809.

GERO, J. S. (1990). Design prototypes: a knowledge representation schema for design. *AI Magazine*, 11(4):26–36.

GOEL, A. & CHANDRASEKARAN, B. (1989). Functional representation of design and redesign problem solving. In Shridharan, N. S., editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Michigan. Morgan Kaufman Publishers, Inc.

GOEL, A. K. (1991). A model-based approach to case adaptation. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society (CogSci'91)*, pages 143–148, Chicago, Illinois.

GOEL, A. K. (1996). Adaptive modeling. In ?, editor, *Proceedings 10th International Workshop on Qualitative Reasoning*, page ?, Stanford Cierra Camp, CA. AAAI Press.

HAEST, M., BASTIN, G., GEVERS, M., & WERTZ, V. (1990). Espion: an expert system for system identification. *Automatica*, 26:85–95.

HAMMING, R. W. (1962). *Numerical methods for scientists and engineers*. NewYork, McGraw-Hill Book Company. 2nd edition.

HINRICHS, T. & KOLODNER, J. (1991). The roles of adaptation in case-based design. In Dean, T. & McKeown, K., editors, *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 28–33. AAAI Press / The MIT Press.

HOWE, A., COHEN, P., DIXON, J., & SIMMONS, M. (1986). Dominic: a domain-independent program for mechanical engineering design. In Sriram, D. & Adey, R.,

editors, *Applications of AI in Engineering problems, proceedings of the 1st International Conference*, pages 289–299, Southampton University, U.K. Springer-Verlag.

ISHII, M. & TOMIYAMA, T. (1996). A synthetic reasoning method based on a physical phenomenon knowledge base. In Sharpe, J., editor, *AI System Support for Conceptual Design, Proceedings of the 1995 Lancaster International Workshop on Engineering Design*, pages 109–123, London. Springer-Verlag.

IWASAKI, Y. (1988). Causal ordering in a mixed structure. In Smith, R. G. & Mitchell, T. M., editors, *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 313–318, Saint-Paul, Minnesota. Morgan Kaufman Publishers.

IWASAKI, Y. & BHANDARI, I. (1988). Formal basis for commonsense abstraction of dynamic systems. In Smith, R. G. & Mitchell, T. M., editors, *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 307–312, Saint-Paul, Minnesota. Morgan Kaufman Publishers.

IWASAKI, Y. & LEVY, A. Y. (1993). Automated model selection for simulation. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning about Physical Systems*, pages 108–116, Seattle, WA.

IWASAKI, Y. & SIMON, H. (1986). Causality in device behavior. *Artificial intelligence*, 29:3–32.

KARNOPP, D. C., MARGOLIS, D. L., & ROSENBERG, R. C. (1990). *System Dynamics: a unified approach*. New York, John Wiley & Sons. Second Revised Edition.

KERSTEN, M. L. (1986). A conceptual modelling expert system. In Spaccapietra, S., editor, *Proceedings of the 5th International Conference on Entity-Relationship Approach*, pages 275–288, Dijon, France. IEEE Computer Society.

KLINKER, G., BHOLA, C., DALLEMAGNE, G., MARQUES, D., & MCDERMOTT, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3:117–136.

KOKOTOVIC, P. V., O'MALLEY, R. E., & SANNUTI, P. (1976). Singular perturbations and order reduction in control theory – an overview. *Automatica*, 12:123–132.

KOLODNER, J. (1993). *Case-Based Reasoning*. Morgan Kaufman Publishers, Inc.

KOSUT, R. L., GOODWIN, G. C., & M. P. POLIS (EDS.) (1992). Special issue on system identification for robust control. *IEEE Transactions on Automatic Control*, 37(7).

KOTON, P. (1988). Reasoning about evidence in causal explanation. In Smith, R. G. & Mitchell, T. M., editors, *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 256–261, Saint Paul, Minnesota. AAAI Press/MIT Press.

KOWALCZYK, W. & TREUR, J. (1990). On the use of a formalized generic task model in knowledge acquisition. In Wielinga, B., Boose, J., Gaines, B., Schreiber, A., & van Someren, M., editors, *Current Trends in Knowledge Acquisition - Proceedings of the European Knowledge Acquisition Workshop (EKAW'90)*, pages 189–221. IOS Press.

KRAUS, T. & MYRON, T. (1984). Self-tuning PID controller uses pattern recognition approach. *Control Engineering*, 31(6):106–111.

KUIPERS, B. (1986). Qualitative simulation. *Artificial Intelligence*, 29:289–338.

KUIPERS, B. (1987). Abstraction by timescale in qualitative simulation. In Forbus, K. & Shrobe, H., editors, *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pages 621–625, Seattle, Washington. AAAI Press/MIT Press.

KUIPERS, B. & BERLEANT, D. (1988). Using quantitative knowledge in qualitative reasoning. In Smith, R. G. & Mitchell, T. M., editors, *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 324–329, Saint-Paul, Minnesota. Morgan Kaufman Publishers.

LEVY, A. Y., IWASAKI, Y., & MOTODA, H. (1992). Relevance reasoning to guide compositional modeling. In *Working notes of the AAAI-92 Workshop on approximation and abstraction of computational theories (AACT-92)*, pages 146–153, San Jose, California. AAAI Press.

LING, R., STEINBERG, L., & JALURIA, Y. (1993). MSG: A computer system for automated modeling of heat transfer. *Artificial Intelligence in Engineering, Design and Manufacturing (AI EDAM)*, 7(3):287–300.

LJUNG, L. (1987). *System identification: Theory for the User*. Englewood Cliffs, NJ, Prentice Hall.

LJUNG, L. & GLAD, T. (1994a). *Modeling of dynamic systems*. Englewood Cliffs, NJ, Prentice Hall.

LJUNG, L. & GLAD, T. (1994b). On global identifiability for arbitrary model parametrizations. *Automatica*, 30(2):265–276.

MAHER, M. & BALACANDRAN, B. (1994). Flexible retrieval strategies for case-based design. In Gero, J. & Sudweeks, F., editors, *Artificial Intelligence in Design '94*. Dordrecht, Kluwer Academic Publishers.

MAHER, M. L., BALACHANDRAN, M. B., & ZHANG, D. M. (1995). *Case-based reasoning in design*. Hove, UK, Lawrence Erlbaum Associates.

MAIDEN, N. A. M. (1995). Reuse-oriented requirements engineering in NATURE. *Software Engineering Notes*, 20(3):90–93.

MANNINO, M. V., CHOOBINEH, J., & HWANG, J. J. (1986). Acquisition and use of contextual knowledge in a form-driven database design methodology. In Spaccapietra, S., editor, *Proceedings of the 5th International Conference on Entity-Relationship Approach*, pages 141–157, Dijon, France. IEEE Computer Society.

MARCUS, S., STOUT, J., & MCDERMOTT, J. (1987). VT: an expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 8(4):39–58.

MARGOLIS, D. L. & YOUNG, G. E. (1977). Reduction of models of large scale lumped structures using normal modes and bond graphs. *Journal of the Franklin Institute*, 304(1):65–79.

MCDERMOTT, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Boston, Kluwer.

MITCHELL, T. M., STEINBERG, L. I., KEDAR-CABELLI, S., KELLY, V. E., SHULMAN, J., & WEINRICH, T. (1983). An intelligent aid for circuit redesign. In *Proceedings of the 3th National Conference on Artificial Intelligence (AAAI-83)*, Washington, D. C. William Kaufman, Inc.

MOSTOW, J. (1989). Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40:119–184.

MUSEN, M. A. (1989). *Automated generation of Model-Based Knowledge-Acquisition Tools*. London, Pitman.

NAGY, P. A. J. & LJUNG, L. (1992). System identification using bondgraphs. In Dugard, L., MSaad, M., & Landau, I. D., editors, *Proceedings of the 4th IFAC Adaptive Sys-*

tems in Control and Signal Processing, pages 61–66, Grenoble, France. International Federation for Automatic Control, Pergamon Press.

NATURE TEAM (1996). Defining visions in context: models, processes and tools for requirements engineering. *Information Systems*, 21(6):515–547.

NAYAK, P. P. & JOSKOWICZ, L. (1996). Efficient compositional modeling for generating causal explanations. *Artificial Intelligence*, 83(2):193–228.

NAYAK, P. P., JOSKOWICZ, L., & ADDANKI, S. (1992). Automated model selection using context-dependent behaviors. In Rosenbloom, P. & Szolovits, P., editors, *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 710–716, San Jose, CA. AAAI Press.

O'HARA, K. & SHADBOLT, N. (1993). Locating generic tasks. *Knowledge Acquisition*, 5:449–481.

PAHL, G. & BEITZ, W. (1996). *Engineering Design: A Systematic Approach*. Berlin, Springer.

PALM, W. (1983). *Modeling, Analysis and Control of Dynamic Systems*. New York, John Wiley & Sons.

PAYNTER, H. M. (1961). *Analysis and design of engineering systems*. Cambridge, MA, MIT Press.

POS, A. & AKKERMANS, J. M. (1996a). 007: A system for automated model revision. In Javar, A., Lehmann, A., & Molnar, I., editors, *Proceedings of the 10th European Simulation Multiconference (ESM'96)*, pages 50–54, Budapest, Hungary. SCS.

POS, A. & AKKERMANS, J. M. (1996b). Using functional abstractions for model-based identification. In McDowell, J. K., editor, *Working Notes of the AAAI-96 Workshop on Modeling and Reasoning with Function*, pages 8–13, Portland, Oregon.

POS, A. & AKKERMANS, J. M. (1997a). Requirement specification for engineering modeling. In *Proceedings of the Tenth Florida Artificial Intelligence Research Symposium (FLAIRS'97)*. In press.

POS, A. & AKKERMANS, J. M. (1997b). Using behavioral abstractions for model parameter identification. To be presented as a poster at the International Joint Conference on Artificial Intelligence (IJCAI'97).

POS, A., AKKERMANS, J. M., & STRAATMAN, R. (1997a). Problem solving for redesign. In *Working notes of the IJCAI-97 Workshop on Problem-solving Methods for Knowledge-Based Systems*. In Press.

POS, A., AKKERMANS, J. M., & TOP, J. L. (1997b). Automated model revision. *IEEE Expert*. In Press.

POS, A., BORST, P., TOP, J. L., & AKKERMANS, J. M. (1994). Reusability of simulation models. In Wognum, P. M., editor, *Proceedings of the 'Workshop Models and Techniques for Reuse of Designs'(ECAI'94)*, Amsterdam, The Netherlands.

POS, A., BORST, P., TOP, J. L., & AKKERMANS, J. M. (1996a). Reusability of simulation models. *Knowledge-Based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9:119–125.

POS, A., WIJNGAARDS, N. J. E., STRAATMAN, R., & REVISE (1996b). Choices in problem solving methods for redesign. IR-419, REVISE-3, Department of Computer Science and Mathematics, Vrije Universiteit van Amsterdam.

RAKE, H. (1980). Step response and frequency response methods. *Automatica*, 16:519–526.

REDFIELD, R. C. (1992). Bond graphs as a tool in mechanical systems simulators. In Falkenhainer, B. & Stein, J. L., editors, *Automated modeling*, volume DSC-41, pages 59–65. ASME.

REUBENSTEIN, H. B. & WATERS, R. C. (1991). The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, 17(3):226–240.

RICH, C. & WATERS, R. C. (1988). The programmer's apprentice project: a research overview. *Computer*, 21(11):10–25.

RICHARDS, B. L., KRAAN, I., & KUIPERS, B. J. (1992). Automatic abduction of qualitative models. In Rosenbloom, P. & Szolovits, P., editors, *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 723–728, San Jose, CA. AAAI Press.

RICKEL, J. & PORTER, B. (1992). Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In Hayes-Roth, B. & Korf, R., editors, *Proceedings of the 12th National Conference on Artificial Intelligence*

(AAAI-94), pages 1191–1198. AAAI Press/MIT Press.

ROSENBERG, R. C. & ANDRY, A. N. (1979). Solvability of bond graph junction structures with loops. *IEEE Transactions on circuits and systems*, 26(2):130–137.

SAY, A. C. C. & KURU, S. (1996). Qualitative system identification: deriving structure from behaviour. *Artificial Intelligence*, 83(1):75–142.

SCHANK, R. C. & LEAKE, D. B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40:353–385.

SCHUT, C. & BREDEWEG, B. (1993a). Automatic enhancement of model parsimony. In Weld, D. S., editor, *Proceedings of the 7th international workshop on qualitative reasoning about physical systems*, pages 194–203, Orcas Island, Washington, USA. University of Washington.

SCHUT, C. & BREDEWEG, B. (1993b). Interactive qualitative model construction. In *Proceedings of the IJCAI-workshop on Machine Learning and Knowledge Acquisition*, pages 172–186, Chamberey, France. IJCAI-93.

SCHUT, C. & BREDEWEG, B. (1994). Supporting qualitative model specification. In *Proceedings of the Second International Conference on Intelligent Systems Engineering*, pages 37–42, Hamburg, Germany. University of Hamburg-Harburg.

SCHUT, C. & BREDEWEG, B. (1995). Supporting qualitative model construction: eliminating incorrectly predicted derivatives. In *Proceedings of the 9th International Workshop on Qualitative Reasoning*, pages 163–172, Amsterdam, The Netherlands.

SCHUT, C. & BREDEWEG, B. (1996). An overview of approaches to qualitative model construction. *The Knowledge Engineering Review*, 11(1):1–25.

SHEN, Q. & LEITCH, R. (1990). A semi-quantitative extension to qualitative simulation. In *Proceedings of the 10th International Workshop on expert systems and their applications*, pages 223–234, Avignon.

SHEPHARD, M. S. & WENTORF, R. (1994). Toward the implementation of automated analysis idealization control. *Applied Numerical Analysis*, 14(1-3):105–124.

SMITH, N., XIA, S., & LUKER, P. (1996). A library-based automated intelligent modeller. In *Proceedings Summer Computer Simulation Conference*, pages 611–614.

SMYTH, B. & CUNNINGHAM, P. (1993). Complexity of adaptation in real-world case-based reasoning systems. In *Artificial Intelligence & Cognitive Science VI*. Queens University Press.

SMYTH, B. & KEANE, M. T. (1996). Using adaptation knowledge to retrieve and adapt design cases. *Knowledge-based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):127–136.

SÖDERMAN, U. & STRÖMBERG, J. E. (1991). Combining qualitative and quantitative knowledge to generate models of physical systems. In Mylopoulos, J. & Reiter, R., editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 1158–1163, Sydney, Australia. Morgan Kaufman Publishers.

STEELS, L. (1990). Components of expertise. *AI Magazine*, 11 (2):28–49.

STRAATMAN, R. (1995). Learning control knowledge in models of expertise. In Fensel, D., editor, *Proceedings of the ECML-95 Workshop on Knowledge Level Modeling and Machnine Learning*, pages I.2.1–I.2.13, Heraklion, Greece.

SUEUR, C. & DAUPHIN-TANGUY, G. (1991). Bond graph approach to multi-time scale systems analysis. *Journal of the Franklin Institute*, 328(5/6):1005–1026.

TOP, J. (1993). *Conceptual modelling of physical systems*. Ph.D. Thesis, University of Twente, Enschede.

TOP, J., BORST, P., & AKKERMANS, J. (1995a). 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., BREUNESE, A., BROENINK, J., & AKKERMANS, J. (1995b). Structure and use of a library for physical systems models. In Cellier, F. E. & Granda, J. J., editors, *Proceedings International Conference on Bond Graph Modelling and Simulation ICBGM'95*, pages 97–102, Las Vegas. SCS.

TOP, J. L. & AKKERMANS, J. M. (1991). Qualitative reasoning about physical systems - an artificial intelligence perspective. *Journal of the Franklin Institute*, 328:1047–1065.

TOP, J. L. & AKKERMANS, J. M. (1994). Tasks and ontologies in engineering modelling. *International Journal on Human-Computer Studies*, 41(4):585–617.

WALTER, E. (1982). *Identifiability of state space models*. Berlin, Springer.

WELD, D. S. (1990). Approximation reformulations. In Dietterich, T. & Swartout, W., editors, *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 407–412, Cambridge, MA. AAAI Press/MIT Press.

WELD, D. S. (1992a). Generating simplified models with confidence. In Falkenhainer, B. & Stein, J. L., editors, *Automated modeling*, volume DSC-41, pages 21–29. ASME.

WELD, D. S. (1992b). Reasoning about model accuracy. *Artificial Intelligence*, 56(2-3):255–300.

WELD, D. S. & DE KLEER, J. (1990). *Readings in Qualitative Reasoning about Physical Systems*. San Mateo, CA, Morgan Kaufman.

WIELINGA, B. J., AKKERMANS, J. M., & SCHREIBER, A. T. (1995). A formal analysis of parametric design. In *Proceedings of the 9th International Knowledge Acquisition Workshop (KAW'95)*, volume 2, pages 37.1–37.15, Banff, Alberta. University of Calgary, SRDG Publications.

WIELINGA, B. J. & SCHREIBER, A. T. (1997). Configuration-design problem solving. *IEEE Expert (in press)*.

WIELINGA, B. J., VELDE, W. V. D., SCHREIBER, A. T., & AKKERMANS, J. M. (1993). Towards a unification of knowledge modeling approaches. In David, J. M., Krivine, J. P., & Simmons, R., editors, *Second-generation expert systems*, chapter 14, pages 299–335. Berlin, Springer-Verlag.

WIERINGA, R. J. (1996). *Requirements Engineering: Frameworks for Understanding*. Chicester, England, John Wiley and Sons.

WILSON, B. H. & STEIN, J. L. (1995). An algorithm for obtaining proper models of distributed systems. *Journal of Dynamic Systems, Measurement, and Control*, 117:534–540.

XIA, S. (1994). Formulation of generic principles of modelling industrial systems for simulation. *Systems analysis, modelling, simulation: journal of mathematical modelling and simulation in systems analysis*, 15(4):283–291.

XIA, S., LINKENS, D. A., & BENNETT, S. (1992). Integration of qualitative reasoning and bond graphs: an engineering approach. In Breedveld, P. C. & Dauphin-Tanguy,

G., editors, *Bond graphs for engineers*, pages 323–332, Amsterdam. North-Holland.

XIA, S. & SMITH, N. (1996). Automated modelling: a discussion and review. *The Knowledge Engineering Review*, 11(2):137–160.

XU, C. W. & LU, Y. Z. (1987). Fuzzy model identification and self-learning for dynamical systems. *IEEE Transactions on Systems, Man and Cybernetics*, 17:683–689.

ZIEGLER, J. G. & NICHOLS, N. B. (1942). Optimum settings for automatic controllers. *Transactions of the ASME*, 64:759–768.

Curriculum Vitae

Anita Pos was born on December 18th 1968 in Utrecht, the Netherlands. She completed secondary school at the College Blaucapel in Utrecht in 1987. From 1986 to 1992 she studied Information Science at the University of Twente and received her M.Sc.-degree in May 1993 in the field of Artificial Intelligence.

From 1993 to 1997 she worked as a PhD student in the Information Systems group at the University of Twente. Her research topics were Artificial Intelligence and physical modeling. In these four years she published several papers at international conferences and in international journals.