Functional Models in Configuration Systems

From the
Department of Mathematics and Computer Science
of the University of Paderborn, Germany

The Accepted Dissertation of
Benno Maria Stein

In Order to Obtain the Academic Degree of
Dr. rer. nat.

Advisor:	Prof. Dr. Hans Kleine Büning
Reading Committee:	Prof. Dr. Michael M. Richter
	Prof. Dr. Wilhelm Schäfer

Date of Oral Examination: June 30, 1995

Acknowledgments

This thesis developed from my work as a research and teaching assistant at the Institute for Applied Computer Science, Department of Mathematics, University of Duisburg, and at the Institute for Applied Computer Science/Knowledge-based Systems, Department of Mathematics and Computer Science, University of Paderborn.

I wish to thank Professor Dr. Hans Kleine Büning, my thesis advisor, for supporting me and my entire work. Furthermore, I wish to thank Professor Dr. Michael M. Richter and Professor Dr. Wilhelm Schäfer for evaluating this thesis.

Finally, I would like to thank all my colleagues for their valuable discussions throughout the development of this thesis, especially Daniel Curatolo, Marcus Hoffmann, Oliver Najmann, and Jürgen Weiner.

Benno Stein

Contents

Preface **vii**

I Classification and Formal Framework

1 Introduction **1**

1.1 A First Glance 1

1.2 Knowledge-based Systems 4

1.3 A Short Discussion of Problem Classes 10

2 Classifying Configuration Problems **15**

2.1 Configuration Scenarios 15

2.2 Relating Configuration to Design 21

2.3 A Classification Scheme 30

2.4 Configuration Methods 41

3 A Formal Framework of Configuration **47**

3.1 Property-based Configuration Problems 48

3.2 Structure-based Configuration Problems 53

3.3 Behavior-based Configuration Problems 57

3.4 Structural versus Functional Descriptions 64

3.5 A Complexity Consideration 69

II Operationalizing Configuration Tasks

4 On Property-based Configuration **75**

4.1 The Rationale of Property-based Configuration 76

4.2 A Basic Algorithm 81

4.3 Improving Performance with Metaknowledge 83

4.4 The Configuration System MOKON 87

5 Behavior-based Configuration in Hydraulics **89**

5.1 Configuration of Hydraulic Systems 90

5.2 Analyzing the Analysis of Hydraulic Systems 94

5.3 Modeling Hydraulic Systems 98

5.4 Solving Π^{c}_{M3} in Hydraulics 103

5.5 Preprocessing of Stationary Behavior 108

6 The *artdeco* System **117**

6.1 Graphic Problem Specification 118

6.2 Inference 120

6.3 Knowledge Acquisition 128

6.4 Realization 130

Summary & Conclusion **135**

References **139**

Preface

In the past much effort has been spent to solve a variety of configuration problems through the use of computers. Why did configuration problems become so interesting?

One major reason is that the diversity of the customers' demands force manufacturers to tailor their products to these demands. Consequently, manufacturers are faced with large sets of variants and complex technical dependencies. These dependencies must be considered when composing the products. In this situation configuration systems are intended to master the process of composition.

Furthermore, configuration systems play a central role in the processing of orders: The design of complex technical systems can be simplified. At the customer's site, the sales personnel can utilize configuration systems to concentrate on sales consultation. Also, when given a definite order, configuration systems can provide information as input for planning and scheduling, production, stock keeping, and logistics.

There is no disagreement around the key role that configuration systems play in bridging the gap between the technical clarification of orders and a highly automated production run.

Objective of Research

In many configuration systems a taxonomical and a compositional hierarchy, i.e. a structural model of the domain, forms the basis for the configuration process. Such a modeling approach is no longer adequate when functional connections make up a major part of the domain knowledge.

Given this situation, a new factor in supporting a configuration task can be achieved when deeper knowledge is employed through the use of functional models. This thesis examines the role of functional models in the field of configuration and contributes to this area in the following respects:

- *Framework of Configuration.* We investigate the question of what kind of models are used in configuration systems and secondly, we develop a classification scheme that covers a wide range of configuration problems. This classification scheme moves the type of description for the configuration objects into the center; it then provides for a more realistic view of the configuration problem's complexity.

  Grounded in this understanding of configuration, we develop a formal framework. This framework gives a precise methodology for studying the phenomenon of configuration from a viewpoint that is independent of any knowledge representation. Based on the models here, we can show that structure-based modeling and resource-based modeling are in some sense equivalent.

- *Configuration Based on Behavior.* Simplifying matters, configuration technology has been applied successfully to problems that can be characterized as follows. Either the components to be composed are described by relatively simple properties, or the structure of the system being configured is known in principle. However, at the present time, the configuration of a system that is not of a predefined structure and that relies on complex behavioral dependencies cannot be automated.—Nevertheless, we will go in this direction.

  We introduce the configuration of hydraulic systems as a problem that is founded on deep physical connections. The solution of hydraulic design problems requires experience in hydraulic model formulation as well as simulation know-how and, up until now, can be mastered by domain experts only.

  This work analyzes the hydraulic design procedure and shows how it can be supported. We develop efficient modeling and inference concepts to process hydraulic engineering expertise. Our modeling approach encloses a tailored behavior description language and allows an object-oriented definition of hydraulic components; the inference approach combines knowledge-based techniques with domain concepts and is able to cope with the problems of model selection, model synthesis, and behavior simulation.

- *Configuration Systems.* Aside from theoretical concepts we present working systems. This thesis introduces the configuration systems MOKON and *art-deco* that we have developed to master real-world configuration tasks and that operationalize large parts of our concepts.

  Both configuration systems exploit the functional meaning of the components involved in the configuration task. MOKON solves configuration problems that rely on property-based component descriptions. *art-deco* realizes, tailored to the hydraulic domain, the description and the processing of behavior and supports the configuration of hydraulic systems in a new quality.

The development of the *art-deco* system shows that the existing configuration technology can be employed in such a way that even design problems can be supported adequately.

Thesis Overview

The thesis is organized into two parts. Part one presents general and theoretical aspects of configuration; part two is comprised of the chapters that are concerned with the operationalization of functional models in configuration systems.

Part I Classification and Formal Framework

Chapter 1 briefly discusses the notions of configuration and configuration systems; it gives an introduction to knowledge-based systems and their development, as well as relating configuration to other knowledge-based problem classes.

Chapter 2 investigates criteria to evaluate configuration problems and systems. It illustrates different configuration scenarios and discusses typical characteristics of configuration and design problems. This chapter also introduces different configuration models and a classification scheme in these two dimensions: *description type* and *problem type*.

Depending on a problem's description type, basic configuration mechanisms exist; some concepts are outlined in the latter part of this chapter.

Chapter 3 introduces a formal framework for configuration. In particular, it formalizes relevant configuration models and presents an equivalence

result concerning two of these models. Also, a number of typical configuration problems is formulated.

Part II Operationalizing Configuration Tasks

Chapter 4 focuses on a particular function-based component model: the property-based component model. The chapter discusses the philosophy and general characteristics of property-based configuration, it presents a basic configuration algorithm and illustrates the role of metaknowledge. The last section gives a short description of MOKON, a configuration system that realizes the outlined concepts.

Chapter 5 focuses on a particular task, the configuration of hydraulic systems, and shows how such a demanding configuration problem is supported. It presents a generic component model that allows the formulation of hydraulic checking and parameterization problems and addresses the processing of this model.

Chapter 6 introduces the philosophy and the inference concepts of *ardeco*, a system that has been developed to support configuration in hydraulics. *ardeco* operationalizes the behavior-based component model and a large part of the concepts presented in the former chapter. Realizing a graphic problem specification and utilizing knowledge-based techniques for hydraulic systems analysis, *ardeco* simplifies the entire hydraulic design procedure.

Part I

Classification and Formal Framework

Chapter 1

An Introduction to Configuration and Knowledge-based Systems

This thesis deals with the configuration of technical systems. In order to become familiar with related terms and concepts, the purpose of this chapter is threefold:

1. The first section reviews the notions of configuration and configuration systems.
2. In section 1.2 we give an introduction to knowledge-based systems and discuss some aspects of their development: Configuration tasks are usually problems knowledge-based systems deal with.
3. The last section relates configuration to other knowledge-based problem classes. We then discuss if the development of knowledge-based systems can be supported by generic "problem solving methods".

1.1 A First Glance

Common definitions for configuration describe it as the process of composing a technical system from a predefined set of objects. The result of such a process is called configuration too and has to fulfill a set of constraints given. Aside from technical restrictions, a customer's demands constitute a large part of these constraints [61], [81].

This informal definition gives a declarative description of the job a configuration system does; it motivates the configuration process from its result. Figure 1.1 depicts this view: $Q_0, \ldots, Q_e$ denote different states of a system being configured, where Q_0, the starting point, is the empty set, while Q_i comprises the *qualities* of the system after configuration step i. Q_e denotes the qualities of the readily configured system. The transformation steps t_j stand for the operations performed by the configuration system within step j.

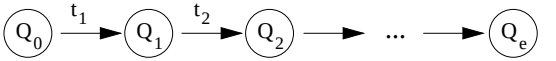


Figure 1.1: *Configuration as a sequence of states and transitions*

A configuration system can be seen as a program that takes a set of demands $D \subseteq Q_e$ as input and computes all information to describe *extensionally* the configured system. I.e., it generates information about the required objects, their type, number, topology etc. such that the emerging configuration fulfills all constraints (cf. figure 1.2).

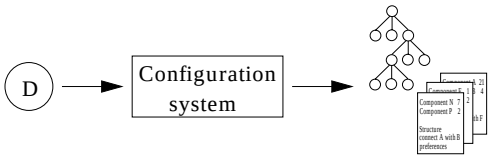


Figure 1.2: *The job a configuration system performs*

The definition above does not imply information about the complexity of configuration problems or how to get the hang of them. This should not be surprising, the development of a configuration system requires the analysis of the domain and the processing of related problems, more or less. As a consequence, we cannot provide a set of "general purpose configuration algorithms" nor build a generic configuration platform.

Nevertheless, in the past a large number of universal configuration approaches have been developed. This fact seems to contradict the preceding paragraph, but under the following interpretation it does not: Most of these approaches represent *abstract concepts* where domain knowledge

and domain-specific solutions have to be integrated. Examples are Chandrasekaran's GENERIC TASK approach [8], the high-level configuration language DSPL developed by Brown and Chandrasekaran [6], or the tool box with configuration modules described by Syska [72]. Also, the different attempts to operationalize the conceptual models of KADS point at the same direction [46], [33].

These approaches have the following in common: They model those parts of a configuration process that are more or less independent of a particular problem. Stated another way, it is intended to obtain *generic* problem solving steps in order to simulate an expert's way of handling design problems. As a result, these approaches provide modules that realize the resolution of contradictions, the test of constraints, or the selection of design operators and plans.

Especially for a complex design problem[1] is such an analysis necessary in order to get a grip on the problem at all. However, the following two points should be considered: (*i*) The power of a configuration system often comes up with the employment of domain knowledge, and (*ii*) a lot of "typical" configuration problems are not so demanding from a technical standpoint. Instead, the problems are of an organizational type and relate to the following questions [49], [81]:

- How can configuration knowledge be maintained?
- To what extent can configuration decisions be explained?
- Which mechanisms are appropriate to specify configuration problems?
- How can domain knowledge be described in a clear—so to speak, domain-specific way?

It should be investigated how configuration systems can be developed to provide support for these problems. There exist approaches aiming at this objective, e.g. the following:

Hein and Tank developed the configuration language AMOR, which shall bridge the knowledge engineering gap[2] [74]. However, it was not intended to restrict AMOR to a single domain and a small class of configuration problems.

[1] We will present a detailed classification of design problems in chapter 2.

[2] Roughly speaking, this term designates the distance between a conceptual configuration problem and a system tackling it. We will take up this term in section 1.2.

McDermott coined the notion of "role-limiting-methods" [55]. His configuration approaches distinguish explicitly between different roles (design) knowledge can play, dependent on its context of use. His team developed among others the system VT which realizes the configuration of elevators [51].

Discussion

Research in the field of configuration can be divided in the following manner: approaches that focus on an abstract process of configuration and approaches that aim at adequate mechanisms to specify configuration knowledge. Clearly, when tackling a real-world problem, none of these approaches could do a complete job. In the former the knowledge has to be integrated—just as with the latter it has to be processed. Employing domain knowledge within a configuration system can affect both fields of research. In this connection the utilization of functional models is one facet of how domain knowledge comes to effect.

The development of a configuration language that is both adequate in its knowledge representation mechanisms and independent of a domain or configuration task is nearly impossible. The objective of adequacy is useful only in connection with a particular problem, and therefore, it is more or less opposed to the objective of independence, which is often claimed too.

1.2 Knowledge-based Systems

In order to tackle a configuration problem, expertise needs to be operationalized. Thus, most of the configuration systems developed in the past were called *knowledge-based* configuration systems or *expert systems* of configuration. This subsection contains a brief introduction to knowledge-based systems and discusses their development. This information may be useful to clear up the scepticism that sometimes is associated with knowledge-based systems or related terms.

Highly sophisticated and complex computational methods can be found especially in engineering domains. However, a set of differential equations describing a physical system along with the numerical methods solving them should not be called a "knowledge-based system" or "expert system". Also the programming techniques that were used to realize a system,

e.g. rule-based or frame-based techniques, are not a useful criterion as to whether a program is an expert system.

Brown and Chandrasekaran define expert systems as programs where we can find some kind of computations that underly intelligent behavior and which therefore are discrete, symbolic, and qualitative [6]. They call this kind of computation *problem space exploratory techniques* and characterize them to be "intelligent" in the following sense:

> *"They explore a problem space, implicitly defined by a problem representation, using general search strategies which exploit typically qualitative heuristic knowledge about the problem domain."*
>
> Brown & Chandrasekaran, [6], p.4

In fact such computational methods make up an important part of expertise. In a nutshell, we will refer to a program as a knowledge-based system if it operationalizes the knowledge, the experiences, or the procedures of an expert in whole or in part.

Architecture of Knowledge-based Systems

The classical view on knowledge-based systems is the following:

knowledge-based system	=	domain-independent inference engine
	+	domain-specific knowledge base
	+	problem-specific database

I.e., an important idea of knowledge-based systems is the distinction between domain knowledge on the one hand and the methods that process this knowledge on the other: Inference mechanisms are applied to the knowledge in the knowledge base and are intended to produce a solution of the actual problem. Knowledge base and inference component may be completed by modules that guide the user, generate explanations, or realize the specification of new knowledge.

A point of criticism related to this view is the explicit separation of knowledge base and inference mechanisms. In fact, a large part of an expert's knowledge needs tailored algorithms determining *how* the knowledge is to be processed. Bylander and Chandrasekaran coined the term *interaction hypothesis* in this context [7].

Thus, a knowledge base will not contain merely facts and rules but also algorithms that realize both the processing and the specification of expertise. This understanding leads to a view of knowledge-based systems as shown in figure 1.3.

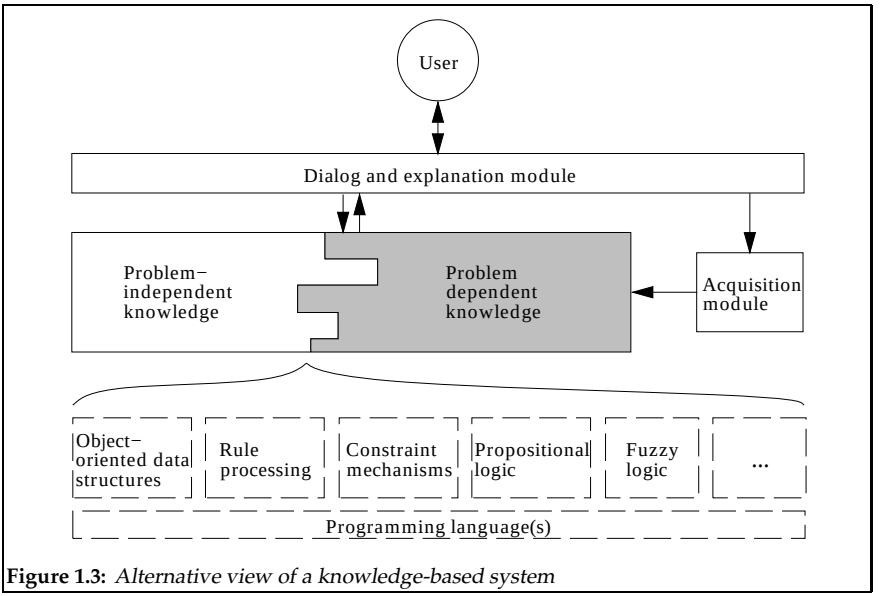


Figure 1.3: *Alternative view of a knowledge-based system*

Note that the knowledge base is divided into a problem-*in*dependent and a problem-dependent part. The former part models the never changing concepts and theories of a domain—example: the descriptions and the algorithms that operationalize Kirchhoff's or Ohm's law. The other part can be filled with knowledge less fundamental for the domain and the problem respectively, i.e., it stores definite situations or new dependencies. The boxes below the knowledge base designate some exemplary techniques that *can* be used to operationalize knowledge. In particular, there is no explicit inference module apart from the knowledge base.

In order to develop such a system, the mechanisms for knowledge specification and knowledge processing have to be oriented by the actual problem. Thus, from today's point of view, it is hardly possible nor very useful to develop universal, that is, domain-independent problem solving systems.

Development of Knowledge-based Systems

The literature on this subject provides many methodologies intended to guide the development of knowledge-based systems. Actually, the problems to be addressed with knowledge-based technology are not of a generic nature. Rather, their solution requires a deep understanding of the related domain as well as the development of new concepts and algorithms, which can never be part of a development methodology. Even so, we can roughly sketch out some ideas of the classical development approaches to disclose the complexity and the weak spots of the development process and knowledge-based systems.

Knowledge-based systems operationalize the knowledge and the experiences of one expert or a group of experts. In this connection chunks of knowledge have to be recorded, better: elicited, then structured, interpreted, and implemented. This is a demanding job that the knowledge engineer is supposed to do; he has to become an expert in the domain of interest.[3]

Figure 1.4 depicts the development process of knowledge-based systems as a sequence of three major stages.

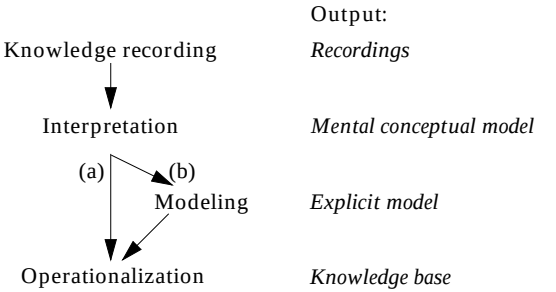


Figure 1.4: *Stages in the development process of knowledge-based systems*

The first stage of the development process, the knowledge recording stage, can partially be automated. There is a whole string of concepts, theories, and tools aiding the recording and structuring of expertise. These approaches can be divided into two classes: (*i*) approaches that merely support the process of knowledge recording, such as unstructured interviews

[3]Sometimes it goes the other way round: A domain expert becomes an expert system specialist.

and repertory grid techniques, and (*ii*) approaches that support parts of the interpretation process in so far as they use, build, or refine a conceptual model of the domain. For approaches of either class there exist tools as well as manual techniques. Examples for tools are CLASSIKA [21], MOLEKA [19], or SALT [50].[4]

The stage of interpretation is the most important and difficult one. As depicted in figure 1.4, two philosophies can be distinguished: the rapid-prototyping approach (a) and the model-based approach (b). Both approaches have in common the fact that the knowledge engineer develops "his" mental conceptual model[5] of the domain and the problem. The model-based procedure differs from the rapid-prototyping concept in that it tries to reveal the conceptual model.

The main advantages of a model-based procedure are as follows: (*i*) the conceptual model is closer to the expert's terminology, (*ii*) the conceptual model discloses the structure of the actual working model, and (*iii*) wrong interpretations of data or an incorrect conceptual model will not lead to a waste of implementational work. On the other hand, these models are not operational and cannot be verified in their dynamical behavior. I.e., there is no guarantee that they really work.

A well-known representative of the model-based approach is the KADS methodology[6] [82], [31]. KADS provides generic constructs that aid a knowledge engineer in modeling a problem's inference structure: the so-called metaclasses, knowledge sources, and models of interpretation.

Operationalizing a knowledge base via rapid-prototyping means to quickly obtain a running system that is based on some exemplary cases. This prototype is extended or rebuilt at a later stage in the development process. The advantages of this approach are the immediate verification of a system's functioning, and the direct flow back of first experiences. Its main disadvantages are thus: The representation of expertise is oriented by implementational terms and restrictions—not by the domain, and no explicit model of the inference process and of the types of knowledge is constructed.

Figure 1.5 relates the two philosophies to their level of abstraction. The distance between the knowledge analysis task on the left-hand side and the

[4] Karbach and Linster give an excellent survey of this field [31].

[5] Norman firstly introduced the term as a designation for the mental model of expertise [58].

[6] KADS stands for "Knowledge Acquisition and Documentation Structuring".

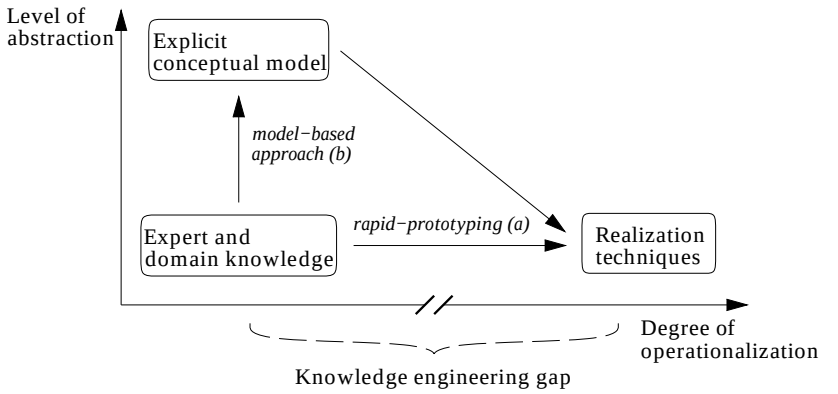


Figure 1.5: *Comparing the two approaches' level of abstraction*

shells, mechanisms, or knowledge bases on the right is sometimes called the *knowledge engineering gap*.

Clearly, model-based approaches like KADS clarify the inference process itself, but they provide only little aid in bridging the gap between knowledge analysis and implementation. The reason for this is that such "modeling-of-expertise"-approaches are placed on the knowledge acquisition side. In other words, they do not lead to a working system.

Because the developers of KADS and other groups of researchers knew about this lack, they began to develop operational equivalents to their theoretical concepts. Examples are ZDEST-2 [33], MODEL-K [80], OMOS [45], or Chandrasekaran's GENERIC TASKS [8]. These approaches should cut down the knowledge engineering gap from the other side—say, from the operational standpoint. Exactly this point is the key idea presented in the next section; the design process of knowledge-based systems would be reduced to a "simple" selection procedure, if we identified universal tasks—the so-called problem classes—for which appropriate algorithms have already been developed.

1.3 A Short Discussion of Problem Classes

Problem classes provide a scheme for classifying the problems of knowledge-based systems. The idea of distinguishing between different problem classes implies the following wishful thinking: When given a particular problem Π, based on a "problem-class-look-up-table" that associates problems with methods solving them, a solution for Π should be easily constructed. We want to state at least some proposition regarding Π's complexity.

A classification of problems might support developers in tasks such as method selection, tool selection, knowledge acquisition, and knowledge implementation. Steels claims as a long-term objective:

> *"At some point, we should end up with a knowledge engineering handbook, similar to handbooks for other engineering fields that relates task features with expert system solutions."*
>
> Steels, [66], p.48

Regardless of whether Steels's idea can be realized completely, it is useful having a look at problem classification schemes for knowledge-based systems. Possible criteria for a classification are (*i*) the type of a problem, i.e., certain characteristics specifying the task to be performed, (*ii*) the methods employed to solve a problem—the so-called *problem solving methods*, or (*iii*) the type of the model realized within a system.

Classifying Problems by their Type

A classification of problems that is based on the problem's type was first introduced by Hayes-Roth et al. [26]. In particular, they found the following classes: interpretation, design, plan, monitor, debug, repair, tutor, and control. Clancey took these classes and developed the hierarchy as depicted in figure 1.6 according to the following idea:

> *"We group operations in terms of those that construct a system and those that interpret a system, corresponding to what is generally called synthesis and analysis."*
>
> Clancey, [11], p.315

By synthesizing a system we mean its *composition*, or at least, its *modification*. Therefore, all types of configuration problems are basically of

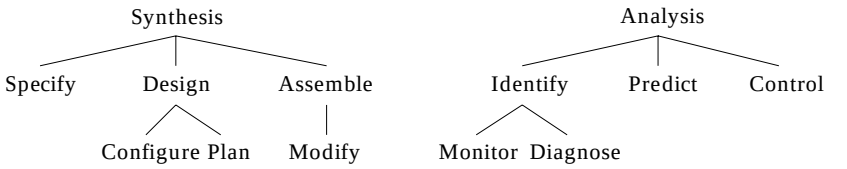


Figure 1.6: *Hierarchy of problem classes according to Clancey*

synthetical nature. Configuration tasks and planning tasks are the most important representatives within the class of synthetical problems. There is still the discussion whether configuration problems can be subsumed under planning problems or vice versa. Simplifying matters, planning is to be considered as the composition of atomic *actions*, whereas a configuration process deals with physical objects.

In contrast to the above, an analytical task requires the *presence* of a system [62]. This system is to be classified and investigated respectively but never modified in the course of an analytical process. The most important representative within the class of analytical problems is diagnosis.

Classifying Problems by Problem Solving Methods

We use the term *problem solving method* for a procedure, a concept, or an algorithm that is employed in knowledge-based systems in order to tackle a clearly defined problem. Such methods can form a lattice of problem classes where the rationale is as follows. All problems subsumed under the same class can be solved by the same method. Well-known examples for problem solving methods are GENERATE-AND-TEST, COVER-AND-DIFFERENTIATE, HEURISTIC-CLASSIFICATION, ESTABLISH-AND-REFINE, or SKELETAL-PLANNING [19].

Bauer et al. distinguish between three types of problem solving methods [2]: (*i*) methods that specify a conceptual procedure for knowledge processing like HEURISTIC-CLASSIFICATION or HYPOTHESIZE-AND-TEST, (*ii*) role-limiting-methods that just need to be "filled" with domain knowledge like COVER-AND-DIFFERENTIATE, and (*iii*) methods that realize basic knowledge processing techniques like Fuzzy inference.

Additionally, a distinction between "strong" and "weak" can be made. These terms were firstly introduced by McDermott [55] and can be found

in Puppe too [61]. The attributes "strong" or "weak" do not characterize a method's efficiency; rather, they indicate its range of application. A weak problem solving method is less specialized and so, it can be used for a wide range of knowledge representation and knowledge processing tasks. As a consequence it scarcely provides support for *problem-dependent* knowledge acquisition and knowledge processing. Examples for weak problem solving methods are the generic search strategies BREADTH-FIRST-SEARCH and BEST-FIRST-SEARCH.

Strong problem solving methods provide tailored concepts for problem-dependent tasks. Due to their specialization, their range of application is rather narrow. On the other hand, they are able to support the knowledge acquisition process to a large extent. COVER-AND-DIFFERENTIATE and PROPOSE-AND-REVISE are examples for strong problem solving methods since they prescribe the structure of the required knowledge.

Until now a generally accepted lattice of problem classes that is oriented with problem solving methods does not exist. Approaches pointing in this direction are the KADS models of interpretation, the mapping model as proposed from Bauer et al., Puppe's identification of problem solving types, or Karbach et al.'s descriptions of problem solving methods, which are placed at an abstract level [32].

Classifying Problems by the Model Employed

Within the classification approaches above problem classes are identified by applicational aspects. This would be useful, if we could expect problems of similar complexity and of similar structure under the same label as e.g. "configuration" . But, there is no compelling evidence why a diagnosis problem Π_{D_1} should be more similar to a diagnosis problem Π_{D_2} instead of to a configuration problem Π_C. Note that similarities can also be found with regard to the type of model operationalized in a knowledge-based system.

Depending on the knowledge that is used in a model, different processing mechanisms have to be employed. So it is likely to comprise problems with respect to the required inference mechanism: problems that can be tackled by a qualitative approach, problems that need some kind of Fuzzy inference, problems that rely on precise mathematical models, or problems that need to be processed at a subsymbolic level.

These classes are oriented by the realization of a model. In contrast

to the formerly introduced classification approaches, which represent the user's point of view, the approach proposed here is intended to support the developers of knowledge-based systems.

Discussion

Developing a classification scheme of problems means both the identification of classes that comprise problems of a certain type and the association of these classes with problem solving methods. The development of such a classification scheme is useful for the following reasons:

1. *Communication.* Sharing a similar terminology will integrate users into the development process and help in avoiding misunderstandings.
2. *Simplification of the Design Process.* The development of knowledge-based systems should not have to start from scratch every time. Here, a classification scheme could support the selection, adaptation, and combination of problem solving methods in order to build knowledge-based systems tailored to a new task.

The realization of this second aspect will be difficult, if we develop a problem classification scheme from an applicational standpoint, as proposed by Clancey or by Hayes-Roth. We should instead follow the ideas outlined in the latter two subsections, which rely on an algorithmic point of view.

To this day predefined toolboxes and problem solving methods have not led to a significant simplification of the design process. However, this point may be a consequence related to the complexity of the tasks of knowledge-based systems—not of the "underdeveloped" expert system technology. I.e., there is only little hope that the knowledge engineering gap is closed by someone else other than the knowledge engineer himself, or, in Twine's words:

> *"Knowledge engineering, as reflected by current practice within the expert system community, is more an artistic discipline than an engineering one."*
>
> Twine, [78]

Chapter 2

Classifying Configuration Problems

What type of criteria is useful in evaluating configuration problems and systems?

This chapter contributes some ideas and an informal classification scheme to this question. It is organized as follows.

Section 2.1 illustrates different configuration scenarios. It is intended to give an idea of the type, complexity, and related problems of important configuration problems. Section 2.2 elaborates on configuration and design problems and shows how they are related to each other. Section 2.3 develops our view of models for configuration. Further on, a lattice of different configuration problems is given in respect to two dimensions: *description type* and *problem type*. Depending on the description type of a problem, basic configuration mechanisms exist; some mechanisms are introduced in section 2.4.

2.1 Configuration Scenarios

Three different configuration scenarios are outlined. Particular solutions are not presented but an idea of the complexity of typical configuration jobs and of related problems is conveyed. Thus, this section provides an applicational background for the theoretical considerations of later sections and chapters.

Configuration jobs play different roles at different stages within the operational procedure of a company. These stages can be characterized by their temporal position and by the departments concerned. Figure 2.1 depicts the typical steps of the operational procedure in a manufacturing company.

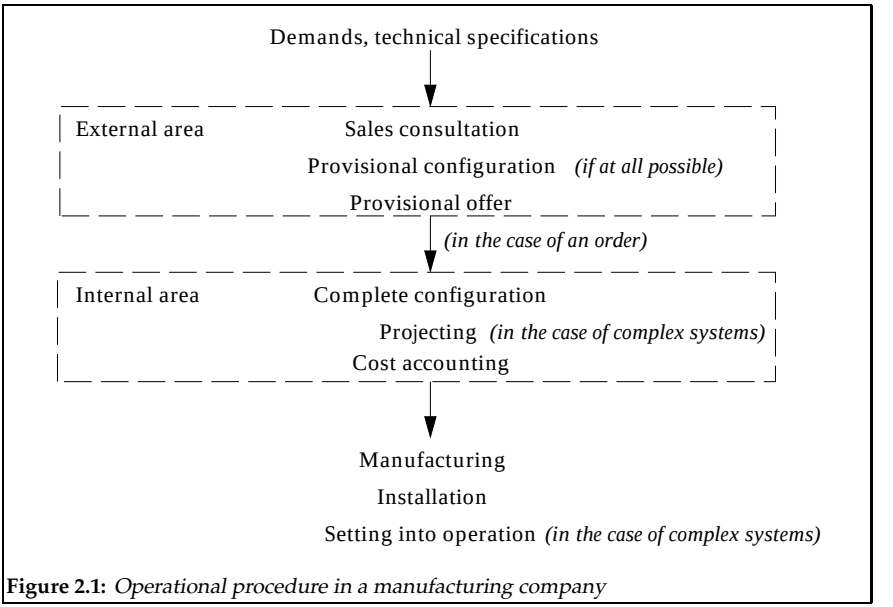


Figure 2.1: *Operational procedure in a manufacturing company*

Within this procedure two major areas where configuration tasks come up are distinguished: (*i*) The internal area, where typical inside jobs are placed, and (*ii*) the sales-oriented external area at the customers' site. Stages below the internal area are not comprised of typical configuration tasks, but configuration results gathered at earlier stages can play an important role here also.

The three examples presented now are arranged with respect to their operational field. Example 1 describes a scenario that is placed at some early stage of the operational procedure, while example 2 and, especially, example 3 affect stages of the internal area.

Example 1

Configuration of Telecommunication Systems. This scenario occurred at a known manufacturer of telecommunication systems [81].

The configuration of telecommunication systems is grounded on technical know-how since the right boxes, plug-in cards, cable adapters, etc. have to be selected and put together according to spatial and other constraints. Typically, there exists a lot of alternative systems that fulfill a customer's demands from which—with respect to some objective—the *optimum* has to be selected. Moreover, since the sales personnel should be able to provide a provisional offer, knowledge about both technical dependencies and actual prices must be available at the *customer's* site. Given an order, a whole string of working documents for the subsequent "pass through" has to be generated such as information for logistic and business purposes, or plans for scheduling, manufacturing, and assembly. Since technical progress goes on, new components will be developed that must be considered within the future process of configuration.

From a configurational point of view, the components of a telecommunication system can be divided into three major classes. One class contains the mandatory technical components for the switchboards: boxes, racks, base circuit boards, electric fans, power supplies, and main distributors. The second class consists of a wide range of plug-in cards and related cable adapters whose selection depends on the actual demands of a customer. The third class comprises components that embody services and extensions that a customer can choose: telephone connections of analog or digital type, fax machines, serial and parallel computer interfaces, dialing features, etc. Figure 2.2 illustrates the dependencies.

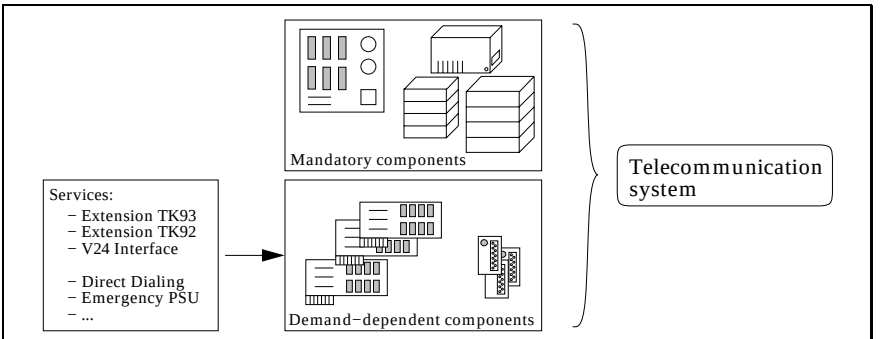


Figure 2.2: *Technical dependencies of telecommunication systems*

Related Problems. Aside from the solution of the technical configuration problem, the following demands are related to a configuration system: (*i*) An optimum configuration must be computed in acceptable time at the customer's site, (*ii*) the configuration system should not be too demanding with respect to hardware resources, and (*iii*) the settling-in period for both the sales personnel and the personnel maintaining the system should be at a minimum.

Example 2

Configuration of Mixing Machines. The design of mixing machines is a complex configuration problem for which different configuration systems have been developed in the past. In this subsection a design problem of a manufacturer described by Brinkop and Laudwein is sketched out [4].

The configuration process depends on a customer's mixing task, i.e. the basic mixing operation, the viscosities and the specific gravities of the products involved, and the dimensions of the tank where the mixing process will take place. The basic structure of a mixing machine is depicted in figure 2.3.

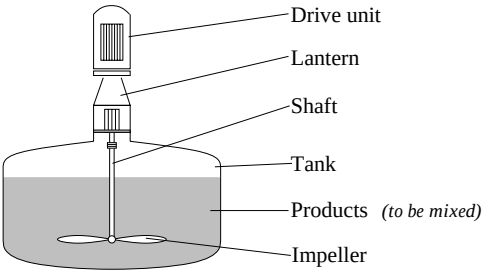


Figure 2.3: *Basic structure of a mixing machine*

There are only a few conceptual elements that make up a mixing machine, but the number of possible variants is extremely high. This results from the large selection in which each basic building block occurs. E.g., there are about 15 different types of impellers, which vary from $15mm$ to $1500mm$ in their diameter. The length of the shaft is variable from $1m$ to $15m$ while its possible diameters vary from $25mm$ up to $125mm$. The drive of a mixing machine is selected from a set of several thousand motors and gear units, and, depending on the mixing task, additional components like

bearings or sealings can be employed. During the configuration process, a lot of physical dependencies have to be processed, and, in addition, industrial norms and quality standards need to be met. Beyond such a technical specification, the output of the configuration process should comprise a reliable offer and a drawing true to scale.

Related Problems. Aside from the solution of the technical configuration problem, the following demands on a configuration system were stated from the manufacturer's site: (*i*) The configuration process should become simpler, (*ii*) design knowledge for acquisition and maintenance purposes should be represented explicitly, and (*iii*) the settling-in period for the sales personnel should be at a minimum.

Example 3

Designing Hydraulic Systems. A hydraulic system consists of mechanical, hydraulic, and increasingly more electronic components. Configuring a hydraulic system is a complex process that starts with design concepts and ends with setting the installed system into operation.

The starting point for this configuration process is a task that a customer wants to be performed by hydraulics. It can be a lifting problem, the actuation of a press, or some complex manipulation job. Usually, the resulting demands on a hydraulic system are specified and illustrated by means of different diagrams. These diagrams indicate the course of forces at the cylinders, the switching positions at the valves, etc. Given this information an engineer defines a circuit's topology, selects components like pumps, valves, pipes, computes their parameters, and checks, among other things, the stationary and dynamic behavior of the system. Figure 2.4 shows switching diagrams and a hydraulic circuit.

During the different checking stages one has to investigate if all geometrical connections fit, if the switching logic realizes the desired behavior, which maximum pressure values occur, etc.

Configuring hydraulic systems is not a typical configuration problem commonly described in the literature on this subject. Although we deal with a finite set of objects to be selected, there are two aspects where this problem differs from the "representative configuration problem": (*i*) The structure of a hydraulic circuit is not of a fixed type. Thus, the search space of possible solutions is of higher order as compared to the two examples presented before. Also, the identification of possible solutions requires

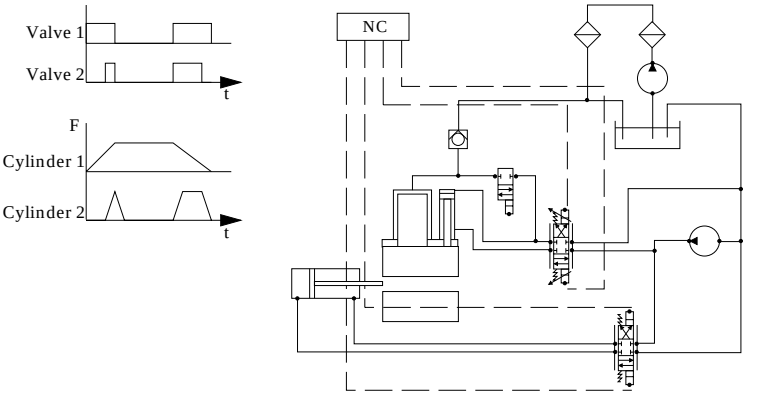


Figure 2.4: *Switching diagrams and a plan of a hydraulic circuit*

complex and difficult computations. (*ii*) Checking the function of a hydraulic system means to check its *behavior*, i.e., some kind of simulation has to be performed[1].

Remarks. Chapter 5 elaborates on the configuration of hydraulic systems. There we discuss to which extent this task can be processed automatically and in which way an engineer can be supported appropriately.

Discussion

The examples above introduced typical problems one is faced with when configuring or, as the case may be, projecting technical systems. The job the sales personnel and the project engineers do can be seen as an informal definition of the term "configuration". A particular configuration task and, as a consequence, a configuration system for its support depend on the domain, organizational aspects, the level of automation the configuration system is intended to provide, and the qualification of the users. From a technical point of view, a configuration system should give answers to the following questions:

- Does a configuration exist that fulfills the customer's demands?
- Does a system, configured manually, work correctly?

[1] In the previous example, during the configuration process of mixing machines, a certain kind of simulation has to be performed, too.

- Of which components is the configuration composed?
- How much does a configured system cost?

In addition, concepts for maintenance and acquisition of new configuration knowledge make up an important part within a configuration scenario. A user should be able to specify (product) knowledge in an adequate way, that is to say domain- and problem-oriented, but not at the programming language level. Clearly, this kind of problem cannot be solved independently of an actual context as well.

The task of designing hydraulic systems shows that a sharp line cannot be drawn between configuration problems on the one hand and design problems on the other. Configuration is generally viewed as a process of selecting and connecting objects while paying attention to some constraints. Especially in the last example, this process turns out to be a sophisticated layout problem that needs an engineer's creativity.

2.2 Relating Configuration to Design

The previous section showed a wide range of configuration and design problems but gave no answers as to how such problems are linked:

- Where does configuration end and design begin?
- How can configuration problems be characterized?

In this section we discuss existing views and present new aspects to answer these questions.

On Design

What is design? We will not investigate this question from its philosophic or sociologic roots here. Rather, we focus on the teleological aspect of design: the creation of new artifacts. And, with the question as to how new artifacts are created, one raises the next questions, that is, if and how this creation process can be supported.

A lot of researchers work in the field of knowledge-based design support—e.g. Chandrasekaran and Gero. They define design as follows.

> *"A design problem is specified by (i) a set of functions to be delivered by an artifact and a set of constraints to be satisfied and (ii) a technology, that is, a repertoire of components assumed to be available and a vocabulary of relations between the components."*
>
> Chandrasekaran, [9], p.60

> *"The metagoal of design is to transform requirements, generally termed function, which embody the expectations of the purpose of the resulting artifact, into design descriptions."*
>
> Gero, [22], p.28

The result of a design process is a definition of the artifact searched for, which can be transposed easily—better: definitely—into a real system. Such a definition might be a graphic or a textual representation of the artifact's exteriors, an algorithmic specification, differential and algebraic equations, topological information, or relationships between building blocks.[2] Subsequently, we outline an abstract model of the design process that leans on Gero's ideas [22].

The purpose of a design process is the transformation of a complex set of functionalities D (= demands) into a design description C (= configuration, composition):

$$D \longrightarrow C$$

"$\longrightarrow$" stands for some transformation, C is considered the artifact's entire set of components and their relations. The transformation must guarantee that the artifact being described is capable of generating the set D of demands. Sometimes it is convenient to regard D as the union set of D_e and D_i, with D_e denoting a set of all explicitly desired demands, and D_i comprising the constraints implicitly established by the domain. Due to the complexity and the diversity of a design process, no universal *theory of design* can be stated, i.e., in the very most cases no direct mapping is given between the elements $d \in D$ and the objects $o \in C$.[3]

Working on a design problem can be compared to a process of *balancing* two sets of behavior: the set of intended or expected behavior B_e, and the

[2] We go a step further than Gero here: Gero distinguishes between a design description Γ, which stands for some kind of informal drawing, and a configuration C, which represents the artifact's elements with their relationships. Since the transformation $C \longrightarrow \Gamma$ is canonical we take C and Γ to be equal and shall refer to C only.

[3] A special case of a direct mapping between $d \in D$ and $o \in C$ is the so-called "catalog look up".

set of observed behavior B_C. B_e can directly be derived from a designer's understanding for D, whereas B_C is the result of an analytical investigation of C that may enclose complex model formulation and simulation tasks:

$$D \longrightarrow B_e\ ,\ \ C \longrightarrow B_C$$

What happens during the balance process? The expected behavior B_e controls a synthesis step, that is, the definition of the configuration C, which in turn causes the behavior B_C. Within the next step, the so-called *evaluation* phase of the design process, the two behavior sets are compared to each other. This comparison produces new information that serves as input for a new synthesis step. Figure 2.5 illustrates the dependencies.

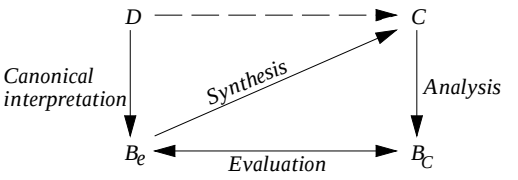


Figure 2.5: *An abstract model of design*

Configuration is Abstracted Design

The model presented in the former subsection is of a universal type, and therefore, it cannot be used to support a design process in operational terms. Consequently, this description of design should be constrained in some way. Before we describe different classes of design, we will illustrate some ideas of how configuration problems and design problems are linked to each other. Summarizing this subsection, we understand configuration as an *abstracted design process*.

The undoubtly most creative design job is the invention of an artifact that utilizes a physical law of nature in an unprecedented way. The invention of the electrical transistor or of the capacitor are well known examples of such achievements. To stay within the electronic domain, let us consider in a further course that an electronic system or device is to be designed where, among others, transistors and capacitors are to be used. This electronic device may be a component of a computer, for example a particular plug-in card or a power supply unit. Such a design job is very difficult and needs

an engineer's creativity, experience, and skill. Now, within a third step, we assume that a computer is to be designed (or configured respectively) using components like the above: a main board, plug-in cards, a power supply unit, etc. Obviously, this is the least demanding step within the entire procedure—"only" selection, positioning, and perhaps, parameterization tasks have to be performed.

Let us take a closer look at the steps 2 and 3. Step one does not play an important role here since there seems to be no chance of automating such a creative process in the foreseeable future. Within the design job of step 2, a particular kind of building block is given where items have to be selected, parameterized, and connected (cf. figure 2.6). The goal is to realize a new complex system of the desired functionality.

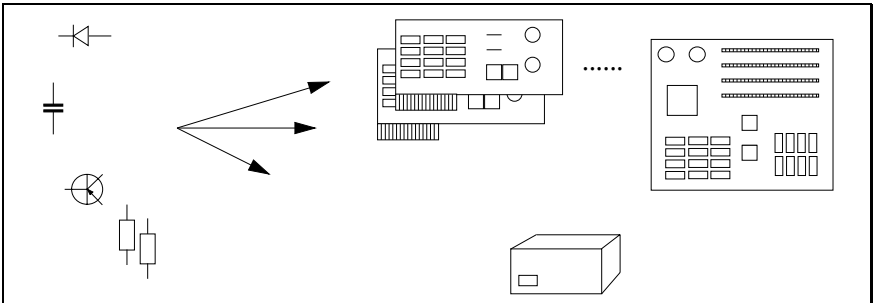


Figure 2.6: *Step 2 in the "design hierarchy"*

Step 2 can be characterized as follows:

- *Low-level Interfaces.* Basic technical components are used to realize a new system. The interfaces where these components can be connected provide only "low-level services". I.e., on this level one argues about voltages, voltage drops, oscillating frequencies, etc.
- *Processing Behavior.* In order to identify or to predict the behavior of such a system, the components' physical behavior has to be processed. In practice this job turns out to be sophisticated since the necessary model formulation and model simulation processes are of complex nature.
- *Weak Problem Specification.* The specification of the design problem refers to an object—in our case: the circuit—which does not exist

at the beginning of the design process. Merely the intention of the functions to be produced by the designed system is given.

- *Weak Theory of Design.* For large parts of the design problem there exists no or, as the case may be, only weak theory of design. There is a lack of procedural knowledge that describes how problem specifications can be mapped onto design decisions. This is a consequence of the design process's complexity.
- *No Structural Information.* There is no generic structure information of the system that could guide the design process. Moreover, since certain functions emerge automatically with the structure, as well as certain structures result from particular functions, a designer has to take care of structure and function simultaneously.
- *Designers are Experts.* The design process described here is carried out by experts. This has to be considered by a knowledge-based system that shall support this process.

When working on a design problem within the third step, we deal with complex building blocks for which the design process is *already completed* (cf. figure 2.7). The goal here is to configure a system that fulfills the particular demands of a customer.

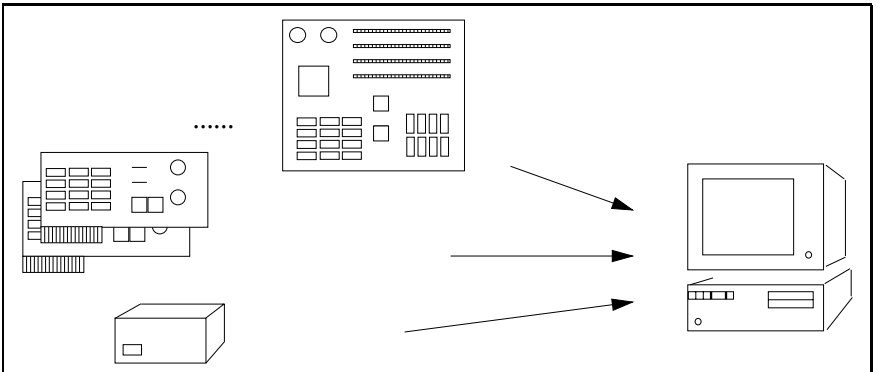


Figure 2.7: *Step 3 in the "design hierarchy"*

Step 3 can be characterized as follows:

- *High-level Interfaces.* The components involved are complex technical systems themselves and provide "high-level functionalities" where

they interface. We could designate them as components where the design knowledge is *compiled in*. In our example they may provide a graphics adapter, a memory extension, etc. The components' interfaces in turn are of such a high-level type as well. Thus, the components can be connected easily since they make the complex underlying physical connections *transparent*.

- *Processing Functionalities.* In contrast to step 2, it is not necessary to simulate a composed system in order to predict its behavior. Rather, there is a checking and synthesizing process on the level of *abstract functionalities*. Nevertheless, checking and synthesizing a set of functions in order to meet a set of demands can turn out to be a sophisticated constraint satisfaction problem [6].
- *Weak Problem Specification.* A customer's demands refer to a system that doesn't exist yet. Thus, we have an implicit problem specification, just like within the former design step.
- *Weak Theory of Configuration.* There is no generic theory at hand that associates a problem specification with configuration decisions. This is not a consequence of the configuration's process complexity but rather of its strong domain dependence. Note that only very few configuration problems can be tackled via catalog look up.
- *Structural Information.* A lot of systems to be configured are structured in some way: The complete system is composed out of subsystems always in a similar manner [4], [24]. This information about the system's structure can be used to guide the configuration process.
- *Configuration Sequence.* During the configuration process of a complex system, it might be that a particular assembling sequence has to be obeyed. In such a case a designer is not only interested in the readily configured system, but also in a *plan* that determines a sequence of assembly steps for the components involved.
- *Configuration Personnel.* Although a lot of configuration jobs are of rather complex nature, they should be carried out by persons who are not technical experts in the domain: Many configuration jobs are sales jobs. This perspective gives rise to additional requirements that have to be considered when developing a configuration system.

A design problem within step 3 is much less complex than a design problem within step 2 because of the substantial reduction of possible

decisions in the designer's scope. This results in an enormous cut down of the potential search space.

The following points play a central role with regard to the design problem's search space or complexity:

1. All possible variants of the artifact being designed are anticipated in some way.
2. All possible variants of the artifact share the same structure.
3. The components used within the design process are of extremely specialized nature. Stated another way, components may only be combined in a very restricted way.
4. The artifact's maximum number of components is bound by a small number.

In the computer configuration example, all of the four points apply—but this need not be the case for any "step-3-design problem". It seems to be obvious that we get a qualitative reduction of a design process's complexity, if only one of the four conditions is fulfilled. This leads to the question, which complexity classes of design should be distinguished at all. The next subsection discusses a broadly accepted view.

State Spaces of Design

According to Brown, Chandrasekaran, Gero, and Tong it is useful to distinguish three classes of design [5], [6], [22], [76]. This classification, at first mentioned by Brown [5], suggests that a design process is for the most part a decomposition problem or a plan synthesis problem. I.e., his classification scheme is largely based on the difficulty of subtasks related to decomposition and plan synthesis. Class 1 encloses creative tasks, while class 2, as well as class 3, constitutes a well-defined state space of problems [22].

Subsequently, a short characterization of the classes depicted in figure 2.8 is given.

Class One $\simeq$ *Creative Design.* This class can be viewed as open-ended. Design problems of this type are outside the state space of innovate design. When solving such a problem, a completely new artifact is created. All kinds of inventions belong to this class. Even if the goals to be achieved are well-defined, there is no, or only a rough idea of *how* these goals can be

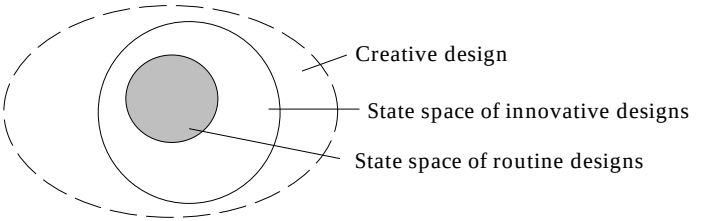


Figure 2.8: *State spaces of design by analogy with Gero*

achieved. I.e., there is no storehouse of solutions from similar problems, and, there is no knowledge of how a problem might be decomposed into less complex ones.

What makes a creative design problem impossible to be supported with knowledge-based systems is the fact that some or even all parameters forming the state space are unknown. The design process is influenced by the designer's perception, experience, and intuitivity.

Class Two ≃ Innovative Design. This class is intended to comprise problems for which powerful decomposition knowledge exists, but design plans for some of the component problems may need a substantial modification [6]. I.e., an artifact's structure is known for the most part. The complexity of such a design problem results from some unknown ranges of values for parameters. Gero argues in this context that both additional adaptation processes and the use of dependency knowledge are required to assist this job.

In [6], the design of a new automobile is given as an example for a typical class-2-design task. It is reasoned that this design problem does not involve new discoveries about decomposition, but on the other hand, "routine design methods" are unable to handle the major technological changes some components undergo.

Class Three ≃ Routine Design. This class is comprised of design problems for which the knowledge about decomposition and synthesis is completely known. The state space of all possible solutions is much smaller as compared to the state space of innovative design. What makes problems of this class tractable is that the structure of the system being designed as well as all parameters' ranges of values are known. Furthermore, there is sufficient knowledge of how values for parameters can be determined, knowledge about constraints that cuts down the search space, and, "repair knowledge"

to deal with incorrectly designed systems—

> *"In spite of all this simplicity, the design task itself is not trivial, as plan selection is necessary and complex backtracking can still take place. The design task is still too complex for simple algorithmic solutions or table look up."*
>
> Brown & Chandrasekaran, [6], p.34

Discussion

A general shortcoming of such classification schemes is the neglect of organizational aspects of design problems like knowledge acquisition, user interaction, or maintenance of design knowledge. Note that these points may constitute a larger part of the entire design problem as the actual "technical" problem does.

Weak spots of this classification from a technical point of view are as follows. The distinction between class-2-design and class-3-design shows no *qualitative* difference, and Gero or Brown and Chandrasekaran characterize the associated classes only vaguely.

Of course, the classification of Brown and Chandrasekaran is not meant to be formal or rigorous. It will probably never be possible to develop a precise classification scheme for design problems. Consider the design of a new electronic circuit: Knowledge about structure may be poor ($\simeq$ class 1), a lot of parameters are unknown ($\simeq$ class 1), but, as the case may be, powerful decomposition knowledge exists ($\simeq$ class 2), or, there is yet sufficient knowledge of how values for parameters can be determined ($\simeq$ class 3). As an aside, the design of a new automobile comprises a lot of highly sophisticated and creative subtasks like car-body styling, utilization of new materials, or electronic motor management and should be instead associated with class 1.

Such design problems could be classified more precisely, if the lattice of problems is further refined and/or oriented by additional domain characteristics.

From our point of view, it is useful and sufficient to differentiate between synthesis problems as follows: (*i*) *design*-problems that contain creative parts and (*ii*) *configuration*-problems that fulfill at least one of the four conditions pointed out on page 27:

1. all variants are anticipated

2. all variants share the same structure
3. components to be configured are specialized
4. components to be configured are of a small number

This specification describes problems similar to those in Brown's design class 3, but it is more concrete. Additionally, the conditions enumerated here define a qualitative difference between class-2-problems and class-3-problems.

2.3 A Classification Scheme

This section states a generic classification scheme for configuration and design problems. The scheme has a *description type* dimension and a *problem type* dimension. The former dimension designates particular domain models, which are also called *component models* here. The latter one distinguishes between different configuration tasks. Since organizational restrictions cannot be evaluated independently from a particular task, this classification scheme is intended to focus mainly on the technical part of a configuration problem.

Models in Configuration

A model is an intentional abstraction of a reality that focuses on definite aspects and consequently, reduces the reality's complexity. A configuration model is an abstraction of a configuration problem and must actually be employed to tackle the task. In order to evaluate a configuration problem's complexity it is necessary to investigate this model. Conversely, the realized configuration model represents a proper measure of a configuration system's power.

When describing models for configuration or design, there is the tradition to focus on *procedural* aspects. Examples for such a view are Chandrasekaran's "propose-critique-modify" methods [9], [6], or Puppe's problem solving methods of construction[4] [61], depicted in figure 2.9.

This process-oriented view is useful to describe how a configuration strategy is realized, how particular configuration steps are generated, or if

[4]The term "construction" can be seen as a synonym for the term "configuration".

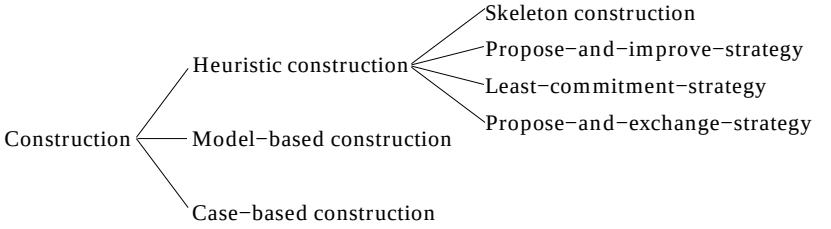


Figure 2.9: *Problem solving methods for construction according to Puppe*

sequential aspects of a configuration—better: planning problem—are the matter of subject. But, this view can be misleading since it neglects the concepts and the constraints of the domain or the task. Therefore, we will introduce the view of component models in configuration. This approach moves the description of configuration objects into the center but not the procedural strategies of the configuration method.

Remarks. The component model view of configuration assumes that there *are* still manifestable objects to be selected, composed, or parameterized. Henceforth, we exclude all those problems where no set of objects is given such as the invention of new artifacts or scheduling problems.[5]

Dimension 1: Component Models

Loosely speaking, the term "component model" designates the knowledge that is used to describe a system's building blocks. This knowledge should depend solely on the configuration problem, i.e., it ought to be both necessary and sufficient to carry out the configuration process. E.g., when configuring a telecommunication system as described in section 2.1, the component model should not contain knowledge about fundamental electronic dependencies.

For the following reasons we will use component models to describe configuration problems:

1. The complexity, that is, the degree of precision of the components' description determines decisively the complexity of the configuration process.

[5]Sometimes, scheduling problems are count as configuration problems.

2. Configuration problems are to a large part domain-dependent. This domain dependence is reflected rather by the type of the component model than by the configuration method.

Thus, a classification of configuration problems that is oriented by component models will not only be closer to the particularities of a domain but also provide a more realistic view of a configuration problem's complexity.

Component models can be classified by the type of description upon which they are based. We distinguish between two major types here: *structure-based* and *function-based* descriptions (dependent on the type of a description, a component model will also be called structure-based and function-based respectively). Note that a component model must not be of a pure type but can rely on a combination of both structural and functional connections.

Structure-based descriptions define relations on particular subsets of the entire object set. A single configuration object is considered to be an abstract atomic entity of boolean domain, which has either the status "selected" or "not-selected". Function-based descriptions model the *properties* or the *behavior* of a single component.

There is a wide difference between these description types:

From the standpoint of knowledge representation, structure-based descriptions define a *global* view on the system to be configured whereas function-based descriptions rely upon *local* connections only. From the standpoint of knowledge processing, structure-based descriptions form an *explicit* definition of the configuration process whereas function-based descriptions constrain *implicitly* the configuration process. Section 4.1 discusses the differences in connection with procedural aspects of configuration.

We identified different component models, from the structure-based as well as from the function-based type, that play a role within configuration and design problems. The interesting models, comprised in figure 2.10, are briefly characterized now.

Descriptions of structure-based component models:

- *Associative Description.* Associative descriptions define the system to be configured by relations that contain no explicit compositional information and no causal dependencies. When given a set of objects $o_1, \ldots, o_n$, the relations between these objects can be expressed by

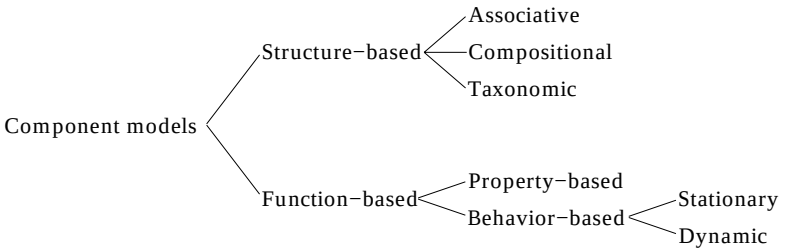


Figure 2.10: *Component models in configuration*

rules in a propositional form[6]. The objects o_i denote the set of boolean variables where "$\wedge$", "$\vee$", "$\neg$", "$\rightarrow$" are the propositional connectives allowed.

Example: $o_i \wedge o_j \rightarrow o_k \vee \neg o_l$

The semantics of this rule is as follows. If the configuration *contains* object o_i and object o_j, then the configuration must also *contain* either object o_k or not object o_l. In practice, a boolean variable is not restricted to denote exactly one object but can stand for a set of objects as well. Associative descriptions have compositional and functional information *compiled in*. They can be viewed as a very compact form of *heuristic configuration knowledge* identified by human experts dealing with the configuration problem over a long period.

- *Compositional Description.* Descriptions of this type specify a compositional view of the system to be configured. Sometimes, this view is called decompositional hierarchy or Gozintho graph. When given a set of objects $o_1, \ldots, o_n$, the decompositional relations can be expressed by rules of the following form:

  $\langle \text{rule} \rangle \longrightarrow \langle \text{object} \rangle \rightarrow \langle \text{object} \rangle \mid \langle \text{objects} \rangle$
  $\langle \text{objects} \rangle \longrightarrow \langle \text{object} \rangle \wedge \langle \text{object} \rangle \mid \langle \text{object} \rangle \wedge \langle \text{objects} \rangle$
  $\langle \text{object} \rangle \longrightarrow o_1 \mid \ldots \mid o_n$

  Example: $o_i \rightarrow o_j \wedge o_k \wedge o_l$

  The semantics of this rule is as follows. Object o_i is *composed* of three objects, namely o_j, o_k, and o_l. These objects in turn may be described

[6]The rule formalism chosen here and in the following descriptions is only one possibility to define structural dependencies. It has nothing to do with the representation of structural knowledge in a concrete system.

by other rules. Objects that can be decomposed are called assemblies; those objects which cannot, at the bottom of the hierarchy, are called components.
One could argue that these dependencies could be defined by an associative description as well. This is correct from a syntactical point of view. What makes compositional knowledge so powerful are operational aspects. A configuration algorithm will rely on the decompositional semantics of the rules that define a system's structure here.

Compositional descriptions are typical for a lot of configuration problems. Although only few problems can be tackled exclusively by this kind of knowledge, it should be employed whenever possible; compositional relationships decisively cut down the search space since they guide the process of configuration (cf. section 2.4).

- *Taxonomic Description.* Taxonomic descriptions establish a rule to classify configuration objects by their type. For an object o a taxonomic refinement would be a set of objects that are of the same generic type but more specialized in some way. When given a set of objects $o_1, \ldots, o_n$, the taxonomic relations can be expressed by rules of the following form:

  $\langle \text{rule} \rangle \longrightarrow \langle \text{object} \rangle \rightarrow \langle \text{object} \rangle \mid \langle \text{objects} \rangle$
  $\langle \text{objects} \rangle \longrightarrow \langle \text{object} \rangle \vee \langle \text{object} \rangle \mid \langle \text{object} \rangle \vee \langle \text{objects} \rangle$
  $\langle \text{object} \rangle \longrightarrow o_1 \mid \ldots \mid o_n$

  Example: $o_i \rightarrow o_j \vee o_k \vee o_l$

  The semantics of this rule is as follows. Object o_i can be *realized* using object o_j, object o_k, or object o_l. An example for a taxonomic hierarchy is the classification of those computer's plug-in-cards that realize a graphics adapter.

Descriptions of function-based component models:

- *Property-based Description.* A property-based description reduces an object's behavior to a finite set of property-value-pairs. The properties may be of symbolic or of numerical type, i.e., the associated domain can be a number field or a simple list containing symbols. Normally, an object's property description is not very detailed from the technical standpoint. E.g., it is unusual to model some kind of gates that define in which way objects can be connected and where information about

properties can be passed. Rather, all properties of the objects involved in a configuration process are collected and processed globally.

A particular instance of property-based descriptions is given with *resource-based* descriptions. A resource-based description of an object o distinguishes between two kinds of properties: those that are supplied and those that are demanded by o. If o is part of the system to be configured, then, dependent on the type, its properties can be consumed or must be provided by other objects.

- *Behavior-based Description.* Similar to the property-based descriptions, the behavior-based modeling approach also specifies a finite set of properties for each object. Additionally, the following object-dependent concepts are realized:

  1. *Gates.* Gates define interfaces where other objects can be connected in order to pass information.
  2. *Constraints.* Here, it is sufficient to regard constraints as a set of relations. These relations are defined on different subsets of F, where F is the union set of an object's private properties and gates.

  A further distinction between stationary and dynamic behavior descriptions might result from the complexity of the constraints that specify the objects' behavior or from particular user demands: Do differential connections have to be considered when connecting objects? Will the system being configured carry out a time-dependent process?

Dimension 2: Configuration Tasks

Configuration tasks are orthogonal to the component models of the preceding subsection. In order to describe these tasks, we define the following different sets according to Stein and Weiner [81]:

- *Space of Compositions, M_{comp}.* Based on a finite set O of objects, the space of compositions comprises the combinations of all subsets of O and their structural permutations: $M_{comp} = \{(X, Y) \mid X \in \mathcal{P}(O), Y \in T(X)\}$; $\mathcal{P}(O)$ denotes the power set of O, $T(X)$ denotes the set of all structural arrangements (topologies) of an object subset X.

- *Space of Configurations, M_{conf}.* This set is defined as the subset of M_{comp} that comprises all configurations which can be realized concerning constructional or manufacturing constraints.
- *Space of Variants, M_{var}.* This set is defined in connection with a set D of demands. All configurations in M_{conf} that fulfill these demands form the space of variants M_{var}. I.e., for each set of demands there exists a particular set $M_{var}(D) \subseteq M_{conf}$. Note that $M_{var}(D)$ can be the empty set.

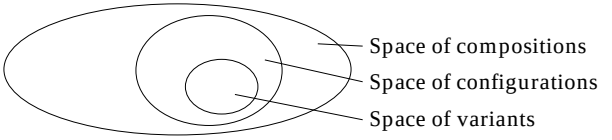


Figure 2.11: *Solution spaces of a configuration problem*

The most important configuration tasks are characterized briefly now. Starting point for each task is the set D of demands.

- *Creating a New Configuration.* Let O be the set of all objects and D the set of desired demands. Creating a configuration $C(D)$ means to select those objects from O that are necessary to realize C with respect to D. Especially for function-based models, the selection process is bound up with the process of finding a configuration's structure. Determining an appropriate structure is the most demanding part of a configuration process. If, after the processes of selection and structure definition, some technical properties of C still remain unknown, a parameterizing step must be performed.

  Often one is interested in an optimum configuration regarding a certain objective. Creating a configuration does not only mean determining some element of M_{var} or other, but finding the best. I.e., a configuration system must be able to investigate the total set M_{var} in an acceptable time.

- *Parameterizing a Configuration.* Let C be a configuration and D the set of desired demands. Parameterizing a configuration means to complete the functional information of C with respect to D. Depending on both the actual problem and D, a parameterization can be done by simple computations for the undetermined properties of C or also by a complex simulation of the entire functional model.

- *Checking a Configuration.* Let C be a configuration and D the set of desired demands. Checking a configuration means to investigate whether C fulfills all elements in D. Investigating if a configuration C is actually an element in M_{var} is much easier than creating a new configuration. Even so, such a task can be demanding when complex technical connections have to be simulated.
- *Adapting a Configuration.* Let C be a configuration and D' the set of desired demands. Adapting a configuration means to transform C into a configuration C' such that D' is fulfilled by C'. Usually, the adaptation of a configuration is more demanding than the creation of a new one. Adaptation encloses the following steps:

  1. Determining the demands D that are fulfilled by C.
  2. Creating a configuration C' that fulfills D'.
  3. Searching a (minimum cost) transformation from C to C'.

  Thus, at least all problems and restrictions of a creation process apply to the adaptation process as well.
- *Evaluating a Configuration.* Let C be a configuration and $c(C, D)$ a cost function. Evaluating a configuration means to compute the configuration's suitability with regard to c. This process must be performed when creating a new configuration in order to find the optimum one. Evaluating a configuration encloses the following steps:

  1. Determining the demands D that are fulfilled by C.
  2. Computing the function $c(C, D)$.

Table 2.1 presents the configuration tasks related to different component models and gives a rough evaluation of the problems' complexity.

Remarks. The evaluations indicate to what extent the configuration tasks can be supported and automated respectively and have to be interpreted as follows: "+" means that the task can be supported widely, so to speak, automated completely for the associated component model, "$\circ$" stands for partly, and "-" for no support. The parameterizing column does not make sense for structural component models.

Classification of the Scenarios

Based on the different component models and the configuration tasks, we are able now to classify the configuration scenarios of section 2.1 with

	Configuration task			
Component model	creating	parameter.	checking	adapting
associative	+	$\perp$	+	+
compositional	+	$\perp$	+	+
property-based	○/+	+	+	○/+
behavior-based	-/○	○/+	○/+	-

Table 2.1: *Matrix of important configuration problems*

respect to their technical complexity.

Configuration of Telecommunication Systems. The configuration of telecommunication systems is based to a large part on simple dependencies of the domain. There is no deep understanding of physical connections needed in order to configure a system. On the other hand, a system's structure depends on the actual demands of a customer. As a consequence, compositional knowledge can hardly be specified.

Classification: Above all, the component model comprises property-based descriptions. Additionally, particular domain heuristics must be employed that control the search. The main configuration task is the creation of new configurations.

Configuration of Mixing Machines. The configuration knowledge comprises deep functional connections on the one hand as well as compositional and taxonomic dependencies on the other. Since both types of knowledge can be processed independently from each other here, the configuration process does not contain creative parts. A lot of computations and recalculations need to be performed in the course of the configuration process, but a mixing machine's basic structure is predefined and will never change.

Classification: The component model consists of hard-wired behavioral connections and, for a smaller part, of structural knowledge. The main configuration task is parameterization.

Designing Hydraulic Circuits. In contrast to the former examples, this configuration or design job contains creative parts. The structure of a hydraulic system is not predefined and has to be created. Secondly, demanding

model formulation and simulation must be performed in order to compute unknown parameters as well as to check the system with regard to the desired demands.

Classification: An essential part of the component model are behavior-based descriptions. Configuration tasks are the creation of new configurations and parameterization.

Table 2.2 comprises these evaluations.

	Task	
Component model	creating	parameter.
structural	b	
property-based	a	
behavior-based	c	b, c

a = Telecommunication
b = Mixing
c = Hydraulics

Table 2.2: *Classification of the configuration scenarios*

Consequences

What does the component model view mean in connection with the development of configuration systems?

First, a distinction between model-based and heuristic configuration approaches is not very useful—a lot of approaches can be called model-based. This shall not exclude the fact that domain heuristics are necessary to process these models efficiently. Second, configuration systems should be seen as programs that operationalize a *description* level and a *processing* level in their knowledge base. The former defines an adequate component model of the domain while the latter realizes model-dependent computation methods, generic search strategies, and heuristic knowledge that controls the configuration process.

Apart from organizational aspects and knowledge acquisition problems, the following points are typical for configuration problems and demanding with regard to the development of configuration systems (cf. also Günter [24]):

- *Large Solution Space.* The number of constructions that could be composed is very large.

- *Rejection of Decisions.* Because of the large solution space, design decisions that are based on heuristics have to be met. As a consequence, decisions might be rejected in the course of the design process and need to be retracted with all their consequences.
- *Identification of Solutions.* The identification of a solution of a design problem or of a single design decision can be very difficult. Often, some kind of complex simulation is required. This situation will become more critical, if we are interested in an optimum solution with respect to a given objective.

In our view, it is useful to distinguish solely between the following two key objectives when developing configuration systems: (*i*) the support of human designers who work on creative tasks, and (*ii*) the nearly complete automation of "low-level" tasks. In fact, this distinction is the operational consequence of the discussion in section 2.2, page 29, where configuration problems were related to design problems.

Usually, problems of type (*ii*) are rather sales-oriented and placed at the customer's site, while the more creative tasks come up in the projecting and constructing divisions of a company. Consequently, depending on the main objective, there are different problems that can be addressed by a knowledge-based configuration system:

Problems of type (*i*) —what can be done by a configuration system:

- supporting auxiliary tasks like the process of drawing and layouting
- automating different parts in the process of model formulation
- computing mathematical models
- generating and simulating test assemblies
- checking organizational restrictions, such as legal conditions

Problems of type (*ii*) —what can be done by a configuration system:

- all items mentioned under (*i*)
- solving the technical problem both completely and automatically
- supporting users without or with less technical background
- calculating solutions that are optimum under given restrictions

- realizing concepts that simplify the knowledge acquisition process

What cannot be done by a configuration system in the foreseeable future:

- carrying out the sales consultation
- automating creative tasks like the design of a new system's structure

2.4 Configuration Methods

The classification scheme of the preceding section introduced different concepts to specify configuration knowledge: the structure-based and the function-based component models. For some of these component models, there exist suitable processing mechanisms. We outline these "configuration methods" here since they give a general understanding of how configuration works. Also, several concepts discussed in subsequent sections of our thesis rely upon the procedural view of particular component models.

General Considerations

In accordance with Maher, we see that a configuration process can be grounded on three main principles [48]:

- *Decomposition.* The decomposition principle states that a design problem can be decomposed into subproblems, which are of either of the following types: (*i*) object—the cognitive model of the configuration process is oriented by the physical components of a system. (*ii*) function—the cognitive model of the configuration process is oriented by the functions a system provides.
- *Cases.* Case-based configuration is grounded on solutions of previously solved problems. To solve a new problem, a similar case has to be identified and eventually modified. Thus, the configuration process employs "generalization knowledge" of the domain only to a small part.
- *Transformation.* Configuration by transformation means to transform an initial set of requirements into a solution. The configuration knowledge can be imagined as a set of transformation rules.

These process principles are not dependent on the component models—i.e.: In order to solve a configuration problem, any principle can be employed. Moreover, Maher claims that there is no connection between these principles and a particular domain or task:

> *" The distinction among the* (process) *models lies in the representation of design knowledge rather than in their appropriateness for a specific design domain or phase of design."*
>
> Maher, [48], p.52

Here, we shall not further investigate configuration processes due to the case-based or the transformation philosophy. The methods outlined subsequently are based on the decomposition principle: balance processing, skeletal configuration, and associative configuration. Beyond these methods, the relevant literature mentions also heuristic configuration approaches working e.g. to the propose-and-revise strategy or to the propose-and-exchange strategy. We do not count these approaches as configuration methods since they do not rely on a component model or on certain configuration knowledge. Rather, they describe universal concepts of how search problems (hence configuration problems too) can be handled.

Configuration by decomposition operationalizes the following intuitive principle: A complex system can be composed (= configured, synthesized) in the same way it can be decomposed (= analyzed). It is obvious that no new dependencies can be deduced by such an approach.

Balance Processing

Balance processing is a basic configuration method to process resource-based component descriptions. It has been operationalized within the configuration systems COSMOS [27], CCSC [41], AKON [37], and MOKON [68]. Note that resource-based descriptions are a particular instance of property-based descriptions. They will be suitable for a configuration problem, if the following conditions are fulfilled:

1. Structural information plays a minor role.
2. The components can be characterized by simple properties, which are supplied or demanded.
3. The components' properties have to be combined in order to provide the system's entire function.

In principle, balance processing operationalizes a *generate-and-test* strategy. The generate part, controlled by propose-and-revise heuristics, is responsible for selecting a set of objects. The test part simulates some kind of balance. In the first step the initial customer's demands are put on the demand side of the balance. Then, an object set is generated and the supplies and demands of these objects are also written on the corresponding sides of the balance. While doing so, identical properties are accumulated due to some rule, e.g. the algebraic "+"-operation. Within the next step, each property on the balance is checked whether the demanded value can be satisfied by the supplied one or not. In most cases, this check is done via a "$\leq$"-comparison. Figure 2.12 depicts the generate and the test phase of this configuration method.

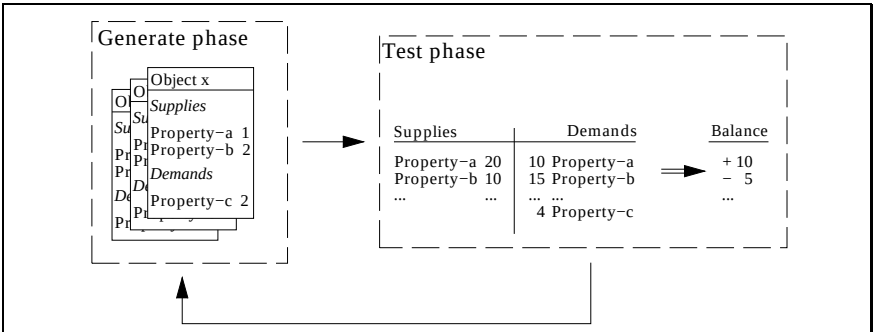


Figure 2.12: *The main phases of balance processing*

If all demands of the balance are fulfilled, the associated object set will represent a solution of the configuration problem. If not, the unsatisfied demands will form the input for the next step. The generate-and-test cycle is repeated until either a solution is found or no further object set can be generated. In chapter 4 we introduce the system MOKON that operationalizes this configuration method. Moreover, we show how the basic balance algorithm can be improved with regard to performance and knowledge acquisition.

Skeletal Configuration

Skeletal configuration is a basic method to process structure-based component descriptions. Among others, it has been operationalized within the configuration systems WIST [38] and PLAKON [13], [73]. The term "skeletal"

shall express that there is a (hierarchical) structure within the configuration problem that *never changes* for all the problems' instances. Skeletal configuration tackles configuration problems where a generic structure already exists or can be developed. An important representative of such configuration problems is the parts-list-processing of complex systems.

It is convenient to represent the skeletal structure by means of an and-or-graph [61]. As pointed out in section 2.3, page 33, the and-nodes realize compositional descriptions while the or-nodes are suitable to specify taxonomic dependencies. Such a combined taxonomic/compositional hierarchy represents explicitly parts of the configuration problem's search space. Figure 2.13 shows a system that on the first layer is composed of the objects $o_1 \ldots o_3$, where in turn o_1 and o_3 can be realized by alternative subcomponents.

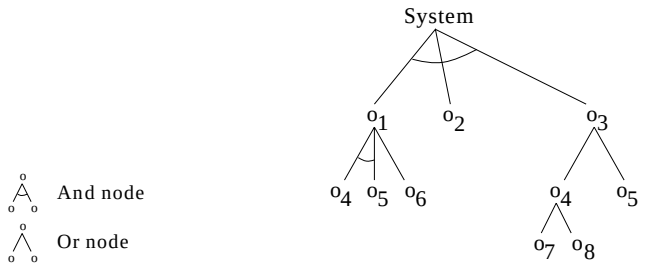


Figure 2.13: *A system's and-or-graph defines the possible configurations*

There are two processing strategies for taxonomic/compositional hierarchies:

1. *Top-Down.* Starting with the root node, each node v is processed as follows. If v represents an and-node, all subnodes of v will be selected. If v represents an or-node, exactly one of its subnodes will be selected according to some rule. Each subnode that is a leaf becomes part of the configured system; inner nodes are processed in a recursive manner. The configuration process will be completed if all selected nodes are either of leaf-node-type or expanded.

2. *Bottom-Up.* If there is information about particular components that shall become part of a configuration, the corresponding nodes will be instantiated. Then, by means of a bottom-up procedure, all those components sharing an and-relation with the instantiated ones are

also selected. If no further inference can be drawn, a top down refinement according to strategy 1 will be invoked, which considers all components previously selected.

In the course of both configuration strategies, a lot of backtracking can take place; the possible choice points are the or-nodes in the hierarchy. Backtracking can be reduced decisively if, aside from compositional and taxonomic descriptions, additional domain heuristics are employed. These heuristics may establish restriction rules between particular components, knowledge that controls the decision process at the choice points, or knowledge that realizes a dependency-directed backtracking. PLAKON, for example, provides powerful truth maintenance and constraint processing mechanisms to support such concepts.

Associative Configuration[7]

As a difference to the resource-based and the structure-based descriptions, associative knowledge provides no explicit domain model that a configuration process can rely upon. Thus, there exists no particular method to process associative configuration descriptions.

But, it is in the nature of configuration that only within simple configuration problems associative descriptions form the main part of the knowledge. Experience has shown that simple knowledge processing techniques such as decision tables and decision trees can be employed successfully here; they are both easy to realize and efficient for less complex decision problems.[8]

Loosely speaking, a decision table can be considered as an inflexible concept to represent and process rules: Each column defines a single rule where the first m rows specify the conditions to be matched while the last n rows specify the associated actions (cf. figure 2.14). Evaluating a decision table means to match from left to right each rule's conditions against the facts given until a fitting left-hand side is found.

A decision tree differs from a decision table in so far that it defines a sequence of how the knowledge is to be processed.

[7] The term "associative configuration" here has nothing to do with the associative configuration method presented by Tank in his dissertation [74].

[8] Puppe calls problems that can be solved by these techniques *definite classification problems* [61].

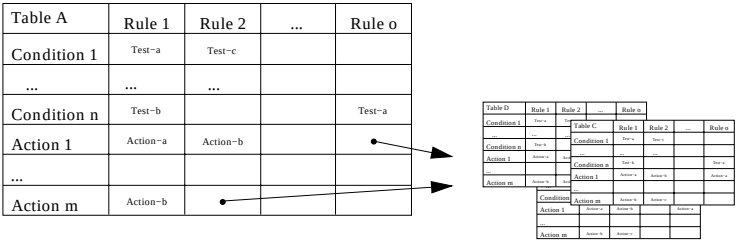


Figure 2.14: *Basic structure of decision tables*

Chapter 3

A Formal Framework of Configuration

Research in the field of configuration covers applicational and non-applicational work:

1. *Development of Configuration Systems.* This kind of research is concerned with the technical aspects of configuration problems and should answer the following question: *How* can a particular configuration problem be solved?
   In this connection it is pointed out how a particular domain can be modeled adequately, how dependencies between configuration objects can be handled, how hierarchies are processed, and how heuristics are employed, etc. This pragmatical point of view is justified because of the complexity of the "universal" configuration problem and makes up an important part of our work as well.

2. *Theory of Configuration Problems.* This kind of research is concerned with the nature and the complexity of configuration problems. The chapter in hand contributes to this research. It presents a precise specification of important configuration problems. Such a specification is useful in comparing given configuration problems with respect to their complexity and in revealing similarities or differences between the associated configuration methods. Besides this fact, a formalization can help in finding approaches that tackle a particular problem since it refrains from domain-specific descriptions and notions.

The formal framework presented originates from a paper by Najmann and Stein [56]. It is oriented by our classification scheme of the former

chapter and introduces three major classes of configuration problems: Section 3.1 defines property-based configuration problems. Section 3.2 defines configuration problems that additionally exploit structural information. Section 3.3 defines configuration problems that are based upon behavior.

By each of these configuration problems, a component model is defined precisely. They are called M1 (property-based), M2 (property-based and structure-based), and M3 (behavior-based).

The remainder of this chapter is organized as follows. In section 3.4 we show that model M1 and model M2 are equivalent—more exactly: the (global) structure information in model M2 can be expressed in terms of (local) property descriptions in model M1. The associated transformation can be done in polynomial time. The last section presents a complexity result concerning a relevant class of configuration problems.

3.1 Property-based Configuration Problems

In order to specify a configuration problem precisely, different sets and operators need to be defined: An object set that comprises the configuration objects given, a functionality set whose elements define all necessary features of the configuration objects, property sets that establish the actual functionality-value pairs of the configuration objects, operators that define computation rules and test predicates for the functionalities.

E.g., if we configured a computer, typical configuration objects would be the different harddisks, CPUs, power supply units, etc. The functionality set would contain elements such as `harddisk_capacity` and `graphics_resolution`. Furthermore, a harddisk could be described by the following property set: {`(adapter_type, SCSI)`, `(access_time, 7)`, `(harddisk_capacity, 120)`}. The computation rule of the functionality `harddisk_capacity` should be "$+$"; a useful test predicate that compares supplies and demands on `harddisk_capacity` would be "$\leq$".

Model M1, now introduced, is suitable to model property-based configuration problems. Most of this model is operationalized within the configuration system MOKON, which is described in section 4.4.

Definition 3.1 (Configuration Problem Π_{M1}). A configuration problem under model M1 (Π_{M1}) is a tuple $\langle O, F, V, P, A, T, D\rangle$ whose elements are defined as follows.

- O is an arbitrary, finite set. It is called the *object set* of Π_{M1}.
- F is an arbitrary, finite set. It is called the *functionality set* of Π_{M1}.
- For each functionality $f \in F$ there is an arbitrary, finite set v_f, called the *value set* of f. $V = \{v_f \mid f \in F\}$ is comprised of these value sets.
- For each object o there is a *property set*, p_o, which contains pairs (f, x), where (*i*) $f \in F$ and $x \in v_f$, and (*ii*) each functionality $f \in F$ occurs at most once in p_o. A property set specifies the values of certain functionalities of a given object. $P = \{p_o \mid o \in O\}$ is comprised of these property sets.
- For each functionality f there is an *addition operator*, a_f, which is a partial function $a_f : v_f \times v_f \rightarrow v_f$. An addition operator specifies how two values of a functionality will be composed to a new value, if a new object is *added* to a given collection of objects, which themselves describe a part of the system to be configured. $A = \{a_f \mid f \in F\}$ is comprised of all addition operators.
- For each functionality f there is a *test*, t_f, which is a partial function $t_f : v_f \times v_f \rightarrow \{\mathit{True}, \mathit{False}\}$. A test t_f specifies under what condition a demand (see below) is fulfilled. $T = \{t_f \mid f \in F\}$ is comprised of all tests.
- D is an arbitrary, finite set of *demands*. Each demand d is a pair (f, x), where $f \in F$ and $x \in v_f$. Additionally, the demand set must have the property that no functionality occurs more than once in D. A demand set D describes the desired properties of the system to be configured.

Remarks. An addition operator does not necessarily specify an addition between two numbers, rather any kind of operation is possible.

Typically, a demand set consists of *internal* and *external* demands. The external demands specify requirements that are supplied, for example, by a customer of the system to be configured. The internal demands are environmental or other requirements normally not specified by the customer.

So far, we have just defined the notion of a "configuration problem". We must, however, define what the solution of such a problem is. A configuration contains both objects and functionalities. Before we give an inductive definition of a configuration, we must define how properties can be composed.

Definition 3.2 (Composition). Let (f, x) and (g, y) be two properties and let $a_f(x, y)$ be defined. Then,

$$\varphi((f,x),(g,y)) = \begin{cases} \{(f, a_f(x,y))\}, & \text{if } f = g; \\ \{(f,x),(g,y)\}, & \text{otherwise.} \end{cases}$$

is called the composition of the properties (f, x) and (g, y).

Remarks. The composition of two properties is a set. This set contains either a single property if the functionalities are equal, or it contains these two properties if they are not equal. The rationale of this composition is as follows. If two objects, which have some properties in common, are included in a set containing the configuration objects, then it is necessary to "compute" the values of these properties in some way. This computation is done by the addition operator. If the addition operator is not defined for the given value constellation, then these two objects cannot occur at the same time in the set of configuration objects.

Based on the above definition of composition, we are now ready to formally introduce the notion *configuration*. A configuration must specify (*i*) the objects, which are parts of the system to be configured, and (*ii*) the entire functionality of the system.

Definition 3.3 (Configuration). Let $\Pi_{\mathrm{M1}} = \langle O, F, V, P, A, T, D\rangle$ be a configuration problem. A configuration is a pair $C = \langle I, Q\rangle$ where I is a set of *items* of the form (k, o) and Q is a set of *qualities* of the form (f, x). An item (k, o) means that object $o \in O$ is used k times in the configured system. A quality (f, x) means that the configured system has the functionality f with value x. A configuration C is inductively defined as follows.

1. $C = \langle \emptyset, \emptyset\rangle$ is a configuration.
2. If $C = \langle I, Q\rangle$ is a configuration and o is an object of O, then $C' = \langle I', Q'\rangle$ will be a configuration, if the following conditions hold:

   (*i*) For every $(f, x) \in p_o$ and for every $(g, y) \in Q$, the composition $\varphi((f, x), (g, y))$ is defined or $Q = \emptyset$.

   (*ii*)

$$I' = \begin{cases} I \setminus \{(k,o)\} \cup \{(k+1,o)\}, & \exists (k,o) \in I; \\ I \cup \{(1,o)\}, & \text{otherwise.} \end{cases}$$

(*iii*)

$$Q' = \begin{cases} \bigcup\{\varphi((f,x),(g,y)) \mid (f,x) \in p_o, (g,y) \in Q\}, & Q \neq \emptyset; \\ p_o, & Q = \emptyset. \end{cases}$$

3. Nothing else is a configuration.

Remarks. Condition (*i*) guarantees that only those objects $o \in O$ will be added to a given configuration C, if all object's properties p_o can be combined with all qualities of C. Condition (*ii*) specifies how one new object can be added to a given set of items I. Condition (*iii*) specifies how a new set of qualities will be constructed, if a new object is added to the configuration.

Although qualities and properties are syntactically equal, we distinguish between them since a property is the feature of an object, while a quality (f, x) is the result of the composition of several objects having the functionality f in their property sets.

Next we give a precise definition of the notion *solution* of a configuration problem.

Definition 3.4 (Solution of Π_{M1}***).*** A configuration $C = \langle I, Q\rangle$ is a solution of a configuration problem $\Pi_{\mathrm{M1}} = \langle O, F, V, P, A, T, D\rangle$ if and only if for each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$, such that $f = g$ and $t_f(x, y) = \mathit{True}$. The set $S(\Pi_{\mathrm{M1}}) = \{C \mid C \text{ is a solution of } \Pi_{\mathrm{M1}}\}$ is called the *solution space* of Π_{M1}.

Remarks. The above condition guarantees that all demands are fulfilled. Generally, there exists more than one solution of a configuration problem Π_{M1}. Sometimes $S(\Pi_{\mathrm{M1}})$ is called the "space of variants" (cf. section 2.3, page 36).

General Configuration Problems

Based on the above definitions, the following problems can be stated:

Problem CONF
Given: A configuration problem Π_{M1}.
Question: Does there exist a solution of Π_{M1}?

Problem FINDCONF
Given: A configuration problem Π_{M1}.
Task: Find a solution of Π_{M1}, if one exists.

Problem COSTCONF
Given: A configuration problem Π_{M1}, a cost function $c : O \rightarrow \mathbf{Q}^+$, and maximum cost $c^* \in \mathbf{Q}^+$.
Question: Does there exist a solution $C = \langle I, Q \rangle$ of Π_{M1} such that $\sum_{(k,o) \in I} k\, c(o) \leq c^*$?

Problem FINDCOSTCONF
Given: A configuration problem Π_{M1}, a cost function $c : O \rightarrow \mathbf{Q}^+$, and maximum cost $c^* \in \mathbf{Q}^+$.
Task: Find a solution $C = \langle I, Q \rangle$ of Π_{M1} with total cost at most c^*, if one exists.

Problem FINDOPTCONF
Given: A configuration problem Π_{M1} and a cost function $c : O \rightarrow \mathbf{Q}^+$.
Task: Find a cost-minimum solution $C = \langle I, Q \rangle$ of Π_{M1}, if one exists.

Remarks. Each of the problems above is essentially a combinatorial problem. In section 3.5 we prove that problem CONF is NP-complete. Note that the other problems are at least as hard as problem CONF.

Example

To illustrate the above configuration model we give a simple example for the problem FINDOPTCONF.

The goal is to build a tower of a given height at a minimum cost. Figure 3.1 illustrates the problem.

block	height	cost
A	1	2
B	2	3
C	3	6
D	3	5

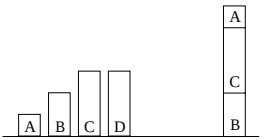


Figure 3.1: *A simple configuration problem*

As a special restriction we claim that the building blocks A and D may not occur both in a configuration. We will subsequently show that this restriction can be formulated with the aid of a particular functionality "restrict" and a corresponding operator a_{restrict}. The requirement is that the height of the tower must be at least 5.

By trivial enumeration of all solutions one can see that two cost optimum solutions exist. One is $\{(2, \mathrm{B}), (1, \mathrm{A})\}$ and the other one is $\{(1, \mathrm{D}), (1, \mathrm{B})\}$. In both cases the costs are 8. We now give the formal specifications of Π_{M1} related to this problem:

1. $O = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$
2. $F = \{\text{height, restrict}\}$
3. $v_{\text{height}} = \{1, 2, ..., 10\}$, $v_{\text{restrict}} = \{\mathsf{a}, \mathsf{d}\}$, $V = \{v_{\text{height}}, v_{\text{restrict}}\}$
4. $p_{\mathrm{A}} = \{(\text{height}, 1), (\text{restrict}, \mathsf{a})\}$, $p_{\mathrm{B}} = \{(\text{height}, 2)\}$, $p_{\mathrm{C}} = \{(\text{height}, 3)\}$, $p_{\mathrm{D}} = \{(\text{height}, 3), (\text{restrict}, \mathsf{d})\}$
   $P = \{p_{\mathrm{A}}, p_{\mathrm{B}}, p_{\mathrm{C}}, p_{\mathrm{D}}\}$
5. $a_{\text{height}}(x, y) = \begin{cases} x + y, & \text{if } x + y \leq 10; \\ \perp, & \text{otherwise.} \end{cases}$
   $a_{\text{restrict}}(x, y) = \begin{cases} \mathsf{a}, & \text{if } x = y = \mathsf{a}; \\ \mathsf{d}, & \text{if } x = y = \mathsf{d}; \\ \perp, & \text{otherwise.} \end{cases}$
   $A = \{a_{\text{height}}, a_{\text{restrict}}\}$. This definition is a formalization of the restriction rule.
6. $t_{\text{height}}(x, y) = \begin{cases} \mathit{True}, & \text{if } x \leq y; \\ \mathit{False}, & \text{otherwise.} \end{cases}$
   $t_{\text{restrict}}(x, y) \equiv \mathit{True}$
   $T = \{t_{\text{height}}, t_{\text{restrict}}\}$
7. $D = \{(\text{height}, 5)\}$

A formal solution of Π_{M1} is given by $C = \langle I, Q \rangle$ with $I = \{(2, \mathrm{B}), (1, \mathrm{A})\}$ and $Q = \{(\text{height}, 5), (\text{restrict}, \mathsf{a})\}$. One can check easily that all demands are fulfilled for the given quality set Q.

3.2 Structure-based Configuration Problems

Model M2, which will be introduced now, is suitable to model structure-based configuration problems. Among others, the configuration systems PLAKON [13], [73] and WIST [38] realize such a model.

Model M2 is similar to model M1 but extended by rules. These rules may be interpreted as assembling restrictions. For example, one would like to formulate the rule *"If harddisk A is used, then either controller B or controller C must be used."* By such restriction rules the skeleton of a technical system can be realized.

Definition 3.5 (Configuration Problem Π_{M2}). A configuration problem under model M2 (Π_{M2}) is a tuple $\langle O, F, V, P, A, T, D, N, R\rangle$ whose elements are defined as follows.

- Let O, F, V, P, A, T, D be defined as in model M1:
  1. O is set a of objects.
  2. F is a set of functionalities.
  3. V comprises all functionalities' value sets.
  4. P contains each object's property description.
  5. A is the set of all functionalities' addition operators.
  6. T contains a test predicate for each functionality.
  7. D comprises all desired properties of the system to be configured.
- With $N = \{1, \ldots, n\}$ let $\Gamma(N, O) = \{[k, o] \mid k \in N, o \in O\}$ denote the set of Boolean variables over N and O.
- A *configuration restriction rule* r is an implication $[k, o] \rightarrow \psi$ where $[k, o] \in \Gamma(N, O)$ and ψ is a logical formula over $\Gamma(N, O)$ using parentheses, '$\neg$', '$\wedge$', and '$\vee$' in the standard way. A *rule set* R is a finite set of configuration restriction rules over $\Gamma(N, O)$.

Let $C = \langle I, Q\rangle$ be a configuration as defined in model M1 where $I \subseteq N \times O$. Furthermore, we agree on the following notions:

Definition 3.6 (Configuration Assignment). A configuration assignment α_I is a function $\alpha_I : \Gamma(N, O) \rightarrow \{\mathit{True}, \mathit{False}\}$ such that for every $[k, o] \in \Gamma(N, O)$:

$$\alpha_I([k, o]) = \begin{cases} \mathit{True}, & \text{if } (k, o) \in I; \\ \mathit{False}, & \text{otherwise.} \end{cases}$$

Definition 3.7 (Satisfying Configuration). A configuration $C = \langle I, Q\rangle$ is called satisfying for a rule set R if and only if every rule $r \in R$ is true under the assignment α_I using the known semantics of propositional logic.

Remarks. The semantics of a restriction rule is illustrated by the following example: Let $r = [1, A] \rightarrow ([2, B] \wedge \neg[1, C]) \vee [3, D]$. The meaning of r is: *"If a configuration contains exactly one object A, then the configuration must contain either two B's and not one C or three D's."*

With the above definitions we are now able to specify a solution for an extended configuration problem Π_{M2} including restriction rules.

Definition 3.8 (Solution of Π_{M2}). A configuration $C = \langle I, Q \rangle$ is a solution of a configuration problem $\Pi_{\mathrm{M2}} = \langle O, F, V, P, A, T, D, N, R \rangle$ if and only if for each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \mathit{True}$ and C is satisfying for the rule set R.

The problems CONF, FINDCONF, etc. stated in subsection 3.1 exist in a similar way for model M2.

The solution space of structure-based configuration problems can be described by a hierarchical graph with two kinds of nodes: and-nodes and or-nodes. An and-node indicates that each direct successor of this node must be selected during the configuration process—more general: to solve the whole problem each subproblem has to be solved. An or-node describes mutually exclusive alternatives. In typical applications such a configuration problem is solved by *skeletal configuration* in a top-down strategy. Skeletal configuration will be appropriate, if we want to configure a system that always has the same basic structure (cf. section 2.4, page 43).

If Π_{M2} is a problem under model M2, then the skeleton of the configured system can be derived from the rules specified in Π_{M2}. The skeleton is the digraph $G = (V, E)$ with $V = O$ and $(o_i, o_j) \in E$, if there is a rule that contains o_i in its left-hand side and o_j in its right-hand side.

Example

In the following example a tower has to be configured that consists of three planes: An A-plane, a B-plane and a C-plane. For each plane there is a particular kind of building block (A-block, B-block and C-block respectively). The goal is to build a tower of a given height at a minimum cost (= FINDOPTCONF). Figure 3.2 describes the task.

In this example, we claim that the height of the tower must be at least 8. Additionally, the building blocks of the tower have to fulfill the following

block	height	cost
A1	1	2
A2	2	4
B1	1	3
B2	3	5
C1	1	2
C2	2	3
C3	4	6

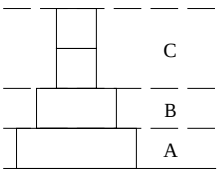


Figure 3.2: *Tower with the planes A, B, and C*

restrictions: For both plane A and plane B exactly one block of the appropriate kind should be selected. Plane C has to be constructed with one or two C-blocks where C3 cannot be combined with any of the other C-blocks. If block C2 is used once, block B1 will not be allowed to occur only once in a configuration.

By completely enumerating one can check easily that the following two cost optimum solutions exist: $\{(1,\ \text{A1}),\ (1,\ \text{B2}),\ (2,\ \text{C2})\}$ and $\{(1,\ \text{A1}),\ (1,\ \text{B2}),\ (1,\ \text{C3})\}$. The cost is 13 in either case.

To describe this problem as a hierarchical configuration problem we introduce particular "dummy blocks" S, A, B, C, which have no properties. With the new building blocks we are now able to give the formal specifications of Π_{M2} regarding our example:

1. $O = \{\text{S}, \text{A}, \text{B}, \text{C}, \text{D}, \text{A1}, \text{A2}, \text{B1}, \text{B2}, \text{C1}, \text{C2}, \text{C3}\}$
2. $F = \{\text{height}\}$
3. $v_{\mathrm{height}} = \{1,\ 2,\ ...,\ 10\}$, $V = \{v_{\mathrm{height}}\}$
4. $p_{\mathrm{S}} = p_{\mathrm{A}} = p_{\mathrm{B}} = p_{\mathrm{C}} = p_{\mathrm{D}} = \{\}$,
   $p_{\mathrm{A1}} = \{(\text{height},\ 1)\}$, $p_{\mathrm{A2}} = \{(\text{height},\ 2)\}$,
   $p_{\mathrm{B1}} = \{(\text{height},\ 1)\}$, $p_{\mathrm{B2}} = \{(\text{height},\ 3)\}$,
   $p_{\mathrm{C1}} = \{(\text{height},\ 1)\}$, $p_{\mathrm{C2}} = \{(\text{height},\ 2)\}$, $p_{\mathrm{C3}} = \{(\text{height},\ 4)\}$
   $P = \{p_{\mathrm{S}}, p_{\mathrm{A}}, p_{\mathrm{B}}, p_{\mathrm{C}}, p_{\mathrm{D}}, p_{\mathrm{A1}}, p_{\mathrm{A2}}, p_{\mathrm{B1}}, p_{\mathrm{B2}}, p_{\mathrm{C1}}, p_{\mathrm{C2}}, p_{\mathrm{C3}}\}$
5. $a_{\mathrm{height}}(x,\ y) = \begin{cases} x + y, & \text{if } x + y \leq 10; \\ \perp, & \text{otherwise.} \end{cases}$
   $A = \{a_{\mathrm{height}}\}$
6. $t_{\mathrm{height}}(x,\ y) = \begin{cases} \mathit{True}, & \text{if } x \leq y; \\ \mathit{False}, & \text{otherwise.} \end{cases}$

$T = \{t_{\text{height}}\}$

7. $D = \{(\text{height}, 8)\}$
8. $N = \{1, 2\}$
9. $R = \{$ [1,S] $\rightarrow$ [1,A] $\wedge$ [1,B] $\wedge$ [1,C],
   [1,A] $\rightarrow$ [1,A1] $\vee$ [1,A2],
   [1,B] $\rightarrow$ [1,B1] $\vee$ [1,B2],
   [1,C] $\rightarrow$ [1,D] $\vee$ [1,C3],
   [1,D] $\rightarrow$ [1,C1] $\vee$ [2,C1] $\vee$ [1,C2] $\vee$ [2,C2] $\vee$ [1,C1] $\wedge$ [1,C2],
   [1,C2] $\rightarrow \neg$ [1,B1] $\}$

Figure 3.3 shows the and-or-graph related to this problem. Note that the graph does not represent explicitly the exception defined by the last rule.

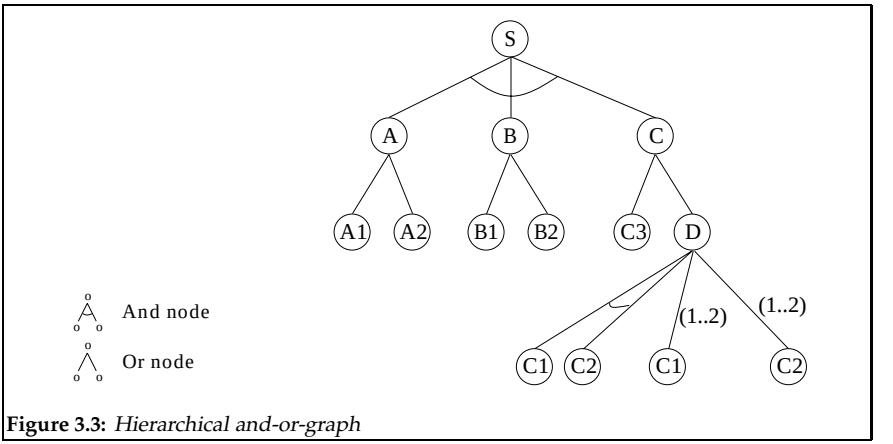


Figure 3.3: *Hierarchical and-or-graph*

A formal solution of Π_{M2} is given by $C = \langle I, Q \rangle$ with $I = \{(1, \text{A1}), (1, \text{B2}), (2, \text{C2})\}$, and $Q = \{(\text{height}, 8)\}$.

3.3 Behavior-based Configuration Problems

Within the former sections we developed the component models M1 and M2. The related configuration problems are largely based on the notion of functionality and represent a kind of *selection problem*: Given is a set of objects where the task is to select objects such that the required demands

are fulfilled. However, property-based and structure-based component descriptions are not sufficient for every kind of configuration problem—remember the behavior-based configuration problem presented within section 2.1, page 19. Therefore, we will modify model M1 by substituting *behavior descriptions* for a component's properties. Additionally, we give a formal definition of such behavior-based configuration problems.

Selecting and putting together objects from a set of objects so that the resulting system provides a desired behavior requires (*i*) the processing of behavior descriptions that are based on physical connections and (*ii*) the use of experience and creativity to control the process of selecting and connecting.

Within engineering domains the process described under (*ii*) is called *model formulation*. According to section 2.2, page 29, we will call a configuration problem of this quality a *design problem*.

Presently, there exists no general theory of how the creativity that guides a process of configuration or design can be automated. Thus, later within this section, we will define the less complex *checking problem*.

Definition 3.9 (Configuration Problem Π_{M3}***).*** A configuration problem under model M3 (Π_{M3}) is a tuple $\langle O, F, V, B, T, D\rangle$ whose elements are defined as follows.

- O is an arbitrary, finite set of objects.
- F is an arbitrary, finite set of functionalities. Each element in F denotes a possibly constant *function* in the parameter "time". For the sake of simplification, we will normally refer to $f(t) \in F$ as f. Note that a function invariant in time is equivalent to a functionality under model M1 or M2.
- For each functionality $f \in F$ there is an arbitrary, possibly infinite set v_f, called the *value set* of f. The elements of v_f are partial functions in the parameter time; i.e., they are defined on a subset of $\mathbf{R}^+$. $V = \{v_f \mid f \in F\}$ comprises these value sets. Note that a set v_f of functions invariant in time is equivalent to a value set v_f under model M1 or M2.
- For each object $o \in O$ there is an arbitrary finite set B_o, called the set of *behavior constraints*. Each behavior constraint $b \in B_o$ defines a relation on $v_{f_{b_1}} \times v_{f_{b_2}} \times \ldots \times v_{f_{b_k}}$ where $f_{b_i} \in F$, $\{b_1, b_2, \ldots, b_k\} \subseteq \{1, 2, \ldots, n\}$, and $n = |F|$. Such a relation may be specified by a function or by a

possibly infinite set of tuples. $B = \{B_o \mid o \in O\}$ is comprised of the sets of behavior constraints.

Based on the definition of behavior constraints, we agree on the following notions:

(*i*) $F_b = (f_{b_1}, f_{b_2}, \ldots, f_{b_k})$ is the tuple of the functionalities associated to the value sets $v_{f_{b_i}}$ upon which b is defined.

(*ii*) A tuple of functions $X = (x_1, x_2, \ldots, x_k)$, $x_i \in v_{f_{b_i}}$, $f_{b_i} \in F_b$ *matches (fulfills)* the behavior constraint b, if X stands in the relation defined by b.

- For each functionality f there is a *test* t_f, which is a partial function $t_f : v_f \times v_f \rightarrow \{True, False\}$. A test t_f specifies under what conditions a demand (see below) is fulfilled. $T = \{t_f \mid f \in F\}$ is comprised of all tests.
- D is an arbitrary, finite set of demands. Each demand $d \in D$ is a pair (f, x) where $f \in F$ and $x \in v_f$. If x is an invariant function of time, the demand will be called "stationary"; otherwise, the demand will be called "dynamic". Note that a stationary demand is equivalent to a demand $d \in D$ under model M1 or M2.

Remarks. Within the configuration problems Π_{M2} and Π_{M1} the configuration objects o are characterized by a set of properties p_o. In this place we introduced for each object o a set of behavior constraints B_o where each behavior constraint $b \in B_o$ defines a relation on a set of functionalities $F_b \subseteq F$.

For example, if we wanted to design an electrical circuit, typical objects in O would be resistors, capacitors, etc. The functionalities in F would specify the typical characteristics of the objects such as electrical resistances or capacities. Behavior constraints in this example would be Ohm's law and other electrotechnical regularities defined upon the functionalities in F.

Definition 3.10 (Configuration). Let $\Pi_{\mathrm{M3}} = \langle O, F, V, B, T, D \rangle$ be a configuration problem. A configuration is a triple $C = \langle E, Q, S \rangle$ where $E \subseteq O$ is a set of objects, Q is a quality set with tuples (f, x) where $f \in F$ and $x \in v_f$, and S is a set of tuples (f, g) with $f, g \in F$. E specifies those *elements* with which the configured system is to be realized. A quality $(f, x) \in Q$ assigns the possibly constant function x to the functionality f. S

defines the *structure* of C by means of the functionality-pairs to be unified. Additionally, we claim the following conditions to hold:

(*i*) Let $F_E = \{f \mid f \in F_b,\ b \in B_o,\ o \in E\}$ comprise all functionalities of C. Then, Q must be both *definite* and *complete* with respect to F_E, i.e.: For each functionality $f \in F_E$, there exists exactly one functionality-value-pair $(f, x) \in Q$.

(*ii*) If $(f, x), (g, y) \in Q$ and $(f, g) \in S$ then $x = y$.

(*iii*) Let $B_E = \{b \mid b \in B_o,\ o \in E\}$ comprise all behavior constraints of C. Then, Q must be *correct* with respect to B_E, i.e., to each $b \in B_E$ must apply:
The tuple of functions $X = (x_1, x_2, \ldots, x_k)$, induced by the functionality-value-pairs $(f_{b_i}, x_i) \in Q$, $f_{b_i} \in F_b$, matches (fulfills) the behavior constraint b. I.e., X stands in the relation defined by b.

Remarks. When selecting objects in order to form a configuration under model Π_{M1} or Π_{M2}, the functionalities of the objects are computed using the associated addition operator. I.e., "only" a selection problem has to be solved. When solving a configuration problem Π_{M3}, aside from such a selection problem the following have to be solved: a structure definition problem, a model formulation problem, and a model processing problem. It must be determined which functionalities of the selected components have to be unified in order to achieve a behavior that fulfills all demands. Such a unification is equivalent to a connection of the objects in a physical sense. It is specified by S where each element $(f, g) \in S$ indicates that the functionality f is to be unified with the functionality g, i.e., $f \equiv g$.

The conditions (*i*) – (*iii*) guarantee the technical correctness of a configuration C, that is to say, if C can be realized from its topology and its behavior: Condition (*i*) establishes that there is a definite specification for all functionalities of C. Condition (*ii*) claims each two functionalities will be equal if they are unified. Condition (*iii*) guarantees that all behavior constraints of C can be fulfilled by the functionality-value-pairs in Q.

Note that functional dependencies within the configuration problems Π_{M1} and Π_{M2} are modeled *globally*, with the aid of the functions' addition operators. In contrast to that, the functional dependencies within Π_{M3} are represented by *local constraints* and a list of functionalities to be unified. These constraints and the information about unification form the input for

a model synthesis process[1] that in turn yields a mathematical description of the system. In order to compute Q, which describes the system's global behavior, this mathematical description has to be processed by direct or by iteration methods.

Definition 3.11 (Solution of Π_{M3}). A configuration $C = \langle E, Q, S \rangle$ is a solution of a configuration problem $\Pi_{\mathrm{M3}} = \langle O, F, V, B, T, D \rangle$ if and only if the following conditions hold:

(*i*) C is a configuration according to *Definition 3.10*, i.e., C is technically correct.

(*ii*) For each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \mathit{True}$.

Remarks. A solution of a configuration problem Π_{M3} specifies which objects have to be selected, how they are parameterized, and how they are connected. Condition (*i*) claims a configuration's correctness while condition (*ii*) guarantees all demands being fulfilled according to the associated test predicate in T. There usually exists more than one solution of a configuration problem Π_{M3}.

The problems CONF, FINDCONF, etc. stated within section 3.1, page 51, can be formulated in a similar way for model M3.

Example

The following example establishes an instance of the behavior-based configuration problem Π_{M3}. Let us consider that we had to design a simple electrical circuit. Given are resistors, capacitors, and inductive coils. The goal is to *select*, *connect*, and *parameterize* the components in such a way that we obtain a damped oscillating circuit with a frequency of about $10kHz$ and a time constant of $0.2s$. Figure 3.4 illustrates the task.

In order to keep this example clear, we refrain from the specification of wires. Even such a simple problem, from the electrotechnical standpoint, needs more than a complete enumeration of possible subsets of the components. The design process is guided by the experience and the knowledge about physical connections. A formal specification of Π_{M3} related to our example is the following:

[1] We will elaborate on this term in chapter 5.

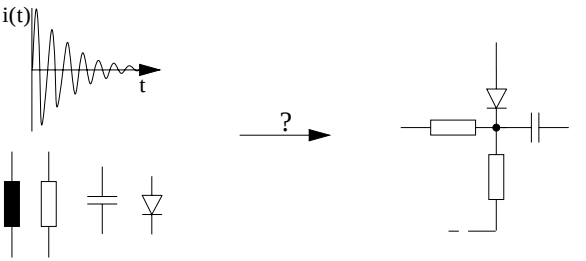


Figure 3.4: *Designing a simple circuit*

1. $O = \{$R (= *resistor*), C (= *capacitor*), L (= *coil*)$\}$
2. $F = \{$ resistance, $\mathrm{U1_R}$, $\mathrm{U2_R}$, $\mathrm{i1_R}$, $\mathrm{i2_R}$, capacity, $\mathrm{U1_C}$, $\mathrm{U2_C}$, $\mathrm{i1_C}$, $\mathrm{i2_C}$, $\mathrm{Q_C}$, inductivity, $\mathrm{U1_L}$, $\mathrm{U2_L}$, $\mathrm{i1_L}$, $\mathrm{i2_L}\}$

   Each element $f \in F$ is a real function; $f : \mathbf{R}^+ \rightarrow \mathbf{R}$.
3. $v_f = \mathcal{C}(\mathbf{R})$, the set of continuous functions on $\mathbf{R}$, $f \in F$,
   $V = \{v_f \mid f \in F\}$
4. $B_\mathrm{R} = \{$R-voltage, R-current$\}$
   $:= \{\mathrm{U1_R} - \mathrm{U2_R} = \text{resistance} \cdot \mathrm{i1_R},\ \mathrm{i1_R} = \mathrm{i2_R}\}$

   $B_\mathrm{C} = \{$C-voltage, C-current, C-charge$\}$
   $:= \{\mathrm{U1_C} - \mathrm{U2_C} = \text{capacity} \cdot \mathrm{Q_C},\ \mathrm{i1_C} = \mathrm{i2_C},\ \frac{d\mathrm{Q_C}}{dt} = \mathrm{i1_C}\}$

   $B_\mathrm{L} = \{$L-voltage, L-current$\}$
   $:= \{\mathrm{U1_L} - \mathrm{U2_L} = \text{inductivity} \cdot \frac{d\mathrm{i1_L}}{dt},\ \mathrm{i1_L} = \mathrm{i2_L}\}$

   $B = \{B_\mathrm{R}, B_\mathrm{C}, B_\mathrm{L}\}$
5. $t_f(\chi, \gamma) :\ \|\chi - \gamma\| \leq 0.1,$ with $f \in F,$ and $\chi, \gamma \in v_f$,
   $T = \{t_f \mid f \in F\}$
6. $D = \{(\mathrm{i1_R}, e^{-5t} \cdot sin(2\pi \cdot 10^4 \cdot t))\}$. This definition determines the circuit's oscillating frequency and its time constant.

A possible solution of the configuration problem is given in figure 3.5. It determines a function for each parameter and fulfills both all constraints and all demands.

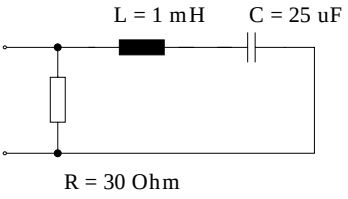


Figure 3.5: *A solution of the example*

Remarks. The solution of such a problem can hardly be found by a generate-and-test procedure but instead needs knowledge about model formulation in electrical engineering. To solve the above problem, a human expert would perform the following steps:

1. Identification of the circuit's topology.
2. Development of a global behavior model from the local behavior constraints. The subsequent differential equation represents a correct model regarding our example: $L \cdot \ddot{i} + R \cdot \dot{i} + \frac{i}{C} = 0$
3. Solution of the differential equation and evaluation of the resulting terms for the frequency and the time constant.

Checking a Configuration's Behavior

As mentioned above, the creative design process can only be automated to a small part. However, the complexity of a behavior-based configuration problem will definitely be reduced, if we restrict ourselves to the *checking* of a given configuration. Often, even such a checking problem turns out to be very sophisticated since we have to automate a model formulation process that is founded on physical relations and to process the resulting model. Chapter 5 introduces such a behavior-based checking problem and describes how it can be solved. In the following paragraphs, we define the checking problem precisely.

Definition 3.12 (Checking Problem Π^{c}_{M3}***).*** A checking problem under model M3 (Π^{c}_{M3}) is a tuple $\langle O, F, V, B, T, D, S \rangle$ whose elements are defined as in *Definition 3.9* and in *Definition 3.11* respectively.

Remarks. A checking problem Π^{c}_{M3} defines a set of objects, their local behavior concepts, and how the objects are connected. I.e., a specification

of a technical system and a set of demands are given where the goal is to *check* whether the system fulfills these demands or not.

Definition 3.13 (Solution of Π^{c}_{M3}). A solution of a checking problem $\Pi^{c}_{M3} = \langle O, F, V, B, T, D, S\rangle$ is a set Q containing tuples (f, x) where $f \in F$ and $x \in v_f$. Q is called a solution of Π^{c}_{M3} if and only if the following conditions hold:

(*i*) $C = \langle O, Q, S\rangle$ is a configuration according to *Definition 3.10*, i.e., C is technically correct.

(*ii*) For each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \mathit{True}$.

Remarks. This definition is similar to *Definition 3.11*. As a difference to the above, all objects of O are used to compose the system to be checked. Π^{c}_{M3} can be viewed as some kind of *constraint satisfaction problem*: The sets O and S establish a network of nodes where each node is characterized by a functionality $f \in F$. B and D define the constraints of this network and may be of both numerical or symbolic type.

Each set Q that is a solution of Π^{c}_{M3} defines a function $\gamma : F \rightarrow \bigcup v,\ v \in V$:

$$Q = \{(f_1, \gamma(f_1)),\ (f_2, \gamma(f_2)), \ldots, (f_n, \gamma(f_n))\}$$

In other words, solving this constraint satisfaction problem means to determine γ. If no such function exists, the checking problem Π^{c}_{M3} will be contradictory, i.e., the specified system does not fulfill the desired demands. A checking problem will be underspecified if more than one function γ exists.

3.4 Structural versus Functional Descriptions

Model M1 is suitable to model *property-based* configuration problems while model M2 is suitable to model *structure-based* configuration problems. Within the configuration problem Π_{M2} under model M2, the skeleton of the configured system is derived from the rules of Π_{M2}. In many applications the skeleton is a tree, and a configuration problem is processed by skeleton-configuration in a top-down strategy. In contrast, a property-based configuration problem is processed by means of balance processing (cf. section 2.4, page 43 and page 42).

The model M2, which additionally contains a mechanism to express *structural knowledge*, seems to be more powerful than the pure property-based model. However, as the following results shows, this outcome is not the case. This fact is quite surprising since one expects that the rule language enables us to formulate more sophisticated configuration problems.

A central theorem of this chapter is the following:

Theorem 3.14 (Equivalence of Model M1 and Model M2). 1. Let Π_{M1} be any instance of problem CONF under model M1. Then, there exists an equivalent instance Π_{M2} of problem CONF under model M2, which can be obtained in polynomial time in the size of Π_{M1}.

2. Let Π_{M2} be any instance of problem CONF under model M2. Then, there exists an equivalent instance Π_{M1} of problem CONF under model M1, which can be obtained in polynomial time in the size of Π_{M2}.

Corollary 3.15 (Equivalence of Model M1 and Model M2). *Theorem 3.14* is also valid for the problems FINDCONF, COSTCONF, FINDCOSTCONF, and FINDOPTCONF under model M1 and model M2.

By proving the theorem, the corollary follows immediately from the proof.

Proof. Part one: Let $\Pi_{\mathrm{M1}} = \langle O, F, V, P, A, T, D\rangle$ be an instance of problem CONF under model M1. Trivially, let $R := \emptyset$, $N := \{1\}$ and let $\Pi_{\mathrm{M2}} := \langle O, F, V, P, A, T, D, N, R\rangle$.

Part two: Let $\Pi_{\mathrm{M2}} = \langle O, F, V, P, A, T, D, N, R\rangle$ with $N = \{1, \ldots, k\}$ be an instance of problem CONF under model M2. We have to show that there exists an instance Π_{M1} of problem CONF under model M1 such that Π_{M2} has a solution if and only if Π_{M1} has a solution.

We construct Π_{M1} as follows. The basic idea of the proof is that the rules are replaced by *new functionalities* and new tests whose behavior is equivalent to these rules.

R is transformed into a logically equivalent set $\tilde{R}$, which contains only 3CNF formulas (i.e., propositional formulas in a conjunctive normal form where each clause has at most 3 literals).

This transformation is performed in two steps. First, a transformation technique due to Tseitin is used to transform each rule into a logically

equivalent propositional formula in a conjunctive normal form [77]. This step requires the introduction of new variables $\bar{\Gamma} = \{[1, \bar{o}_1], \ldots, [1, \bar{o}_T]\}$. Secondly, every formula obtained from step one is transformed into an equivalent 3CNF formula by introducing the new variables $\hat{\Gamma} = \{ [1, \hat{o}_1], \ldots, [1, \hat{o}_S]\}$. Note that both transformations can be made in quadratic time and linear space. Let $\tilde{\Gamma} = \Gamma(N, O) \cup \bar{\Gamma} \cup \hat{\Gamma}$.

Thus, every rule $r \in R$ is transformed into a set $3\mathrm{CNF}(r) = \{r_1, \ldots, r_S\}$ such that r is satisfiable if and only if every formula $r_i \in 3\mathrm{CNF}(r)$ is satisfiable. Let $\tilde{R} = \bigcup_{r \in R} 3\mathrm{CNF}(r)$.

The introduction of the new variables implies that the new object set O_{M1} is defined as $O_{\mathrm{M1}} := O \cup \bar{O} \cup \hat{O}$ with $\bar{O} = \{\bar{o}_1, \ldots, \bar{o}_T\}$ and $\hat{O} = \{\hat{o}_1, \ldots, \hat{o}_S\}$.

The subsequently described steps define how Π_{M1} is constructed from the new rule set $\tilde{R}$ and the new object set O_{M1}.

1. For every $r \in \tilde{R}$ we construct a new functionality g_r. The value sets of these functionalities in turn contain sets and are defined in step 3. For each $o \in O_{\mathrm{M1}}$ that occurs in a rule $r \in \tilde{R}$ we will extend the property set of o by the element $(g_r, \{(1, o)\})$.
2. To construct the value sets of the new functionalities we need the following "union" of an item set I and a singleton $\{(1, o)\}$, $o \in O_{\mathrm{M1}}$:

$$I \uplus \{(1,o)\} = \begin{cases} I \setminus \{(k,o)\} \cup \{(k+1,o)\}, & \text{if } o \in O, (k,o) \in I, \\ & \text{and } k \leq n, n = |N|; \\ I \cup \{(1,o)\}, & \text{if } o \in \bar{O} \cup \hat{O}, \\ & \text{and} (1,o) \notin I; \\ I, & \text{otherwise.} \end{cases}$$

3. A value set v_{g_r} is inductively defined as follows. Note that Λ is a help variable.

   (*i*) $\{\{(1, o)\} \mid o \text{ occurs in } r\} \subseteq \Lambda$.

   (*ii*) If $x \in \Lambda$ and o occurs in r, then $x \uplus \{(1, o)\} \in \Lambda$.

   (*iii*) Nothing else is in Λ.

   (*iv*) $v_{g_r} := \Lambda \cup \{r\}$.

   Note that v_{g_r} contains both sets of number-object pairs and the rule r itself. Also note that the computation of v_{g_r} can be done in a finite number of steps since $n + 1$ bounds the number k that can occur in a pair (k, o), which itself occurs in a set $x \in v_{g_r}$.

4. As a demand d_r for r we define $d_r := (g_r, r)$.

5. The test $t_{g_r}(x, y)$ of g_r is defined only for $x = r$ and $y \in v_{g_r} \setminus \{r\}$:

$$t_{g_r}(r, y) = \begin{cases} \mathit{True}, & \text{if } r \text{ is true under } \alpha_y; \\ \mathit{False}, & \text{otherwise,} \end{cases}$$

   where α_y is restricted to the variable set $\Gamma_r = \{[k, o] \mid k \leq n + 1,\ o \text{ occurs in } r\}$. Note that other variables than those in Γ_r cannot occur since $n + 1$ bounds the number k.

6. The addition operator $a_{g_r}(x, y)$ of g_r is defined only for $x \in v_{g_r} \setminus \{r\}$ and $y \in \{\{(1, o)\} \mid o \text{ occurs in } r\}$:

$$a_{g_r}(x, y) := x \uplus y$$

7. Henceforth, let $\rho(o) = \{(g_r, \{(1, o)\}) \mid r \in \tilde{R}_o\}$, where $\tilde{R}_o := \{r \in \tilde{R} \mid \text{o occurs in r}\}$.

8. The elements of $\Pi_{\mathrm{M1}} := \langle O_{\mathrm{M1}}, F_{\mathrm{M1}}, V_{\mathrm{M1}}, P_{\mathrm{M1}}, A_{\mathrm{M1}}, T_{\mathrm{M1}}, D_{\mathrm{M1}} \rangle$ are now defined as follows:

$$\begin{aligned} O_{\mathrm{M1}} &:= O \cup \bar{O} \cup \hat{O}, \\ F_{\mathrm{M1}} &:= F \cup F_R \text{ where } F_R := \{g_r \mid r \in \tilde{R}\}, \\ V_{\mathrm{M1}} &:= V \cup V_R \text{ where } V_R := \{v_{g_r} \mid r \in \tilde{R}\}, \\ P_{\mathrm{M1}} &:= \{p_o \cup \rho(o) \mid o \in O\} \cup \{\rho(o) \mid o \in \bar{O} \cup \hat{O}\} \\ A_{\mathrm{M1}} &:= A \cup A_R \text{ where } A_R := \{a_{g_r} \mid r \in \tilde{R}\}, \\ T_{\mathrm{M1}} &:= T \cup T_R \text{ where } T_R := \{t_{g_r} \mid r \in \tilde{R}\}, \\ D_{\mathrm{M1}} &:= D \cup D_R \text{ where } D_R := \{(g_r, r) \mid r \in \tilde{R}\}. \end{aligned}$$

Case A. We have to show: If $C = \langle I, Q \rangle$ is a solution of Π_{M2}, then there exists a solution $C_{\mathrm{M1}} = \langle I_{\mathrm{M1}}, Q_{\mathrm{M1}} \rangle$ of Π_{M1}. We show that there exist an item set $\triangle I$ and a quality set $\triangle Q$ such that $I_{\mathrm{M1}} = I \cup \triangle I$, $Q_{\mathrm{M1}} = Q \cup \triangle Q$, and $C_{\mathrm{M1}} = \langle I \cup \triangle I, Q \cup \triangle Q \rangle$ is a solution of Π_{M1}. Due to this construction of I_{M1} and since $D_{\mathrm{M1}} = D \cup D_R$, we need only to consider the "difference" demands D_R (the original demands D are satisfied by I).

We have to construct a $\triangle I$ in such a way that I_{M1} induces a quality set $\triangle Q$ with the following characteristic: For each $d = (g_r, r) \in D_R$ there exists a property $(g_r, y) \in \triangle Q$ with $t_{g_r}(r, y) = \mathit{True}$.

Let $d = (g_r, r)$ be any demand of D_R where $r \in \tilde{R}$ is a 3CNF rule of the form $r = l_1 \vee l_2 \vee l_3$ with $l_i \in \{[k,o] \mid [k,o] \in \tilde{\Gamma}\} \cup \{\neg[k,o] \mid [k,o] \in \tilde{\Gamma}\}$. Note that r is satisfied if some l_i is satisfied. Since C is a solution of Π_{M2}, it follows that all rules $r \in R$ are satisfied. Hence, all 3CNF rules in $\tilde{R}$ are satisfiable by some truth assignment $\alpha_{I_{\mathrm{M1}}}$. Note that $\alpha_I \subseteq \alpha_{I_{\mathrm{M1}}}$; this guarantees that an object $o \in O$ occurs with frequency k in I if and only if object o occurs with frequency k in I_{M1}. Without loss of generality, we can assume that l_1 ($= [k_1, o_1]$) is satisfied under $\alpha_{I_{\mathrm{M1}}}$.

Case A1. Let $o_1 \in O$. If $l_1 = [k_1, o_1]$ then $\alpha_{I_{\mathrm{M1}}}([k_1, o_1]) = \mathit{True}$, hence (k_1, o_1) must occur in I. The definition of a_{g_r} (cf. the "$\uplus$-operator") guarantees that a $y \in v_{g_r}$ with $(k_1, o_1) \in y$ is inevitably constructed as the quality value of g_r. If $l_1 = \neg[k_1, o_1]$ then $\alpha_{I_{\mathrm{M1}}}([k_1, o_1]) = \mathit{False}$ (hence $(k_1, o_1) \notin I$). Now, either $(k_2, o_1) \in I$ with $k_2 < k_1$ or $k_2 > k_1$, then $(k_2, o_1) \in y$, or o_1 does not occur in I at all and $(k_2, o_1) \notin y$. As before, an appropriate $y \in v_{g_r}$ is constructed as the quality value of g_r.

Case A2. Let $o_1 \in \bar{O} \cup \hat{O}$. If $l_1 = [k_1, o_1]$ then $\alpha_{I_{\mathrm{M1}}}([1, o_1]) = \mathit{True}$ and we put $(1, o_1)$ in $\triangle I$. The quality value y of g_r will contain $(1, o_1)$. If $l_1 = \neg[1, o_1]$ then $\alpha_{I_{\mathrm{M1}}}([1, o_1]) = \mathit{False}$ and o_1 is not allowed to be in $\triangle I$. Hence, the quality value y of g_r cannot contain $(1, o_1)$. So, $\triangle I$ is defined as the collection of all tuples $(1, o_1)$ found in case A2. Furthermore, let the g_r (with $r \in \tilde{R}$) and their corresponding y form the set $\triangle Q$. As seen above, with these definitions of $\triangle Q$ and $\triangle I$ it is guaranteed that for each $d = (g_r, r) \in D_R$ there exists the quality $(d_r, y) \in \triangle Q$ such that $t_{g_r}(r, y) = \mathit{True}$.

Case B. We have to show: If $C_{\mathrm{M1}} = \langle I_{\mathrm{M1}}, Q_{\mathrm{M1}} \rangle$ is a solution of Π_{M1}, then there exists a solution $C = \langle I, Q \rangle$ of Π_{M2}. Since $C_{\mathrm{M1}} = \langle I_{\mathrm{M1}}, Q_{\mathrm{M1}} \rangle$ is a solution of Π_{M1} all demands in $D_{\mathrm{M1}} = D \cup D_R$ are satisfied. Let $I = \{[k,o] \mid o \text{ occurs in } O\}$ and let $\Delta Q = \{(f, x) \in Q \mid f \in p_o, o \in \bar{O} \cup \hat{O}\}$.

(*i*) Clearly, $C = \langle I, Q_{\mathrm{M1}} \setminus \Delta Q \rangle$ satisfies all demands $d \in D$ since objects that occur in $I_{\mathrm{M1}} \setminus I$ have no properties for which a demand $d \in D$ exists.

(*ii*) To see that $C = \langle I, Q_{\mathrm{M1}} \setminus \Delta Q \rangle$ satisfies all rules $r \in R$, one need only to consider the transformation above, which guarantees that α_I is a satisfying truth assignment since $\alpha_I \subseteq \alpha_{I_{\mathrm{M1}}}$ and $R \iff \tilde{R}$. $\diamond$

Example

With the specification of the transformation of model M2 into model M1, we are now able to reformulate the configuration problem of section 3.2 as a problem that is solely based on functionalities and their computation. Figure 3.6 describes the configuration problem as an and-or-graph.

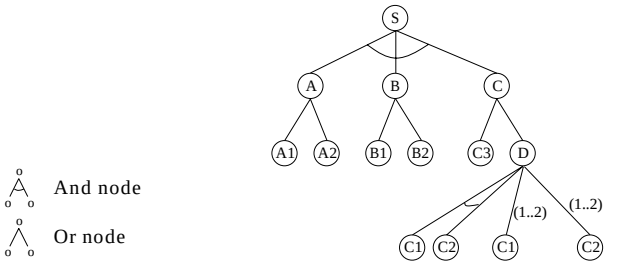


Figure 3.6: *The configuration problem of section 3.2*

Here, we will not perform this transformation explicitly; the necessary transformation steps were specified above. Note that in the reformulated problem the structural dependencies between the objects must be derived from their properties.

Figure 3.7 shows the reformulation of the configuration problem and illustrates in which way the transformation of M1 into M2 comes into effect. The transformation of the above rules yields eight new functionalities $g_1, \ldots, g_8$. In the figure, both functionalities and configuration objects are vertices; an edge (o_i, g_j) indicates that the object o_i has the functionality g_j in its property set.

3.5 A Complexity Consideration

Although we did not specify an algorithm for any of the above problems CONF, FINDCONF, etc., we present in this section a result regarding the computational complexity of CONF under model M1 and M2 respectively. Note that (*i*) all other problems under one of the above models are at least as hard as CONF, and (*ii*) all problems under model M3 are at least as hard as their equivalents under model M1 and M2 respectively.

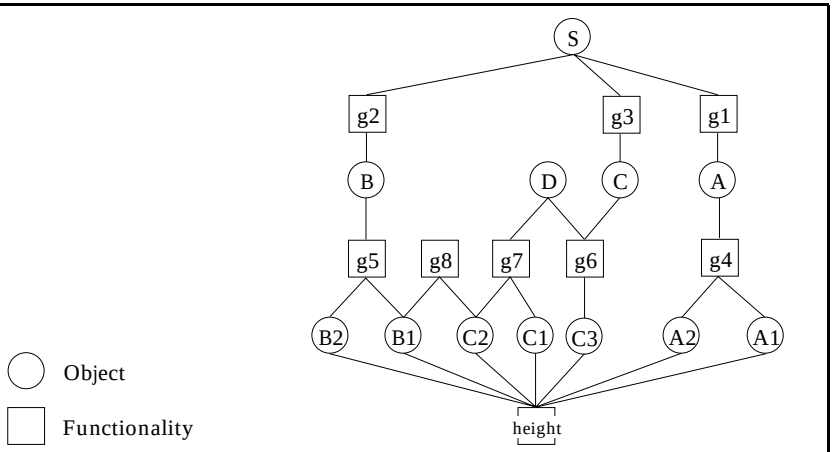


Figure 3.7: *The new dependency graph*

Theorem 3.16 (NP-Completeness of* CONF*). Problem CONF under model M1 or M2 is NP-complete.

Lemma 3.17 (NP-Completeness of* CONF$_R$*). Problem CONF with restriction rules (CONF$_R$) under model M1 or M2 is NP-complete.

We first prove the lemma, then the theorem.

Proof of Lemma. Let Π_{M2} be any instance of CONF$_R$. Clearly, CONF$_R$ $\in$ NP since a nondeterministic algorithm need only to guess a configuration $C = \langle I, Q\rangle$ and check in polynomial time whether this configuration is a solution of Π_{M2}.

We transform 3SAT (which is NP-complete, Cook, [12]) to CONF$_R$. Let $X = \{x_1, \ldots, x_n\}$ and $Z = \{c_1, \ldots, c_m\}$ be any instance of 3SAT where X is the set of variables and Z is the set of clauses. We must construct a configuration problem $\Pi_{\mathrm{M2}} = \langle O, F, V, P, A, T, D, N, R\rangle$ such that Π_{M2} has a solution $C = \langle I, Q\rangle$ if and only if Z is satisfiable.

The key idea is to transform every clause into a logically equivalent restriction rule. This is established as follows.

1. With every variable x_i we associate an object of the same name and a functionality f_i, i.e., $O = \{x_1, \ldots, x_n\}$, and $F = \{f_1, \ldots, f_n\}$.

2. The value set of a functionality f_i is the set $v_{f_i} = \{1, 2\}$, for each $1 \leq i \leq n$. Let $V = \{v_{f_i} \mid 1 \leq i \leq n\}$.
3. The property set of an object o_i is the set $p_{o_i} = \{(f_i, 1)\}$. Let $P = \{p_{o_i} \mid 1 \leq i \leq n\}$.
4. For every $f_i \in F$ the addition operator a_{f_i} is defined as $a_{f_i}(x, y) = 2$, if $x = y = 1$; in all other cases a_{f_i} is undefined.
5. With each functionality f_i we associate a test $t_{f_i}(x, y) \equiv x \leq y$ for all $x, y \in v_{f_i}$. Let $T = \{t_{f_i} \mid 1 \leq i \leq n\}$.
6. The demand set is defined as $D = \{(f_1, 1), \ldots, (f_n, 1)\}$.

Note that if $C = \langle I, Q \rangle$ is a solution of Π_{M2}, then the definition of A, D, and T imply that every object x_i must occur either once or twice in I.

The set of restriction rules is constructed as follows. Let $c_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$ be a clause of Z where l_{i_j} is a literal over X, for $1 \leq j \leq 3$ and $1 \leq i \leq m$. Define

$$\tau(l_{i_j}) = \begin{cases} [1, x_{i_j}], & \text{if } l_{i_j} \text{ is a positive literal;} \\ [2, x_{i_j}], & \text{if } l_{i_j} \text{ is a negative literal.} \end{cases}$$

Note that c_i is logically equivalent to $\neg l_{i_1} \rightarrow l_{i_2} \vee l_{i_3}$. Therefore we associate with c_i the rule $r_i = \tau(\neg l_{i_1}) \rightarrow \tau(l_{i_2}) \vee \tau(l_{i_3})$. For example, if $c_i = x_2 \vee \neg x_4 \vee x_5$, then $r_i = [2, x_2] \rightarrow [2, x_4] \vee [1, x_5]$. It is obvious that this transformation can be made in polynomial time.

First, suppose that $\alpha : X \rightarrow \{\mathit{True}, \mathit{False}\}$ is a satisfying truth assignment for Z. It is easy to see that the following configuration $C = \langle I, Q \rangle$ fulfills all demands and satisfies all "restriction rules":

$$I = \{b(x_i) \mid 1 \leq i \leq n\}$$

with

$$b(x_i) = \begin{cases} (1, x_i), & \text{if } \alpha(x_i) = \mathit{True}; \\ (2, x_i), & \text{if } \alpha(x_i) = \mathit{False}. \end{cases}$$

and

$$Q = \{(f, c(x_i))\} \mid 1 \leq i \leq n\},$$

with

$$c(x_i) = \begin{cases} 1, & \text{if } \alpha(x_i) = \mathit{True}; \\ 2, & \text{if } \alpha(x_i) = \mathit{False}. \end{cases}$$

Since every c_i is logically equivalent to its transformed rule r_i it is obvious that c_i is satisfiable if and only if r_i is satisfiable. Therefore, all

rules $r \in R$ are satisfied, which can be seen from the definition of $b(x_i)$. To see that the demands are also fulfilled, one should note that every object x_i occurs in I with either property $(f_i, 1)$ or $(f_i, 2)$ and that, therefore, the demand $d_i = (f_i, 1)$ is fulfilled.

Conversely, suppose that $C = \langle I, Q \rangle$ is a configuration which fulfills all demands and which is satisfying for the rule set R. Since it fulfills all demands, it follows that each object x_i occurs exactly once or twice in I. Therefore, every rule $r \in R$ can be applied in some manner. Since every rule r_i is satisfied by the assignment α_I it follows that its corresponding clause c_i must also be satisfied. ◇

Proof of Theorem. Let Π_{M1} be any instance of CONF. Clearly, CONF $\in$ NP since a nondeterministic algorithm need only to guess a configuration $C = \langle I, Q \rangle$ and check in polynomial time whether C is a solution of Π_{M1}.

We transform CONF$_R$ to CONF. Here we make use of *Theorem 3.14*, which states that every $\Pi_{\mathrm{M2}} \in$ CONF$_R$ is equivalent to a $\Pi_{\mathrm{M1}} \in$ CONF. The only point that remains to be shown is that the transformation used in the proof of *Theorem 3.14* can be made in polynomial time.

Here we can restrict ourselves to 3CONF$_R$ whose instances contain only rules of the form $r = [k_1, o_{i_1}] \rightarrow [k_2, o_{i_2}] \vee [k_3, o_{i_3}]$. Clearly, 3CONF$_R$ is NP-complete since every instance of 3SAT is transformed to an instance of 3CONF$_R$ (with $k_i \in \{1, 2\}$, $1 \leq i \leq 3$) in the previous proof.

Let R be a rule set with 3-element rules. It is obvious that CNF$(r) = \{r\}$ for every rule $r \in R$. That means, no rule transformation must be made. One can also easily check that all other transformations from 3CONF$_R$ to CONF can be made in polynomial time. ◇

Part II

Operationalizing Configuration Tasks

Chapter 4

On Property-based Configuration

The main theme of this chapter can be summarized with the following quotation:

> *"Models of design processes provide guidance in the development of knowledge-based systems for design. The basis for such models comes from research in design theory and methodology as well as problem solving in AI."*
>
> Maher, [48], p.49

—chapter 2 and 3 presented a classification and a formalization of different component models. Now we shall concentrate on a particular function-based component model: the property-based component model. We will show how this component model can be used to develop efficient configuration systems.

The chapter is organized as follows.

Section 4.1 discusses the philosophy and general characteristics of the property-based configuration approach. Section 4.2 presents a basic algorithm that realizes different configuration tasks. Section 4.3 shows how the basic configuration algorithm can be improved by the use of metaknowledge. The last section gives a short description of MOKON, a configuration system that realizes the concepts and algorithms outlined.

4.1 The Rationale of Property-based Configuration

The computational complexity of a configuration approach depends on the type of the underlying component model. Property-based configuration makes use of functional connections of a domain. I.e., the component model being established and processed is grounded on functional properties of the components being configured.

Aside from the computational complexity, so to speak, the knowledge processing costs, there exists another central characteristic of component models: the "knowledge acquisition costs". With this term we designate the effort necessary for maintenance, modification, and acquisition of configuration knowledge.

In order to understand the advantages of functional component descriptions, the next two subsections illustrate some relationships between different component models and knowledge processing as well as knowledge acquisition costs. The last subsection elaborates on the link between property-based and resource-based descriptions.

No Function in Structure

Knowledge processing costs and knowledge acquisition costs are a direct consequence of the *no-function-in-structure-principle*, which in turn is related to the "locality" of a component's description.—What does the no-function-in-structure-principle (= NFIS-principle) mean?

Loosely speaking, a component will fulfill the NFIS-principle, if its description does not depend on its context of use [10], [16], [40].

To understand the rationale of this principle, let us consider that we had a toolbox of components with each component representing some kind of physical device. The components possess gates where they can be connected to other components, and to each gate a particular parameter p is associated. E.g., p could designate a physical quantity such as electric current, voltage, or temperature, but also other kinds of information are eligible. Also, each component may establish a relation on its gates, say, parameters. Examples for such relations are $p_1 + p_2 = p_3$ (adder) or Ohm's law.

The information about the gates and the relation form a component's description. Note that the description of a component that fulfills the NFIS-principle must remain valid under all circumstances, regardless of how it is connected. The following example will illustrate this characteristic.

To model an electric circuit, we are given wires, switches, etc. For the sake of clarity, we focus on the behavior description of a switch. A switch has two gates, and the relation that specifies its behavior is as follows. *"If the switch is open, there is no flow of current at its gates. If the switch is closed, there is a flow of current at its gates."*

Figure 4.1 depicts two circuits where switches of this type are used. For circuit A our description of the switch is sufficient. However, since this description is not *context-free*, it fails to be a valid description for the switches of circuit B: Although one switch of this circuit is closed, there cannot be a current at its gates.

Figure 4.1: *Two simple circuits*

Note that the switch's description seems to be local, since it refers to its own gates only. By contrast, the following description refers not only to its own but also to external (environmental) gates: *"The light bulb will shine, if the switch is closed."*

Nevertheless, both descriptions should be called global since they make assumptions about their context of use.

To investigate the behavior of a system that is composed by locally described components, some kind of *model formulation* is necessary. The first step is that all local and all environmental behavior descriptions must be determined. Secondly, these local descriptions must be composed and transformed into a global description such that the behavior of the entire system is modeled correctly. Such a global description might be an equation system of differential and non-differential connections. If so, the solution of the equation system should also be counted along with the model formulation process. Conversely, a global description establishes a system whose process of model formulation is essentially completed.

Obviously, the following rule applies: (*i*) A component will obey the NFIS-principle only, if its descriptions are local, and (*ii*) processing local component descriptions is usually much more demanding than processing global descriptions.

Component Models and Locality

Different component models have different portions of configuration or design knowledge *compiled in*. This quality determines the role component models play with respect to knowledge processing and knowledge acquisition costs. Moreover, this role can be explained in terms of the NFIS-principle or the "locality" of a component description and is now discussed for the component models of section 2.3.

- *Associative Component Model.* Models of this type are global with respect to the readily configured system. In more detail: Associative knowledge consists of rules that describe *explicitly* component relations and configuration actions. This knowledge is derived—and thus, justified—with the global system in mind. Clearly, descriptions of this type violate the NFIS-principle. They make assumptions about the components in hand, the system's structure, the environment, etc. The advantage of such a description is that it can be processed both efficiently and easily. No other constraints need to be checked beside those that are specified in the condition part of the rules. Often, large parts of an expert's knowledge consist of such rules.

  Two main disadvantages of associative configuration descriptions are the following:

  1. The explicit formulation of all configuration dependencies can lead to redundant descriptions. Moreover, the configuration knowledge cannot be structured, and consequently, it is difficult to maintain and to check existent connections or to specify new ones.
  2. Associative descriptions establish no causal dependencies. I.e., the configuration knowledge cannot be used to generate explanations of configuration steps other than the associative rules themselves.

- *Compositional Component Model.* Models of this type are global, too. Therefore, the statements given above concerning the NFIS-principle

apply here as well. Processing compositional knowledge is efficient because of the semantics of compositional relationships. (Remember the top-down and the bottom-up strategy described in section 2.4, page 43.)

Compositional knowledge can be represented with little redundancy, and the skeleton model of the configuration description is often used to structure the knowledge. Nevertheless, knowledge acquisition is not easy, since the modification, exchange, or addition of a component affects its entire subskeleton. The drawback of weak explanation mechanisms applies to this type of knowledge as well.

- *Taxonomic Component Model.* Taxonomic knowledge organized in an or-hierarchy is global. I.e., it is efficient since components of the same type can be found in constant time. But, taxonomic knowledge can also be local—namely, if the components' characteristics, which form the base for a classification, are made explicit. In this case some kind of classification algorithm like in KL-ONE needs to be employed [3]. Note that such a description might come pretty close to a property-based description. In any case, since taxonomic connections are usually employed to describe and to organize compositional knowledge, this evaluation should not be overrated.

- *Property-Based Component Model.* Models of this type are local. The configuration knowledge is not specified by explicit relations between the components but rather by the components' local properties. These properties are used to derive necessary compositional and taxonomic dependencies.

  Example: If we wanted to configure a computer's power supply, an associative description would explicitly define the crucial relations between different power supply units and other components. When realizing a property-based modeling, all components have the property `(current_value, X)`. During a configuration process all components' current values are computed in order to select a suitable power supply. Compared to the associative case, this configuration process exploits deeper dependencies of the domain.

  Components that are described by their local properties fulfill the NFIS-principle, of course. This results in an important advantage of property-based models: Configuration knowledge can be acquired and maintained easily. Modification, exchange, and addition of components will never affect other parts (components) of the configura-

tion knowledge. Also expressive explanations of configuration decisions can be generated since the configuration process's underlying model is functional, that is to say, causal.

These advantages are bought with a considerable increase in knowledge processing costs. Compared to the structural component models, a larger search space has to be processed. The reason for this is that no explicit configuration decisions, which would guide the configuration process, are predefined. Rather, the configuration process is constrained implicitly by the local component descriptions that must form a correctly working global model when put together.

- *Behavior-Based Component Model.* In principle, all characteristics of property-based component models apply to behavior-based models as well. We say "in principle" here since the processing of behavior descriptions is usually so difficult that a configuration process that is solely based on such deep connections cannot be operationalized completely.

Remarks. Component models that fulfill the NFIS-principle can be maintained much easier than those models which do not. This difference results from the fact that no global dependencies are specified within the descriptions of such models.

There is a tradeoff between knowledge processing cost on the one hand and knowledge acquisition cost on the other. Especially when realizing a property-based component model instead of a structure-based one, the benefit of efficient knowledge processing is given up for the—often more desirable—benefit of user-friendly knowledge acquisition. Figure 4.2 illustrates this tradeoff qualitatively.

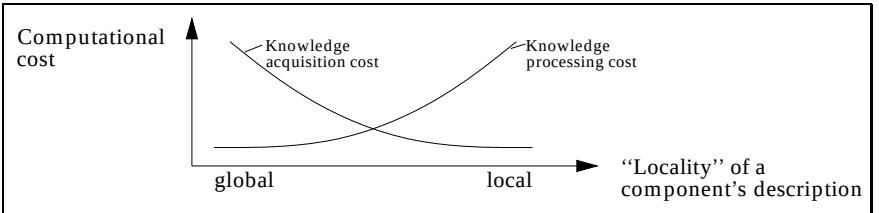


Figure 4.2: *The tradeoff between knowledge processing and acquisition cost*

The Supply-and-Demand Semantics

Back to property-based component models. How can property-based component models be realized? Experience has shown that it is often useful to consider the properties of a property-based component model as some kind of *resource* [27]. I.e., a component characterized by a resource f either supplies or demands a certain amount of f. In figure 4.3 resource-based component descriptions are depicted graphically.

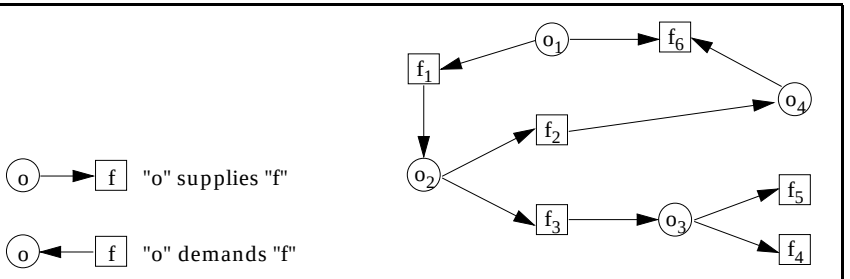


Figure 4.3: *Resource-based component descriptions*

Note that this dependency network represents a simplified functional model of the domain. Configuration that is founded on such connections means instantiation and simulation of this functional model. The representation also shows the power of property/resource-based descriptions with respect to knowledge acquisition and maintenance. All relationships are of the canonical form

$$\text{component} \leftrightarrow \text{resource} \leftrightarrow \text{component}$$

As a consequence, components can be added or removed easily by the creation or the removal of resource links. The role of each component in the configuration problem becomes clear, and basic plausibility checks become possible. Example: Each resource must be supplied by at least one component.

4.2 A Basic Algorithm

This section presents a basic configuration algorithm to process resource-based component descriptions. Most of the resource-based configuration problem is equivalent to the property-based configuration problem

Π_{M1}= $\langle O, F, V, P, A, T, D\rangle$ under model M1, defined in section 3.1, page 48:

- O is set a of objects.
- F is a set of functionalities.
- V is comprised of all functionalities' value sets.
- P contains each object's property description.
- A is the set of all functionalities' addition operators.
- T contains a test predicate for each functionality.
- D is comprised of all desired properties of the system to be configured.

The task is to find a solution (i.e. a configuration) $C = \langle I, Q\rangle$ of Π_{M1} such that for each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ with $f = g$ and $t_f(x, y) = \mathit{True}$.[1]

Remarks. Resource-based modeling distinguishes between supply and demand properties of the components. This supply-and-demand semantics is not reflected explicitly by the objects' property sets p_o in the definition of Π_{M1}. Of course, this fact is not a restriction of model M1 since supplies and demands can be defined implicitly by the addition operators: Without loss of generality we may claim that $g = (v_f, a_f)$, $v_f \in V$, $a_f \in A$, $f \in F$, represents a group in the algebraic sense. Then, if (f, x) is a supply of f to the amount of x, (f, y) will be a demand of f to the same amount, if $a_f(x, y)$ produces the neutral element of g. Henceforth, we will use p_o^S (or p_o^D) to refer to an object's supplied (or demanded) properties.

If there exists a configuration C that solves the resource-based configuration problem, C can be determined with the algorithm SATBALANCE. This algorithm operationalizes a generate-and-test strategy. In the generate part a set of objects is selected while in the test part the objects' functionalities are balanced. Usually, the selection process is controlled by the set of the actually unsatisfied demands and by domain-dependent propose-and-revise heuristics. During the balance process all supplies and demands of a functionality are accumulated and checked as to whether the demanded value can be satisfied by the supplied one or not. The algorithm will terminate if all demands are satisfied or no further object set can be generated.

[1] This task corresponds to the problem FINDCONF under model M1.

A non-deterministic version of the basic balance algorithm is given now. It is non-deterministic since it employs the functions $select\_demand$, $create\_alternatives$, and $select\_alternative$, which heuristically control the search. We elaborate on these functions in the next section. Additionally, we need the help function $test\_balance$ that takes a demand set D and an item set I as input and returns the set of actually unsatisfied demands.

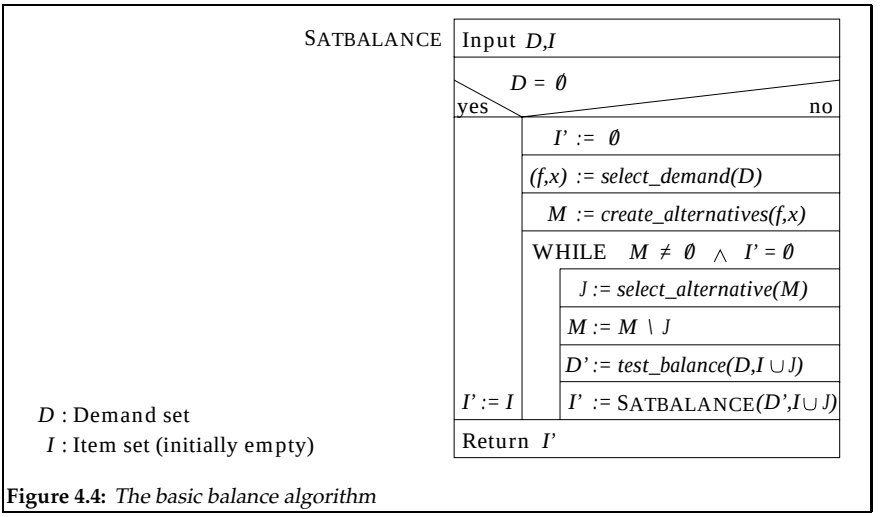


Figure 4.4: *The basic balance algorithm*

Remarks. SATBALANCE can be modified easily to solve the configuration task "adapting". Thus, all configuration tasks defined in section 2.3 are solved by the algorithm above.

In order to solve a realistic configuration problem, SATBALANCE must be extended. E.g., it should be possible to process minimum and maximum restrictions, to define mandatory functionalities, or to define an optimum criterion.

4.3 Improving Performance with Metaknowledge

The efficiency of the algorithm SATBALANCE depends on the domain heuristics operationalized in the functions $create\_alternatives$, $select\_alternative$, and $select\_demand$ [36]. Such heuristics are called metaknowledge. Metaknowledge is knowledge about knowledge and defines how (configuration) knowledge is used and which role (configuration) knowledge plays.

In the functions mentioned metaknowledge plays a key role since it controls the configuration process:

- $create\_alternatives$, $select\_alternative$. Creating alternatives means to form sets of components that satisfy a particular demand; selecting an alternative means to define an order based upon these sets. Metaknowledge related to both jobs is stated in the form of preferences: Should the alternatives be formed and selected with respect to the component costs, or some kind of indirect cost—caused by the new demands of an alternative, or with respect to an alternative's number of components? Answers to these questions are founded on experience, on domain-dependent constraints, or on connections found out empirically.
- $select\_demand$. Metaknowledge related to this function should answer the question, which unsatisfied demand is to be processed next. There exist domain *in*dependent relationships that can be exploited to answer this question. The next subsection shows how information about the demand processing order is deduced from the component descriptions.

The Planner's View

Property-based configuration problems can be described in terms of planning. The following are given:

1. an initial state, characterized by Π_{M1},
2. a goal state C (configuration) that is only described intensionally by the fact that C fulfills all demands related to Π_{M1}.

What we are looking for is a *plan* that describes how to get from the initial state to the goal state. Usually, such a plan is a concatenation of a finite number of so-called operators. When dealing with property-based configuration, a plan has to determine which of the unsatisfied demands should be processed next and how the selected demand could be satisfied. Processing such a plan would result in the readily configured system.

Especially the selection of unsatisfied demands plays a crucial role when processing a property-based configuration problem. If the dependencies between the functionalities are considered, a lot of backtracking can be avoided. Hence, we are interested in a plan that defines a sequence by

which unsatisfied demands can be processed so that backtracking is minimized. Before we describe how such a plan is found we introduce the following definitions.

Definition 4.1 (Serializable). Let Π_{M1} be a configuration problem under model M1. Π_{M1} is called *serializable*, if a permutation $s = (f_{s_1}, f_{s_1}, \ldots, f_{s_n})$, $f \in F$, $n = |F|$, can be determined such that the following holds true: For each demand $d = (f_{s_i}, x) \in D$ that can be satisfied at all, there exists an item set J whose objects exclusively demand functionalities f_{s_j} with $j > i$. The permutation s shall be called the *linear configuration plan* of Π_{M1}.

Definition 4.2 (Strongly Serializable). Let Π_{M1} be a configuration problem under model M1 and s a linear configuration plan of Π_{M1}. Π_{M1} is called *strongly serializable* if the demands are processed in order of s and no backtracking is necessary, regardless of the alternatives chosen to satisfy the demands.

Remarks. In other words, a configuration problem Π_{M1} is serializable if there exists a total order s of F (and hence of D) such that the satisfaction of a demand (f_{s_i}, x) will never lead to an additional demand (f_{s_k}, y), $s_i, s_k \in s$ with $k < i$. Note that despite the existence of a linear configuration plan, backtracking might occur during the configuration process due to the following reason: When processing a demand d, usually a lot of alternative component sets are locally suited to satisfy d. Selecting a "wrong" alternative will cause the configuration process to reach a dead end within subsequent configuration steps.

The latter definition describes configuration problems that will be solved by means of a greedy algorithm (if the linear configuration plan guides the configuration process). Typically, very few property-based configuration problems are of this type.[2]

Note that a linear configuration plan can only exist if no circular dependencies occur between the functionalities in F.

Theorem 4.3 (Linear Configuration Plan). Let Π_{M1} be a configuration problem under model M1 and n the number of all components' supplied and demanded properties, $n := \sum_{o \in O} |p_o^S| + \sum_{o \in O} |p_o^D|$. Then, a linear

[2]The configuration system R1/XCON [54], [64] deals with strongly serializable problems only. R1/XCON was developed by the Carnegie-Mellon University and DEC from 1978–1980 to perform the configuration of VAX computers. It is still maintained and improved today.

configuration plan can be computed within a time complexity of $O(n)$, if one exists.

Before we prove the theorem we motivate its idea with respect to the component-property graph of figure 4.5.

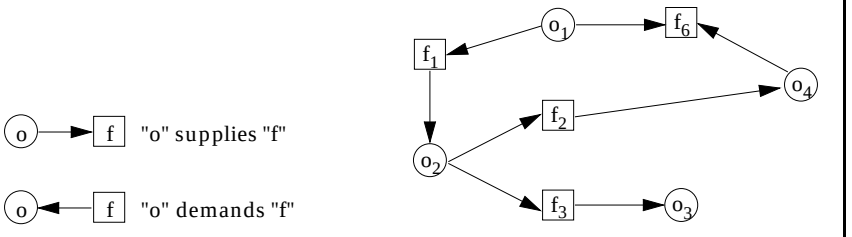


Figure 4.5: *A sample component-property graph*

A demand $d \in D$ should only be processed, if all components of the system that also need this functionality are already determined. E.g., since component o_3 supplies nothing, it should be selected first, and while o_1 demands nothing, it should be selected last. Obviously, a component's number can be determined if its outdegree in the component-property graph is zero (on condition that the components selected and the functionalities processed are deleted in the graph). Note that the sequence of nodes we get by this procedure corresponds to a reversed topological sorting of the graph. The order by which functionalities occur in this sorting defines the optimum configuration plan.

Proof. We need to prove that for an arbitrary component-property graph $g(V, E)$ a topological sorting can be computed in $O(n)$, $n := \sum_{o \in O} |p_o^S| + \sum_{o \in O} |p_o^D|$. By the construction of $g(V, E)$ we can see that $|V| = |O|$ and $|E| = \sum_{o \in O} |p_o^S| + \sum_{o \in O} |p_o^D|$. Moreover, all edges of g are directed since no functionality is both supplied and demanded by the same component. Because $g(V, E)$ might contain cycles, all strongly connected components of g have to be computed first. According to Aho et al. this computation can be done in $O(|E|)$ for a connected directed graph [1].

When exploiting the information about the strongly connected components, we are able to construct the condensed graph where each strongly connected component is represented by one node. The condensed graph is computed as follows. All nodes of a strongly connected component are associated with the same number. Then, for each strongly connected component S a new adjacent list is computed by merging the adjacent lists of

the nodes in S. During this merger $O(|E|)$ comparisons are performed. Now the resulting (connected) graph can be topologically sorted with a complexity of $O(|E|)$. ◇

Remarks. The proof outlines the steps that are necessary to compute a linear configuration plan. If a strongly connected component contains more than one node, no linear plan will exist since the functionalities involved in a cycle have to be considered simultaneously. In such a case, the order between the strongly connected components that is computed in the second step is not total with respect to F. Note that there might exist more than one linear configuration plan since the topological sorting of a graph is not necessarily definite.

4.4 The Configuration System MOKON

The configuration system MOKON operationalizes property-based configuration. It automates the configuration process for all tasks described in section 2.3 (creating, adapting, and checking a configuration) [69]. Aside from the concepts presented in the previous sections, MOKON also realizes graphical support for knowledge base maintenance.

Component Model Representation. The property-based component model in MOKON provides a scheme for defining both objects and properties. Table 4.1 depicts some exemplary object and property definitions.

Identifier	Conf_Object	
Supply	Property_A	400
	Property_B	Yes
Demand	Property_C	Red
	Property_D	17
Min	0	
Max	2	
Cost	220	
Stock	130	

Identifier	Property_A	Property_F
Type	Numerical	Date
Operator	+	Date_Minimum
Comparison	$\geq$	Date$\leq$
Constraint	Realizable	Unrealizable
Priority	5	1
Interface	Internal	Ask_User
Range	[80, 2000]	Today - 31.12.94
Default	-	Today

Table 4.1: *Component and property definitions in* MOKON

For maintenance reasons, all configuration objects are organized in a taxonomic hierarchy.

Some of the attribute fields in the property definition scheme need to be explained:

- *Operator.* Declares the computation method for the functionality. The values of this attribute field correspond to the addition operators a_f of Π_{M1}.
- *Comparison.* Declares the test predicate t_f for a functionality f. The test predicate defines whether a supply fulfills a given demand.
- *Constraint.* Defines if a demand at this functionality can be fulfilled at all. This information is used within the function $create\_alternatives$.
- *Interface.* Defines if alternatives are to be selected by MOKON or by the user.

Configuration Method. The main configuration method in MOKON is balance processing. The basic algorithm is extended by associative knowledge such as minimum/maximum restrictions and by default mechanisms, and, moreover, it provides a priority control. The priority control detects cyclic dependencies and computes a linear configuration plan, but also allows a user to define his individual preferences.

For a given demand set D the space of variants $M_{var}(D)$ can be computed. The search process realizes a global optimization with respect to the total costs of a configuration and has two search strategies built in: (*i*) "select-component-minimum-alternative" and (*ii*) "select-cost-minimum-alternative". Additionally, MOKON allows dependency-controlled and interactive backtracking as described by Marcus et al. [51].

Knowledge Base Inspecting. The component-property graph that is defined implicitly by object descriptions can be visualized in MOKON. The drawing of a component-property graph is heuristically controlled and considers the semantics of the configuration knowledge. Arbitrary objects and functionalities of the graph can be selected and investigated with respect to the cause-effect relations upon which they are based. Thus, the component-property graph browser provides both explanation facilities and acquisition support.

Chapter 5

Behavior-based Configuration in Hydraulics

As stated in the heading, here we focus on a configuration problem where a *behavior-based* component model must be employed: The configuration of hydraulic systems.

The configuration of hydraulic systems is founded on deep physical connections. It needs creativity and, at the present time, cannot be automated.

Nevertheless, powerful design support in hydraulics is still possible. This chapter presents a new view to the configuration of hydraulic systems and shows how to tackle sophisticated parts of the hydraulic design procedure.

Section 5.1 introduces the configuration of hydraulic systems and discusses the consequences for a configuration support. It becomes clear that an *automated analysis* of hydraulic systems will be the major engine of configuration support. Thus, we investigate in section 5.2 the analysis step in greater detail. Section 5.3 presents a generic component model that allows the formulation of hydraulic checking and parameterization problems, and section 5.4 addresses the processing of this component model. Here we show how the analysis step and, as a consequence, the checking and parameterization problem in hydraulics can be automated. In addition to the inference concepts in section 5.4, we develop in section 5.5 a preprocessing approach for a particular class of hydraulic behavior constraints.

5.1 Configuration of Hydraulic Systems

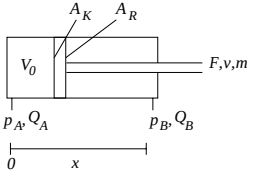
The starting point for a configuration process is a task that shall be performed by hydraulics. This task might be a lifting problem, the actuation of a press, or the realization of a robot's kinematics. The result of the configuration process is a system consisting of hydraulic and, eventually, some mechanical and electronic components.

Hydraulic Components

Hydraulic components are the building blocks in our configuration process. It is useful to introduce some of their underlying physical principles in order to convey an idea of the configuration process's complexity. Note that hydraulic engineering is a domain which cannot be treated in detail here.

Hydraulic components can be divided into three classes: (*i*) working components like cylinders, (*ii*) control components like valves, and (*iii*) service components such as pumps, tanks, and pipes. All components are described by their stationary and dynamic behavior.

Cylinders are the actuators of a hydraulic system; they transform hydraulic energy into mechanical energy. Valves in the form of relief valves, throttle valves, proportional valves, or directional valves control flow and pressure of the hydraulic medium. Pumps provide the hydraulic energy, i.e., the necessary pressure p and flow Q. Figure 5.1 and 5.2 show the basic structure of a differential cylinder and a proportional valve respectively. Below these figures a small extract of the related behavior description is given.



A_K, A_R = Piston areas		p = Pressure	
d_z = Piston friction		Q = Flow	
E'_{Oil} = Oil elasticity		V_0 = Cylinder volume	
F = Force		v = Velocity	
m = Mass		x = Elongation	

Figure 5.1: *Basic structure of a differential cylinder*

$$
\begin{array}{lcll}
F & = & p_A \cdot A_K - p_B \cdot A_R - d_z \cdot v & \text{(stationary force balance)} \\
Q_A & = & A_K \cdot v & \text{(continuity condition)} \\
F(t) & = & p_A(t) \cdot A_k - p_B(t) \cdot A_R - m \cdot \dot{v}(t) - d_z \cdot v(t) & \text{(force balance)} \\
\dot{p}_A(t) & = & \frac{E'_{Oil}}{V_0 + A_K \cdot x(t)} \cdot \big(Q_A(t) - A_K \cdot v(t)\big) & \text{(pressure rise)}
\end{array}
$$

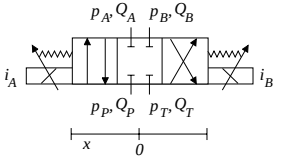


Figure 5.2: *Basic structure of a proportional valve*

$$
\begin{array}{lcll}
x & = & K_p \cdot (i_A - i_B) & \text{(position of valve piston)} \\
R_h(x) & = & R_{h_{min}} \cdot \frac{x_{max}}{x} & \text{(valve resistance)} \\
position & = & \text{crossed, if } i_A < i_B & \text{(valve position)}
\end{array}
$$

$$
\text{If crossed} \left\{ \begin{array}{lcll}
Q_P & = & -Q_B & \text{(continuity condition)} \\
p_P & = & p_B + sign(Q_P) \cdot R_h \cdot Q_P^2 & \text{(valve pressure drop)}
\end{array} \right.
$$

Note that hydraulic components may have states that determine which part of their behavior description is actually valid.

Configuration Procedure

The demands on a hydraulic system result from the task to be performed and are specified by different diagrams. These diagrams indicate the course of the forces and velocities of the cylinders, the switching positions of the valves, and other dependencies. Based on such diagrams, an engineer has to design the system's topology, to select the necessary components, and to check among others the stationary and the dynamic behavior of the system [42].

The most creative part in the configuration process is the design of a system's topology. The selection and parameterization of the components are also demanding and need a lot of experience as well as technical and mathematical know-how. Due to the complexity of the configuration process, we cannot get from the demand set D to the readily configured system

C in one single step. Rather, there is a cycle of synthesis, parameterization, analysis, evaluation, and modification of different configurations C. In accordance with Gero, Figure 5.3 illustrates this cycle. Here, B_e denotes the expected behavior that can be derived canonically from D, S_C denotes the hydraulic system's structure, and B_C denotes the behavior that is produced by the configured system C.

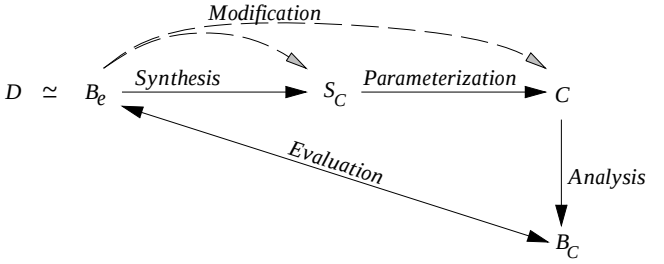


Figure 5.3: *An abstract view of hydraulic system design*

Let us take a closer look at the configuration procedure.

Synthesis. Within the synthesis step the topology of a hydraulic system is designed. I.e., the basic interplay of the components is defined such that the emergent system might fulfill the demands qualitatively. At the end of this step exists a plan of a circuit ($= S_C$) that defines the connections of all cylinders, the important control valves, and the pumps and tanks.

Parameterization. Within the parameterization step the components of the planned circuit are dimensioned. Typical parameters to be determined are the geometries of the cylinders, the resistances of the valves, the power of the pumps, and different pressure thresholds.

Analysis. Main job of the analysis step is the simulation of the hydraulic system C. In this connection, the engineer decides to which level of detail the components' behavior must be modeled in order to obtain useful simulation results. While the stationary behavior is always investigated, the dynamic behavior is simulated for sensitive parts of the system only. Note that both cases are difficult from the mathematical standpoint since equation systems with non-linear and differential relations must be processed. Moreover, several assumptions about the components' states have to be met in order to set up a global behavior description. If a state's assumption is wrong, the whole computation will fail and a new global description

must be set up. Proposing useful assumptions needs both experience and a thorough investigation of the system's topology.

Evaluation. Within the evaluation step the analysis results are balanced with the demands D. Among others, the following questions must be answered: Does the switching logic realize the desired behavior? Will the piston velocities and forces be as prescribed? Are the maximum pressure values permissible?

Modification. Input for the modification step is the interpretation of the deviations found during the evaluation stage. E.g., if a hydraulic system has logical faults, the topology must be adapted or redesigned. Correcting dimensional faults means to select components of the same type but with a different characteristic: valves, for instance, are exchanged with respect to their hydraulic resistance, cylinders with respect to their cross-section. After such a modification all computations have to be performed again.

Consequences for a Configuration Support

Let us again consider the design procedure in figure 5.3. Efficient configuration support seems to be hardly possible because of the creativity that is needed within the synthesis step $B_e \longrightarrow S_C$. On second thought, however, the situation is not hopeless: Neither the creative synthesis step nor the experienced-based modification steps are time-consuming for a *human expert*. Put another way, a reduction of just the analysis step's complexity would lead to a noticeable simplification of the entire design procedure.

This observation will guide our philosophy of a configuration support—rather than automating the entire configuration process we will concentrate on those tasks that ground on the *analysis* of hydraulic systems. In this connection the *checking* tasks play a key role: Checking a hydraulic system comprises complex analysis and evaluation jobs and is essential to detect both technical faults and violations of user demands.

Note that automating the time-consuming checking tasks will allow human experts more room for creative jobs.[1] Moreover, efficient checking concepts form the base for other tasks such as parameterization and optimization:

- *Parameterization.* Unknown parameters must be determined and checked, if they fulfill all restrictions.

[1] We discussed the two levels of support in section 2.3, page 39.

- *Optimization.* A parameterization problem usually has several solutions. The optimum can only be found by skillfully investigating all consistent alternatives.

5.2 Analyzing the Analysis of Hydraulic Systems

As argued in the previous section, an automated analysis of hydraulic systems is the key for a configuration support in hydraulics. This section investigates the analysis step in more detail.

From Local Behavior to Global Behavior

Loosely speaking, hydraulic systems analysis takes a circuit diagram as input and produces a behavior description of the entire circuit. For this job, aside from the simulation problem, some kind of *model synthesis* problem has to be tackled also.

By model synthesis we comprise all steps that are necessary to set up a model which is both correct in a physical sense and *locally* unique.

Note that even though a circuit diagram may establish a correct physical model, it is not locally unique as a rule: Each component of the circuit is defined by a set of behavior constraints from which the actually relevant ones must be selected. Verifying the correctness of a local behavior description needs an expensive simulation of the *entire* system in most cases.

The indeterminacy of local behavior descriptions originates from the following reasons:

1. *Level of Description.* Each hydraulic component can be described at various levels. To avoid superfluous computational effort, an adequate level of detail, respecting both the rest of the circuit and the simulation intention, has to be determined for each component. On the other hand, we have to ensure that all components' descriptions fit together.
2. *Dynamical Simulation.* It must be analyzed which part(s) of the circuit require a dynamical investigation. For the possible components a stationary *and* a dynamic behavior model must be determined.
3. *Component States.* Most components have different physical states,

each coupled with a particular behavior description. The actual validity of a state depends on the entire system and the actual input parameters. Example: A pressure relief valve may be opened or closed.

4. *Topology.* A hydraulic system's topology can change with a component's state. Example: Dependent on its switching position a proportional valve connects different parts of a hydraulic network.
5. *Physical Thresholds.* Even for a fixed state the direction or the absolute value of a physical quantity, which is a-priori unknown, may cause different behaviors of a component. Example: A turbulent flow is described by another pressure drop law than a laminar flow.

Tackling points 1 and 2 requires an engineer's experience and heuristical knowledge. The points 3, 4, and 5 reveal that the analysis step $C \rightarrow B_C$ also contains a selection step $C \rightarrow M_C$, where M_C denotes a set of behavior descriptions that are valid in the actual situation of the hydraulic system. Often, an additional cycle of model selection and model simulation is necessary in order to solve this selection problem. Figure 5.4 illustrates the situation.

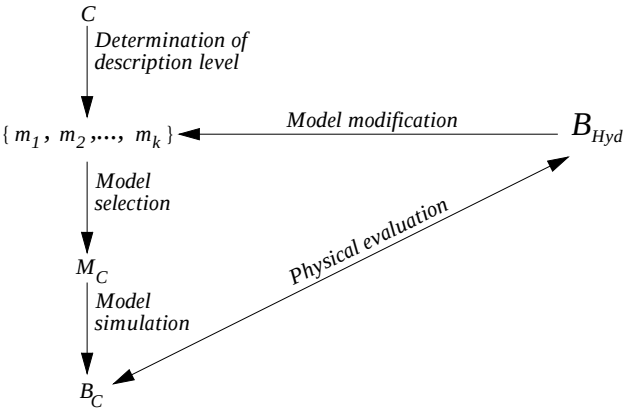


Figure 5.4: *What happens during the hydraulic analysis step*

Remarks. Let $\{m_1, \ldots, m_k\}$ comprise the behavior descriptions, say, models of all components at the adequate level according to point 1 and 2. From this set a subset is selected ($= M_C$), simulated ($\Rightarrow B_C$), and compared to $\mathcal{B}_{Hyd}$, which stands for the universal behavior laws of hydraulics. In the case that

the simulated behavior B_C is physically contradictory or undetermined, M_C must be modified.

This cycle of selection, simulation, evaluation, and modification constitutes an inherently combinatorial problem; it is solved when the physically correct behavior descriptions according to the points 3, 4, 5 and $\mathcal{B}_{Hyd}$ are determined.

Note that this model synthesis problem is not treated explicitly in literature on the subject. Existing simulation tools leave the problem to the user who has to set up the correct equations and conditions respectively. The next subsection exemplarily illustrates this statement.

Existing Tools

We found special-purpose and standard tools that support the design of hydraulic systems, e.g. MOSIHS [60], OHCS [57], MOBILE [34], or SIMULINK [53]. The majority of these tools has been developed to envision the dynamical behavior of (hydraulic) systems. Typically configurational aspects like different checking or optimization tasks are addressed only to a small part. Another characteristic of such simulation tools is that they hardly support model synthesis. Subsequently, the efficiency that comes up with an automated model synthesis is pointed out at the commonly used simulation tool SIMULINK.

Modeling a system with SIMULINK requires the specification of a directed behavior graph whose nodes are mathematical functions; in fact, the complex transformation of a hydraulic circuit diagram into a mathematical description is left to the user. Figure 5.5 shows a simple circuit consisting of a cylinder and two hydraulic resistances. Its SIMULINK-counterpart is depicted in figure 5.6.

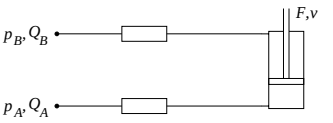


Figure 5.5: *Hydraulic specification of a simple circuit*

Remarks. Note that only simple stationary dependencies are modeled in figure 5.6. Also note that if a user wanted to investigate the same circuit at

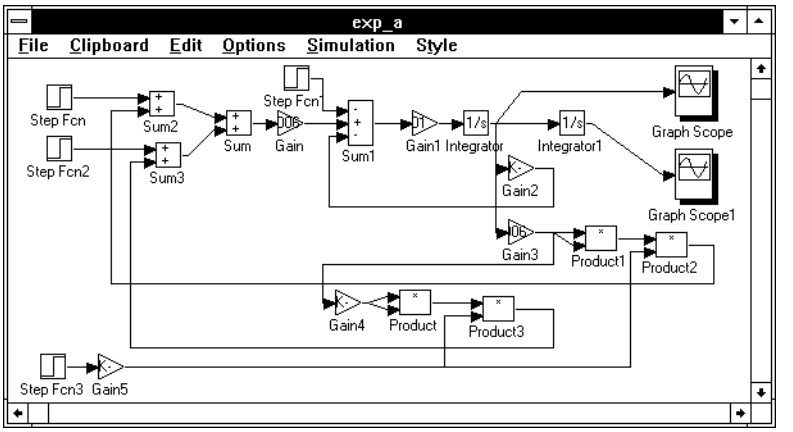


Figure 5.6: *Specification of the example in* SIMULINK

some deeper level, he would have to start from scratch with the analysis process.

While the *circuit* diagram can be understood and created by every engineer, a description level similar to that in figure 5.6 requires a hydraulic specialist. Certainly, the building blocks of SIMULINK are flexible, but they are too simple to noticeably reduce the analysis step's complexity.

Discussion

An automated analysis step is a necessary condition for all kinds of configuration support such as automated checking, parameterization, or optimization of hydraulic systems. The difficulty of the analysis step's automation comes up in the following points:

1. *Arbitrary Structures.* Hydraulic circuit analysis needs the processing of arbitrary structures and thus, some kind of model synthesis: determination of the components' description level, selection of local behavior descriptions, and composition of the local descriptions to a global behavior model. By contrast, if all structures of hydraulic systems were already known, the corresponding global models could be precalculated.

2. *Definable Component Behavior.* The behavior of all existing and future

components developed cannot be anticipated. Also, there is disagreement of how hydraulic components behave with respect to particular physical details. Thus, not only a hydraulic system's structure but also the components' descriptions must be user-definable to a point.

3. *Heterogeneous Constraints.* Behavior descriptions, user demands, necessary physical knowledge, heuristic design knowledge, etc. form a set of heterogeneous constraints that is comprised of several types of numerical and symbolic relations. These constraints, enclosing the dependencies between different types of constraints of course, must be both exactly formulated and processed.

5.3 Modeling Hydraulic Systems

In order to automate parts of the formerly described configuration procedure, we need a modeling concept for hydraulic systems. This section presents a domain-oriented component model for hydraulic systems that, in particular, addresses the previously discussed aspects: modeling of arbitrary hydraulic structures, definable behavior at the *component level*, and coupling of heterogeneous constraints. Using this model, we are able to precisely define the hydraulic checking and parameterization problem.

A Component Model for Hydraulic Configuration

The "classical" approaches to the description of technical systems are discussed by deKleer and Brown [16], Kippe [35], Kuipers [40], Struß [70], or Voss [79]. Struß's approach is similar to that of Kippe's COMMODEL system. Both approaches are derivatives of deKleers's and Brown's modeling ideas that are largely founded on the locality-principle and the no-function-structure-principle[2] [16].

The approaches mentioned have the following concepts in common: (*i*) They distinguish between components that realize the actual behavior and, for connection purposes, linking-components without behavior. (*ii*) The properties of the medium transported (e.g. electric current or oil viscosity) are modeled as an integral part of the components. Bound up with these concepts are some disadvantages concerning the structure and

[2] The no-function-structure-principle is introduced in section 4.1, page 76.

behavior definition of technical systems [28]. The approach now introduced is more flexible.

The modeling idea is grounded on the distinction of objects, gates, unifiers, and the types of information transported. The unifiers serve as sources or sinks of the information types in a technical system; the objects are characterized by their behavior constraints and their gates. Each gate is connected to exactly one unifier. Note that objects will have access to the information of those unifiers connected to their gates. Also note that objects which share the same unifiers share the unifiers' information too.

Objects, gates, and unifiers define a system's topology and its possible flows of information. Figure 5.7 illustrates these dependencies, *Definition 5.1* formalizes them.

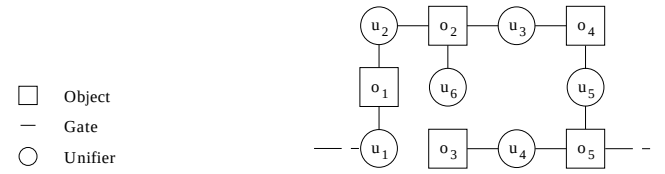


Figure 5.7: *Building block abstraction of a technical system*

Definition 5.1 (Building Block Model). A building block model is a tuple $\langle O, M, U, \gamma, \delta \rangle$ whose elements are defined as follows.

- O is an arbitrary finite set. It is called the object set.
- M is an arbitrary finite set. It is called the set of *information types*.
- U is a finite multiset of *unifiers*. Each $u \in U$ denotes an arbitrary subset of M, i.e., $u \subseteq M$, $u \in U$.
- $\gamma : O \to \mathbf{N}$ is a function and specifies for each object $o \in O$ the total number of gates.
- $\delta : O \times \mathbf{N} \to U$ is a partially defined function. $\delta(o, n)$, $o \in O$, $n \in \mathbf{N}$ specifies the unifier of object o at gate n and is defined for $n \leq \gamma(o)$ only.

Using these concepts, a hydraulic system is modeled as follows.

1. Each component of the system (valve, cylinder, pipe, etc.) is associated one-to-one with an element in O. $\gamma(o)$ defines a component's number of gates; e.g., if o is associated with a pipe then $\gamma(o) = 2$.
2. All physical quantities that have no manifestation within a component (pressure, flow, force, velocity, etc.) form the set M of information types.
3. δ is defined implicitly by the system's topology: For each link between two components in the hydraulic system a unifier with an appropriate subset of M is introduced. Example: If in the real system a certain pipe is connected with a particular valve fitting, within the building block modeling both the pipe's and the valve's fitting will share the same unifier u; u should then provide the information types "flow" and "pressure".
4. Each object is described by a set of behavior constraints that models the behavior of its associated component. An object's behavior constraints can refer only to those information types (e.g. a pressure or a flow) the object has access to via its gates. Note that same information types in different unifiers will establish different constraint parameters.

Figure 5.8 shows an example.

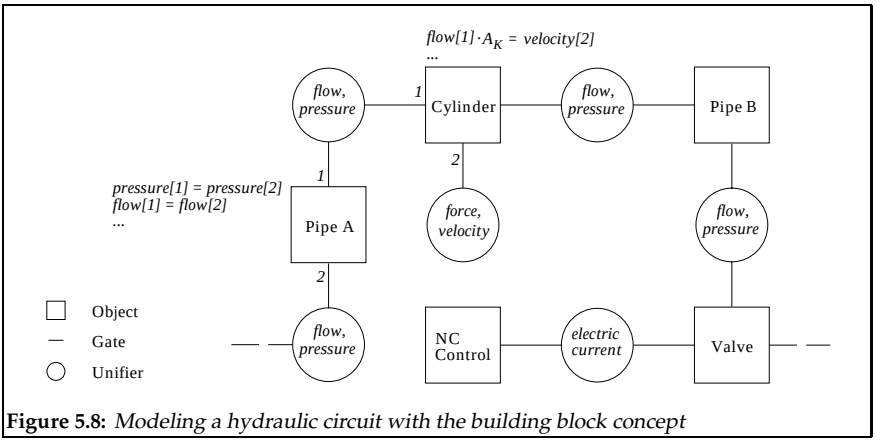


Figure 5.8: *Modeling a hydraulic circuit with the building block concept*

Until now we left open how behavior constraints for the objects in O can be formulated. We now outline the key concepts of a language for behavior constraints that is tailored to the recently defined building block model.

- Relations, defined over numerical and symbolic parameters, are the basic elements of our constraint language. A relation may be defined as follows. (*i*) explicitly, in the form of a finite set of tuples; (*ii*) implicitly, in the form of a boolean statement or an equation, which is composed of numerical or symbolic expressions.

  Whereas boolean statements are treated as tests, the semantics of equations is also handled in a deductional manner. If an unknown parameter constitutes one side of an equation while the other side can be evaluated, the "="-sign will produce a parameter assignment. If both sides of an equation can be evaluated, the "="-sign will produce a boolean test. In all other cases an equation may be transformed according to algebraic or some other rules.

- A behavior constraint is usually associated with a particular object o. The parameters with which the constraint is defined refer to the information types at the gates of o and to the internal properties of o. The exact assignment of a parameter is indicated by its tag, which is either a gate specifier or the key word "self". E.g., the following algebraic constraint defines the pressure drop of a valve between gate 1 and gate 2 due to Bernoulli:

$$pressure[1] - pressure[2] = R_H[\text{SELF}] \cdot flow[1]^2$$

- To each parameter a domain is defined. This domain is either an interval $v \subseteq \mathbf{R}$ or a finite set itemizing each single value allowed. Examples:

$$\begin{aligned} v_{valve\_resistance} &= [0.01,\ 0.2], \\ v_{valve\_position} &= \{\text{crossed, blocked, parallel}\} \end{aligned}$$

  The semantics of the latter is that, aside from *Unknown*, exactly one element of the specified set is admissible for the parameter $valve\_position$.

- Parts of an object's behavior description need to be activated or deactivated—e.g. to imitate the different states of a hydraulic component. A universal concept to represent such *model selection constraints* is given with rules. Thus, we allow behavior constraints to be embedded within rules. The condition part of a rule is a boolean statement composed of numerical and symbolic expressions; it specifies the conditions under which a behavior alternative is valid. The conclusion part defines the behavior constraints of the alternative and possibly

additional rules. The following example shows a simplified pressure relief valve where the state "activated" is associated with a specific behavior:

IF $state[\text{SELF}] = \text{activated}$
THEN $Q[1] = -Q[2]$ AND $p[1] - p[2] = R_H[\text{SELF}] \cdot \sqrt{|Q[1]|}$
ELSE $Q[1] = 0$ AND $Q[2] = 0$

The constraints in the conclusion part will be considered during behavior processing only if the condition part is evaluated as *True*.

- Behavior constraints can be supplied with metaknowledge that specifies hints to be exploited during the behavior processing. This knowledge may indicate a constraint's description level, whether a constraint contains stationary or dynamic dependencies, or some other processing directive. Table 5.1 gives some examples.

Constraint	**Metaknowledge**
$p[1] - p[2] = R_H[\text{SELF}] \cdot Q[1]^2$	level-0 description, stationary behavior
$Q[1] + Q[2] + Q[3] = 0$	level-independent description, process locally

Table 5.1: *Metaknowledge specifications for behavior constraints*

Remarks. The building block model of *Definition 5.1* along with the outlined behavior language make up our component model for the configuration of hydraulic systems.

Instances of Π^{c}_{M3} in Hydraulics

Remember the behavior-based configuration problems under model M3, defined in section 3.3. Obviously, the component model above is a *device-oriented interpretation* of model M3. It represents the *engineer's* view of configuration in the sense that it introduces domain concepts, defines a semantics, and is oriented by knowledge acquisition purposes.

Hence, hydraulic analyzing, checking, and parameterization problems are particular instances of the generic behavior-based checking problem Π^{c}_{M3}, defined on page 63.

More precisely—a checking problem under model M3 (Π^{c}_{M3}) is defined by the tuple $\langle O, F, V, B, T, D, S \rangle$. These elements are related to our component model as follows.

- O is equivalent to the objects of the building block model and denotes the hydraulic components.
- All parameters in the objects' behavior constraints form the set F of functionalities. Each element $f \in F$ denotes a (possibly constant) function in the parameter "time".
- V is comprised of all functionalities' value sets.
- The set B of behavior constraints is comprised of the behavior descriptions of the hydraulic components at some definite level.
- T is comprised of all functionalities' tests.
- D is the set of demands on the hydraulic system, stated by a customer.
- The pairs in S correspond to the elements in the unifiers $u, u \in U$ and define the structure of a hydraulic system.

Remarks. The checking problem Π^{c}_{M3} in hydraulics defines a hydraulic system and a set of demands. Checking this system means both determining a set Q of tuples (f, x), $f \in F$, $x \in v_f$, which are consistent with all behavior constraints, and verifying if none of the demands is violated.[3] These jobs are done during the hydraulic analysis and evaluation step respectively.

Tackling a *parameterization* problem means to solve the checking problem Π^{c}_{M3} for a hydraulic system that is underspecified within one constraint or another.

5.4 Solving Π^{c}_{M3} in Hydraulics

This section introduces the necessary concepts to solve instances of Π^{c}_{M3} in hydraulics. Figure 5.9 shows the steps that are performed when analyzing, checking, or parameterizing a hydraulic system.

Remarks. Given is a hydraulic system C for which, in the first step, an adequate description level has to be determined. Based on this modeling of C and the set D of demands, an instance of Π^{c}_{M3} can be stated. Within the subsequent synthesis and simulation steps the set Q of functionality-value pairs, which explicitly envisions the behavior of C, is inferred. Q constitutes a solution of Π^{c}_{M3} if both all behavior constraints and all demands can be fulfilled.

[3] A precise definition is given in section 3.3, page 63.

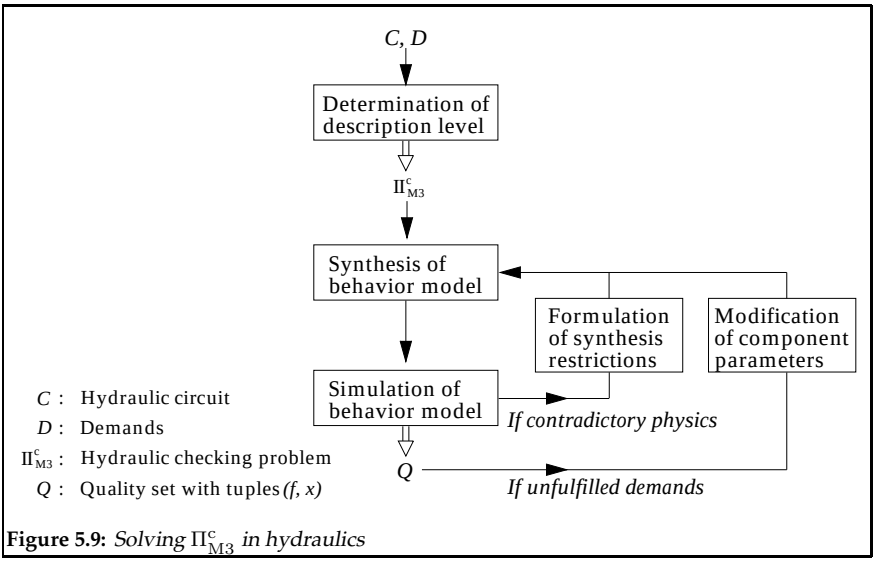


Figure 5.9: *Solving* Π^{c}_{M3} *in hydraulics*

Determination of the Description Level

The components of a hydraulic system can be described at different levels of detail. To avoid superfluous computational effort a component's behavior description should be as simple as possible—but still sufficiently detailed to model all necessary relations.

To get a grip on this model formulation problem we developed, along with Lemmen [42] and Suermann [71], the concept of variable hydraulic modeling levels. In particular, Lemmen defined a model formulation scheme that, dependent on a component class, distinguishes up to five predefined description levels. Given a hydraulic system C, the adequate description level can be inferred for each component by means of a knowledge-based decision procedure that is based on the natural frequencies and gain factors of the components in C, a user's simulation intention, and the topology of C. A detailed description of the model formulation scheme and the decision procedure can be found in [71] and [43].

Since the determination of the adequate description level is a high degree hydraulic engineering problem, it shall not be extended here. Henceforth, we assume all components of a hydraulic system described at some definite level; analyzing, checking, or parameterizing such a system is an instance of Π^{c}_{M3}.

Synthesis and Simulation of Behavior Models

Given is an instance of Π^{c}_{M3} in hydraulics. Π^{c}_{M3} defines a constraint satisfaction problem consisting of numerical and symbolic relations. Actually, Π^{c}_{M3} cannot be solved by a universal "constraint satisfaction algorithm" but needs the interplay of several computation methods along with a global control mechanism that separates and triggers subjobs, maintains alternatives, and controls model synthesis.

As argued before model synthesis is not a deterministic procedure here. There exist choice points where the valid component model must be selected, depending on the actual input values, parameter alternatives, or physical regularities. For each component $o \in O$, O defined by Π^{c}_{M3}, let $M_o = \{m_{o_1}, m_{o_2}, \ldots, m_{o_k}\}$ be comprised of the k behavior alternatives of o. If a component o has a locally unique model, say, a pipe for instance, $|M_o| = 1$. Let $\mathcal{M}_C$ be the Cartesian product of the $M_o, o \in O$. Obviously, $\mathcal{M}_C$ comprises the possible global models of the hydraulic system C defined by Π^{c}_{M3}, and thus, $\mathcal{M}_C$ defines the total synthesis search space.

Before all physical parameters of a hydraulic system C can be computed, a physically consistent model $M_C \in \mathcal{M}_C$ has to be determined. Conversely, whether a behavior model M_C is physically consistent can solely be verified via simulation. To illustrate the search for a consistent behavior model in $\mathcal{M}_C$, it is useful to think of the components' behavior constraints being divided into model selection constraints (cf. page 101) and behavior constraints. The search procedure can be outlined as a cycle containing the following inference steps:

1. *Component Selection.* Select a component with undetermined behavior.
2. *Model Selection.* Select a behavior alternative for this component.
3. *Synthesis.* Identify and evaluate active model selection constraints, and synthesize the emergent behavior model.
4. *Simulation.* Simulate the synthesized behavior model by evaluating the behavior constraints.
5. *Modification.* In the case of physical inconsistencies or unfulfilled demands, formulate synthesis restrictions and trace back to a choice point.

Figure 5.10 illustrates the search process graphically.

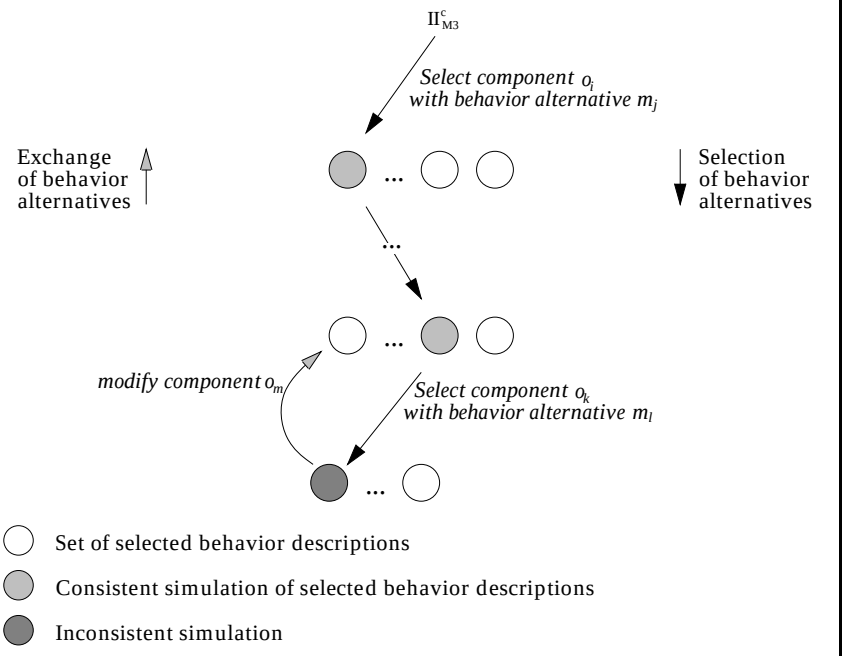


Figure 5.10: *Exploring the synthesis search space $\mathcal{M}_C$*

The search comes to an end if either a global, consistent behavior model is found that fulfills all demands or no further choice point exists.

Remarks. Different components constrain the model synthesis step in a different manner. Hence, the order by which undetermined components are processed may play a crucial role.

The evaluation of behavior constraints, mentioned in the simulation step of the above search procedure, is a demanding problem. The components' functional constraints must be parsed, symbolic relations must be separated from numerical relations, equations have to be transformed, equation systems have to be formulated in some normal form, rules have to be processed, etc. Moreover, through the variety of constraint types and constraint dependencies, the backtracking mentioned in the model modification step will become a sophisticated job, too. In this text we do not go into constraint processing details but merely give an overview of the inference methods necessary to solve Π^c_{M3}.

Necessary Inference Methods

The following numerical subjobs must be performed when processing Π^{c}_{M3} in hydraulics:

- *Solving Linear Equation Systems.* Input is a linear equation system in the matrix form $A \cdot x + b = 0$ where A is regular. Output is the vector x that solves the equation system.
- *Solving Non-Linear Equation Systems.* Input is a continuous, non-linear function $f : \mathbf{R}^n \rightarrow \mathbf{R}^n$ and a possibly empty set of restrictions constraining the value ranges of the solution in $m \leq n$ dimensions. Output is a vector x that fulfills both the equation $f(x) = 0$ and all restrictions.
- *Solving Initial Value Problems.* Input is a system of ordinary differential equations $y' = f(x, y)$ where $f : G \rightarrow \mathbf{R}^n$ is a continuous function defined on $G = [a, b] \times \mathbf{R}^n$, and an initial condition $\alpha \in \mathbf{R}^n$. Output is a function $y = u(x)$ that solves the differential equation system and fulfills the initial condition $u(a) = \alpha$.

Algorithms that process the itemized problems are given in [63]. Aside from methods dealing with numerical problems, we need the following methods for symbolic value processing:

- *Local Value Propagation.* Sources for input are the sets F (functionalities), V (value domains), B (behavior constraints) as described on page 102, and an initial value assignment X. Each behavior constraint $b \in B_o$ defines a relation on $v_{f_{b_1}} \times v_{f_{b_2}} \times \ldots \times v_{f_{b_k}}$ where $f_{b_i} \in F$, $\{b_1, b_2, \ldots, b_k\} \subseteq \{1, 2, \ldots, n\}$ and $n = |F|$. X defines a tuple $(x_1, x_2, \ldots, x_n)$, $x_i \in v_i \cup \{\text{unknown}\}$, $v_i \in V$ where some or all elements may be unknown.

  With local value propagation we designate a deduction mechanism that exploits only one constraint definition in B at the same time computing values for the unknowns in X. An example for such a mechanism is the following:

  1. Select a behavior constraint $b \in B$ where all parameters except one are known. If no such constraint exists, local propagation comes to an end. Otherwise, without loss of generality, we assume x_j to be the unknown parameter.
  2. Determine a value for $x_j \in v_j$ such that the relation defined by b is fulfilled and all behavior constraints also defined on v_j stay

consistent. If no such x_j can be determined but earlier choice points exist, invoke backtracking. Otherwise, local propagation terminates and $\langle B, X \rangle$ is called inconsistent.

3. Continue with 1.

Remarks. The algorithm computes a globally consistent solution if one exists. Nevertheless, dependent on the constraints and the domains they are defined upon, several approaches for local propagation and consistency definitions exist, which will not be elaborated on here. Constraints may be defined *extensionally*, by a complete enumeration of the relations' tuples, or *implicitly* by a set of functions. In the latter case the power and flexibility of local propagation depends on the algorithms that realize the evaluation of the functions. Note that the principle of local constraint evaluation can be applied to any type of relation.

- *Rule Inference.* Rule inference in the form of forward chaining is required to evaluate selection constraints, functional constraints, and demand constraints. Note that not only plain symbolic relations but also dependencies between different sets of behavior constraints must be handled.
- *Algebraic Transformation.* Algebraic transformation capabilities are necessary to process the implicitly defined constraints and to generate the input forms for the numerical computation methods.

5.5 Preprocessing of Stationary Behavior

Processing stationary behavior constraints requires methods capable of coping with equation systems, local propagation, and algebraic transformation. The approach presented now substitutes the computation of *local* connections with that of *global* connections. The key idea is to preprocess the topology of a hydraulic system; global structure information is "compiled" into local behavior constraints, which are added to the original constraints. The resulting component descriptions can be processed by local propagation, which is more efficient than the solution of equation systems necessary for the original descriptions.

Motivation

A large part of a hydraulic system can be considered as a network consisting of resistances, sources, and sinks. Pipes and valves establish the hydraulic resistances (R), while pumps, tanks, and—of course—cylinders act as sources (s) and sinks (t) respectively. Figure 5.11 depicts an example.

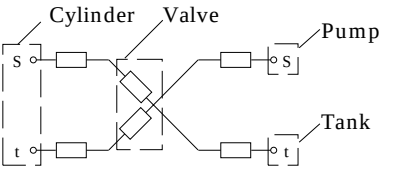


Figure 5.11: *An abstract view on a simple circuit*

A central task of the hydraulic checking problem Π^{c}_{M3} is the computation of a total flow distribution and of all pressure drops for such a network. As a difference to electrical resistors, hydraulic resistances define non-linear connections—more exactly: in most cases the potential difference $p_1 - p_2$ at a hydraulic resistance is proportional to the quadratic flow:

$$p_1 - p_2 = R_h \cdot sign(Q) \cdot Q^2$$

The previous section introduced local propagation as a method that evaluates *single* behavior constraints of components in order to compute unknown functionalities. This method is very efficient since no global relations are exploited but will terminate if no constraint can be found where all parameters except one are known. E.g., if two hydraulic resistances are connected in parallel, the computation of the flow distribution and the pressure drops often requires the solution of a non-linear equation system. Figure 5.12 gives examples of structures that may be part of a hydraulic resistance network.

The edges denote hydraulic resistances while the points stand for components where all incident resistances are connected (unified). Such components might be T-connections or other connections that establish an area of equal potential. Within a context-free modeling the only behavior constraint of a connection is the *continuity condition*:

$$Q_1 + Q_2 + \ldots + Q_n = 0$$

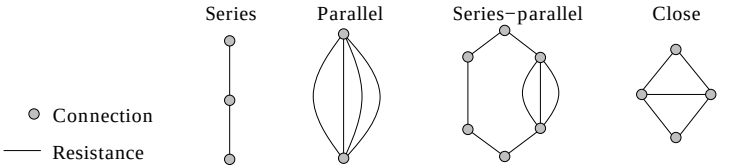


Figure 5.12: *Possible substructures in a resistance network*

Preprocessing means the introduction of additional constraints, called *proportion constraints* here. Proportion constraints are computed from the resistances between a source and a sink. They are installed in the connections and provide information about how the flow Q is distributed. Example:

$$Q_1 = c_1 \cdot Q_e \ \wedge \ Q_2 = c_2 \cdot Q_e \ \wedge \ Q_1 + Q_2 = Q_e, \ c_1, c_2 \in \mathbf{R}^+$$

Q_e is introduced as a new variable and denotes the entire flow at a connection; c_1 and c_2 determine how this flow is distributed; Q_e is substituted for $Q_1 + Q_2$ in the original continuity condition. Obviously, with the aid of proportion constraints, local value propagation will be sufficient to distribute the flow at the connections and to compute all related pressure drops, if at a network source (e.g. at a pump) a flow is given. Of course, proportion constraints violate the no-function-in-structure-principle since they are not context-free. They exploit global information about resistances and the network's structure.

The following subsections show how those structures of a network that cannot be tackled by local propagation are found and how the related proportion constraints are computed.

Finding All Relevant Substructures

In principle, proportion constraints could be computed for a network's global structure in a single computation step. However, for flexibility reasons it is much more appropriate to cut a network into subnetworks and to compute proportion constraints locally for each substructure. The smaller a separated subnetwork is the more flexible the computed proportion constraints can be used.

Proportion constraints can be computed only for subnetworks of a particular structure. We now define these structures and present methods to

identify them as parts of a global network. We use the following definitions of graph theory in the standard way:

1. A *multigraph* G is a triple $\langle V, E, g\rangle$ where $V, E \neq \emptyset$ are finite sets[4], $V \cap E = \emptyset$, and $g : E \to 2^V$ is a mapping with $2^V = \{U \mid U \subseteq V,\ |U| = 2\}$. V is called the set of points, E is called the set of edges, and g is called the incidence map.

2. A graph $H = \langle V_H, E_H, g_H\rangle$ will be called *subgraph* of $G = \langle V, E, g\rangle$, if $V_H \subseteq V$, $E_H \subseteq E$, and g_H is the restriction of g to E_H. A subgraph will be called an *induced subgraph* on V_H, if $E_H \subseteq E$ contains exactly those edges incident to the points in V_H. For $T \subset V$, $G \setminus T$ denotes the subgraph induced on $V \setminus T$.

3. A tuple $(e_1, \ldots, e_n)$ will be called a walk from v_0 to v_n, if $g(e_i) = \{v_{i-1}, v_i\}$, $v_i \in V$, $i = 1, \ldots, n$. G will be called *connected*, if for each two points $v_i, v_j \in V$ there is a walk from v_i to v_j. If G is connected and $G \setminus v$ is not connected, v establishes an *articulation point*.

4. $\kappa(G)$ is called the *connectivity* of G and is defined as follows: $\kappa(G) = min\{|T| : T \subset V$ and $G \setminus T$ is not connected$\}$. G is called *k-connected*, if $\kappa(G) \geq k$.

Figure 5.13 illustrates the definitions.

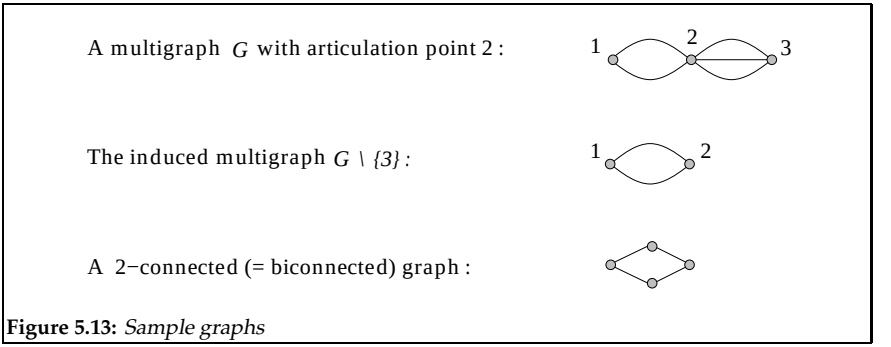


Figure 5.13: *Sample graphs*

The edges of a multigraph stand for the hydraulic resistances such as pipes and valves; the points connect the resistors and represent areas of equal potential. We need multigraphs instead of graphs here since components of a hydraulic system may be connected in parallel. Moreover,

[4]We restrict ourselves to finite graphs here.

we will restrict ourselves to particular multigraphs, the so-called resistance networks.

Definition 5.2 (Resistance Network). A *resistance network* N is a tuple $\langle G, \rho, s, t\rangle$ where $G = \langle V, E, g\rangle$ is a connected multigraph, $\rho : E \to \mathbf{R}^+$ is a mapping, and $s, t \in V$ are two points. $\rho(e), e \in E$ is called the resistance of e; s, t are called the source and the sink respectively.

Remarks. In contrast to "flow networks" or "capacitated networks" that define capacity values for the edges of a graph, ρ defines resistance values in the physical sense of hydraulics. Resistance networks and capacitated networks are used in connection with *flows*. We refrain from a precise definition of flows but shall point out three important characteristics:

1. A flow defines both a flow value and a flow direction on the edges of a resistance network.
2. To each point $v \in V \setminus \{s, t\}$ applies the continuity condition: The total of all input flows equals the total of all output flows.
3. All incident edges of s (of t) establish output (input) flows only. The total of s's output flows equals the total of t's input flows. The latter is a direct consequence of 2.

Note that a flow distribution in a physical sense has nothing to do with a flow mapping under a maximum-capacity interpretation of ρ. In the former case, a flow distribution depends on the resistance *ratios* of *all* edges. In the latter case, all edges' capacity values must be considered as well, but the capacities are independent of their actual context in the graph: Every edge can be checked *locally*, whether its related flow violates the edge's capacity.

We are interested in those parts of a network[5] whose flow distribution can be computed independently of the rest. For obvious, physical reasons each subnetwork whose resistance behavior can be reproduced by a substitute resistance—i.e., by a single edge—constitutes such a part. The following definitions will be useful.

Definition 5.3 (Independent Subnetwork). Let be $N = \langle G, \rho, s, t\rangle$ a network, H a subgraph of G induced on $V_H \subset V$ with $|V_H| > 2$, and ρ_H the restriction of ρ to E_H. Then, a network $N_H = \langle H, \rho_H, s_H, t_H\rangle$ will be called an *independent subnetwork* of N, if the following conditions hold:

[5]Henceforth, we shall refer to a "resistance network" simply as "network".

(*i*) Every walk from s (from t) to a point in H contains either s_H or t_H.

(*ii*) Every walk from a point in $G \setminus V_H$ to a point in H contains either s_H or t_H.

An independent subnetwork $N_{H_1} = \langle H_1, \rho_{H_1}, s_{H_1}, t_{H_1} \rangle$ will be called *minimum independent subnetwork* of N, if there exists no independent subnetwork $N_{H_2} = \langle H_2, \rho_{H_2}, s_{H_2}, t_{H_2} \rangle$ where H_2 is induced on a proper subset of V_{H_1}.

Remarks. Since any two adjacent points establish a source and a sink respectively, we claim H to be defined on more than two points. Condition (*i*) guarantees some kind of "dipole character" of H. Condition (*ii*) ensures that there are exactly two connections for a substitute resistance. Note that this is not implied by (*i*) . Also note that proportion constraints computed for minimum independent subnetworks provide the maximum flexibility in the course of local propagation.

Before we turn our attention to the question as to how minimum independent subnetworks are detected, we need a further definition:

Definition 5.4 (Triconnected Component, Biconnecting Points). Let $G = \langle V, E, g \rangle$ be a multigraph, $|V| > 2$, $\kappa(G) = 2$, and $\{v_i, v_j\} \in V$ two points such that $H = G \setminus \{v_i, v_j\}$ is not connected. Moreover, let be H_1 a resulting connected component of H and $V_{H_1} \subset V$ the set of points inducing H_1.

If no two points $\{w_i, w_j\} \in V$ can be found such that a resulting connected component of $G \setminus \{w_i, w_j\}$ is induced on a proper subset of V_{H_1}, then the graph H_2 induced on $V_{H_1} \cup \{v_i, v_j\}$ shall be called a *triconnected component* of G. The points v_i, v_j shall be called the *biconnecting points* of H_2 related to G.

Figure 5.14 gives an example.

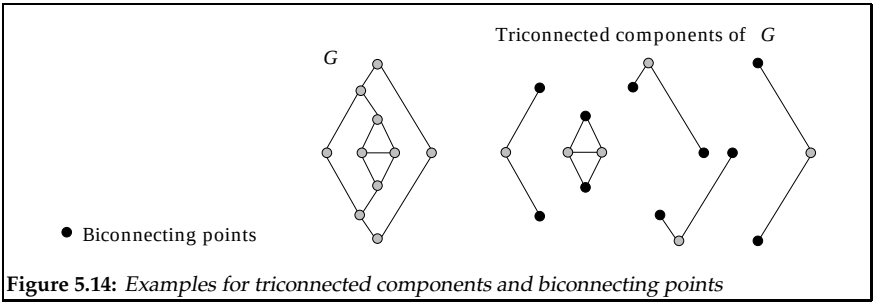


Figure 5.14: *Examples for triconnected components and biconnecting points*

Lemma 5.5 (Independent Subnetwork). Let be $N = \langle G, \rho, s, t\rangle$ a network and $|V| > 2$.

1. If $\kappa(G) = 1$, let $G_1, \ldots, G_n$ denote the biconnected components of G. Then, if $|V_i| > 2$, there exist two points $s_i, t_i \in V_i$ such that $N = \langle G_i, \rho_i, s_i, t_i\rangle$ is an independent subnetwork of N, $i \in \{1, \ldots, n\}$.
2. If $\kappa(G) = 2$, let $G_1, \ldots, G_n$ denote the triconnected components of G, and let s_i, t_i denote the biconnecting points of G_i related to G. Then, $N_i = \langle G_i, \rho_i, s_i, t_i\rangle$ will be a minimum independent subnetwork of N, if $V_i \setminus \{s_i, t_i\} \cap \{s, t\} = \emptyset$, $i \in \{1, \ldots, n\}$.
3. If $\kappa(G) > 2$, N will not contain an independent subnetwork.

Proof. (1) Let G_i be a biconnected component. We have to investigate two cases: Either V_i contains one or two articulation points of G. Case A. v is the only articulation point in V_i. Then, either s or t must be in V_i. If $s \in V_i$, set $s_i = s$ and $t_i = v$; if $t \in V_i$, set $s_i = v$ and $t_i = t$. Case B. Let v_i, v_j be the articulation points in V_i. Then, $s_i = v_i$ and $t_i = v_j$. We can check easily that $\langle G_i, \rho_i, s_i, t_i\rangle$ fulfills the definition of an independent subnetwork.

(2) Let G_i be a triconnected component. Part one (independent subnetwork). According to the definition of triconnected components, a walk from any point in $G \setminus G_i$ to a point in G_i contains either s_i or t_i (this fact is a consequence of the biconnectivity of G). Thus, condition (*ii*) of the independent subnetwork definition is fulfilled. Also, condition (*ii*) together with the restriction that $V_i \setminus \{s_i, t_i\} \cap \{s, t\} = \emptyset$ fulfills part (*i*) of the independent subnetwork definition. Part two (minimality). We assume N_i not to be minimum. Then, there exists an independent subnetwork $H = \langle H, \rho_H, s_H, t_H\rangle$ with $H = G_i \setminus T, T \subset V_i$. I.e., G_i cannot be a triconnected component. This contradicts the condition.

(3) This assertion is obvious since $\kappa(G) \leq 2$ is a direct consequence of the independent subnetwork definition. ◇

Remarks. The independent subnetworks found for $\kappa(G) = 1$ are not necessarily minimum. They might be processed further until the connectivity of their related graph is > 1. The complexity for the computation of all minimum independent subnetworks of a network N with a graph $G = \langle V, E, g\rangle$ can be estimated with $O(|V| \cdot |E|)$. We outline only the proof idea. All triconnected components of a graph G with $\kappa(G) > 1$ can be found as follows. A point $v \in V$ is selected and the graph H, induced on $V \setminus \{v\}$, is investigated with regard to its biconnected components. Obviously, the biconnected components of H are triconnected components of G, if they

cannot be extended by v, the point initially removed. For an articulation point w, found during the biconnected component search, the graph induced on $V \setminus \{w\}$ does not need to be investigated. In the worst case $\kappa(G) > 2$ and for each $v \in V$, the induced graph is investigated with regard to biconnected components. According to Tarjan, the biconnected components of a graph G can be computed in $O(|E|)$ [75].

In practice the identification of independent subnetworks will be less complex. Often there exist particular structures, which can be detected easily: the so-called series-parallel networks. Also, for physical reasons the computation of substitute resistances for series-parallel networks is much easier than for networks relying on close-connected graphs. Hoffmann provides a more detailed discussion of this topic [28].

Installing Proportion Constraints

Subsequently, we describe the general installation procedure of proportion constraints in a network $N = \langle G, \rho, s, t\rangle$. Initially, $N_2 = \langle G_2, \rho_2, s_2, t_2\rangle$ is a copy of N.

1. Identify an independent subnetwork $N_H = \langle H, \rho_H, s_H, t_H\rangle$ of N_2 .
2. Compute the flow distribution of N_H.
3. Install the proportion constraints in the original graph G.
4. Condensate G_2, i.e., replace H by the new edge e_H and redefine ρ_2.
5. Continue with 1 until s_2 and t_2 are the only points of G_2.

Remarks. Before close connected subgraphs are treated, all series-parallel structures should be searched and replaced. Note that due to the condensation of G_2, new series-parallel and close connected structures may emerge. Figure 5.15 illustrates the process.

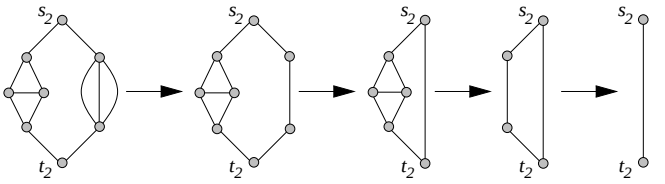


Figure 5.15: *Condensation of G_2 during constraint computation*

The computation of substitute resistances and flow distributions for N_H with graph $H = \langle V_H, E_H, g_H \rangle$ is based on the continuity conditions in V_H and the pressure drop equations instantiated for each $e \in E_H$:

- *H is Series Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 are restricted to E_2 where $\rho_2(e_H) := \sum \rho_H(e_i)$, $e_i \in E_H$.

- *H is Parallel Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 is restricted to E_2 where $\rho_2(e_H)$ can be computed from the following equation:

  $$\sqrt[2]{\frac{1}{\rho_2(e_H)}} = \sum \sqrt[2]{\frac{1}{\rho_H(e_i)}},\ e_i \in E_H$$

  In s_H and t_H, the following proportion constraints are installed:

  $$Q_i := c_i \cdot Q_H,\ c_i := \sqrt[2]{\frac{\rho_2(e_H)}{\rho_H(e_i)}},\ e_i \in E_H, \text{ and } Q_H = \sum Q_i$$

- *H is Close Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 is restricted to E_2 where $\rho_2(e_H)$ can be computed from the following connection: The continuity conditions of all points $\in V_H \setminus \{s_H, t_H\}$ along with the pressure drop equations for each $e \in E_H$ and the equation $p_s - p_t = x$, $x \in \mathbf{R}^+$ form a non-linear equation system. Under the restriction that all pressure drops are positive, it can be shown that this equation system has a definite solution in the flows Q_i, $i = 1, \ldots, |E_H|$. Then, $Q_H := \sum Q_i$, e_i is incident to s_H (t_H), establishes the total flow through H. As a result, $\rho_2(e_H) := x \cdot Q_H^{-2}$. Now, a proportion constraint can be formulated for each flow variable $Q_i = c_i \cdot Q_H$, $e_i \in E_H$, where the c_i are computed from the solutions of the equation system $c_i := Q_i \cdot Q_H^{-1}$. These proportion constraints and the equation $Q_H := \sum Q_i$, e_i is incident to s_H (to t_H) are installed in s_H (in t_H).

Discussion

The above preprocessing approach focuses on optimization tasks where the structure of a complex hydraulic system is still defined but several alternative situations need to be investigated and evaluated. It will not be useful in the investigation of small hydraulic systems or for a single simulation.

Chapter 6

The [art]*deco* System[1]

[art]*deco* is a system that supports the configuration of hydraulic systems; it operationalizes a large part of the concepts presented in the former chapter. [art]*deco* solves several instances of Π^{c}_{M3}, that is, the analysis and the checking of hydraulic systems, and the parameterization of single component parameters [39], [44], [67].

However, solving instances of Π^{c}_{M3} is not enough to support a user in configuring hydraulic systems. When supporting configuration in hydraulics, aside from a knowledge processing task, there is also a *problem specification* and a *knowledge acquisition* task to be tackled.

By the term "problem specification" we denote the procedure of formulating an instance of Π^{c}_{M3} in hydraulics, or: How can a user specify his problem in an acceptable time?—Knowledge acquisition is of equal importance; configuration support in hydraulics will be useless if a user cannot integrate his individual knowledge, his experience in component modeling, or specifications of new components. Both aspects were taken into account when developing [art]*deco*. Besides efficient inference concepts, [art]*deco* realizes *graphic problem specification* and provides a language to model component behavior.

[1] The system itself, its philosophy, and its realization originated from the DFG research project No. Kl 529/3-1, where the institute MSRT, University of Duisburg, (Prof. Dr. H. Schwarz), and the institute Knowledge-based Systems, University of Paderborn, (Prof. Dr. H. Kleine Büning), were involved. The [art]*deco* system in its actual form has been developed by D. Curatolo, M. Hoffmann, and B. Stein. As an expert in hydraulic engineering R. Lemmen contributed to the development in an advisory capacity.

This chapter introduces the philosophy and some concepts of artdeco but does not engage in the details of realizational aspects.

6.1 Graphic Problem Specification

A configuration process that is grounded on behavior descriptions is usually so complex that its complete automation is not possible. In this case, the job of a configuration system is not to *solve* but rather to *support* the creative design process (cf. section 2.3, page 39, and section 5.1, page 93). When given such a behavior-based configuration problem, technical dependencies might be so complicated that problem specification, knowledge acquisition, and maintenance can solely be understood on a very abstract level, e.g., on the level of a technical drawing.

This is the situation when designing hydraulic systems, where the philosophy of artdeco comes into play:

The working document of the design process is the circuit diagram. Consequently, it would be fair to specify hydraulic checking problems at the same level of abstraction. Graphic symbols should be selected and connected to a circuit, but in contrast to a CAD system, aside from the drawing, a *functional model* of the hydraulic system should be generated as well.

artdeco realizes such graphic problem specification. While the circuit diagram of a system is drawn, a knowledge base containing all necessary physical connections is created. In a second step arbitrary sections of the hydraulic system can be checked concerning individual demands. I.e., the model composition/formulation process as well as complex physical dependencies are made transparent: Nearly all information obligatory for the checking and simulation process is derived from the technical drawing. Brought down to a simple formula, artdeco = CAD + behavioral semantics.

Specifying Hydraulic Problems with artdeco

There is always a gap between the semantics of a configuration problem on the one hand and the syntax for its specification on the other. artdeco addresses this situation by bridging the gap between the process of drawing and the process of investigating/simulating a technical system; it can be considered as a visual language to specify a particular class of technical problems.

Being in art*deco*'s application mode, a user selects hydraulic components from a component catalog and arranges them on the working area (cf. figure 6.1).

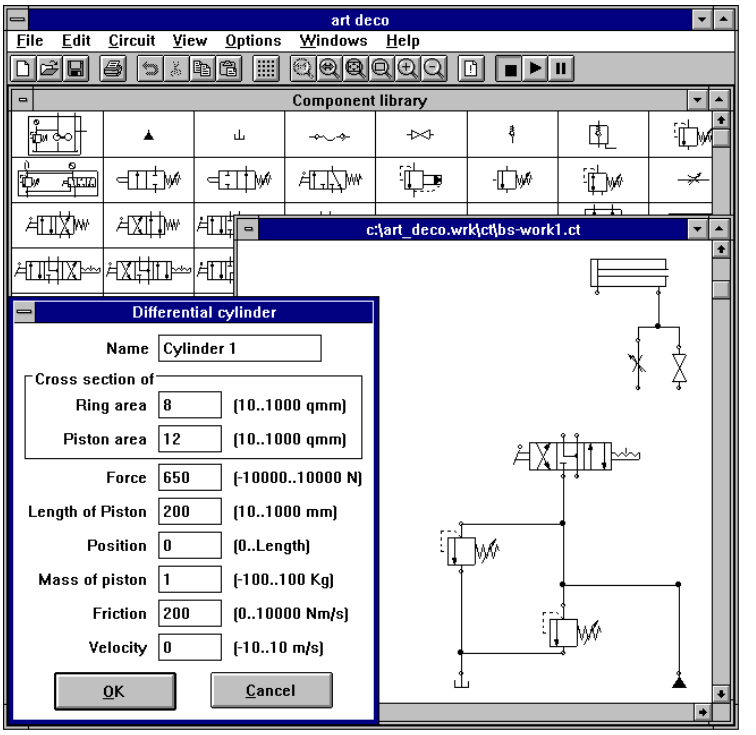


Figure 6.1: art*deco's application mode*

While drawing a line between two components' gates, the appropriate pipes are selected and instantiated. Among other things, it is checked whether the incident gates are of the same type. The necessary information concerning the topology is generated as well. Within the circuit diagram, all parameters of the hydraulic system can be predefined, changed, or supplied with alternative values.

After the inference process is invoked, art*deco*'s inference engine searches for a consistent parameter assignment. Figure 6.2 shows a circuit where such an assignment has been found.

The use of art*deco* as a prototype has shown that its philosophy of graphic

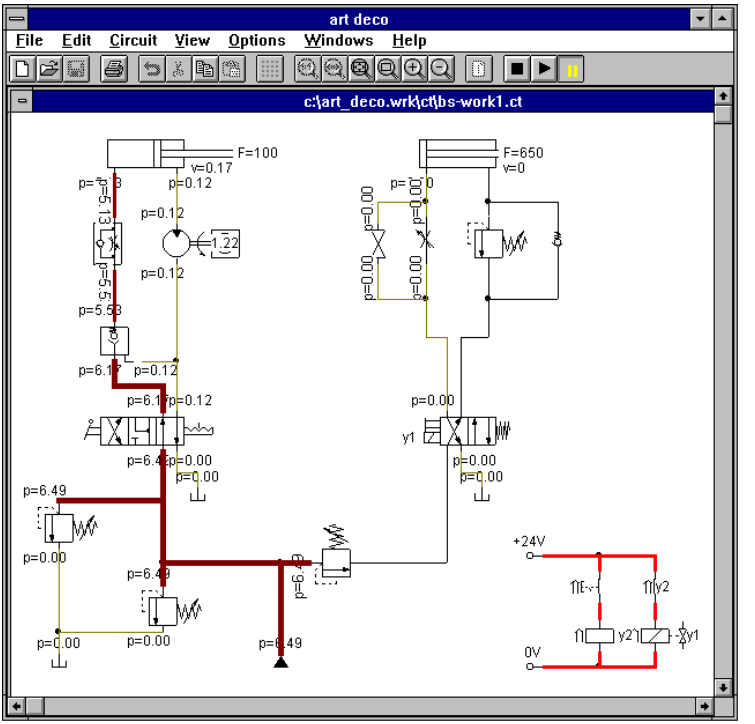


Figure 6.2: *art deco's application mode*

problem specification leads to a decisive reduction of the specification complexity. Perhaps, graphic problem specification as realized here is the only chance to efficiently support complex configuration tasks in hydraulics.

6.2 Inference

art deco takes the graphical description of a hydraulic circuit and investigates the system with respect to the following faults:

1. *syntactical* faults like open pipes
2. *geometrical* faults like wrong connections
3. *logical* faults like piston movements contradicting to valve positions

4. *dimensional* faults like pumps whose power range is exceeded

If none of these faults is found, all components' states are determined, and, along with any unknown velocities, forces, and component parameters, an entire distribution of the flow and the pressure is computed.

The subsequent paragraphs outline the underlying inference process.

Constraint Processing at First Glance

From the standpoint of problem specification and knowledge processing, we distinguish between the following constraint classes in *ardeco*:

- *Connection Constraints.* Connection constraints establish if and how two components may be connected. Processing these constraints means checking all connections' types and port sizes, the mechanical couplings, and for open pipes. Since this checking step is both the least demanding one and separable from other constraint processing jobs, it is always performed first.
- *Topological Constraints.* Topological constraints are given by the circuit diagram and define the physical structure of the hydraulic system. A correct realization of this structure is achieved as follows: All local component parameters are replaced by global variables, which in turn are unified according to the connection information. This unification step takes place before the functional constraints are processed.
- *Behavior Constraints.* Behavior constraints define the local behavior of components and are user-definable. They consist of relations defined over numerical and symbolic parameters. Parameters can be constrained through a domain; the constraints themselves can be supplied with metaknowledge.
- *Model Selection Constraints.* Model selection constraints are used to define different component states; they associate a state description with a particular behavior alternative. A behavior alternative comprises a set of behavior constraints.
- *Demand Constraints.* Demand constraints comprise internal and external restrictions. Internal demand constraints check for violations of universal behavior laws of hydraulics; external demand constraints model a user's demands and are specified in the form of rules and simple relations. In contrast to behavior constraints, the demand

constraints are propagated *destructively*. I.e., they are not exploited to derive new dependencies but rather to check them. To perform *early pruning* in the course of constraint processing, demand constraints are checked after each inference step.

While a user is drawing a circuit diagram, a building block model of the hydraulic system is constructed (cf. page 99). When the inference process is started, ARA-*deco* processes the connection constraints and the topological constraints as defined by the building block model and formulates an instance of Π^{c}_{M3}. This instance of Π^{c}_{M3} in turn is processed in a cycle of model synthesis and model simulation (cf. figure 6.3 and figure 5.9 on page 104 respectively).

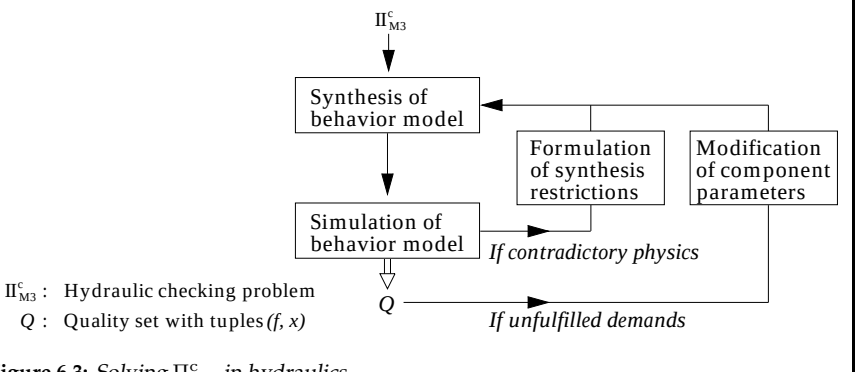


Figure 6.3: *Solving Π^{c}_{M3} in hydraulics*

Efficient Model Synthesis

Section 5.4 introduced the idea of a synthesis search space $\mathcal{M}_C$, which forms the set of possible global behavior models for a given system C. Actually, section 5.4 left open how a consistent behavior model can be searched efficiently, i.e., a behavior model that fulfills all behavior constraints and all demands.

Note that even for a rather simple circuit, $\mathcal{M}_C$ might contain several thousand elements. And, checking the consistency of an element $M_C \in \mathcal{M}_C$ usually requires the simulation of M_C. Thus, an intelligent exploration of the synthesis search space $\mathcal{M}_C$ is the key factor which decides whether a solution of Π^{c}_{M3} can be found at all in an acceptable time.

To get a grip on this model synthesis problem we developed domain-independent and domain-dependent concepts to control the exploration of $\mathcal{M}_C$. The essentials of our concepts are outlined below.

- *Incremental Constraint Update.* Each time a new value is inferred, its side effects on the model selection constraints are computed immediately.
- *Dependency Recording.* Within each inference step the inferred values are labeled with the responsible assumptions. The dependency recording in ^ard*deco* establishes cause-effect links between single parameters as well as between different sets of constraints, regardless of their type.
- *Topological Analysis.* The topology of a hydraulic circuit is investigated in order to determine global dependencies that have an effect on the constraint selection (e.g. flow direction analysis).
- *Domain Heuristics.* Heuristics that define preferences on behavior alternatives are evaluated during the inference process.

These concepts are tied together to a model synthesis control, which makes up a large part of ^ard*deco*'s inference engine. Figure 6.4 and 6.5 show ^ard*deco*'s entire inference procedure at an abstract level.

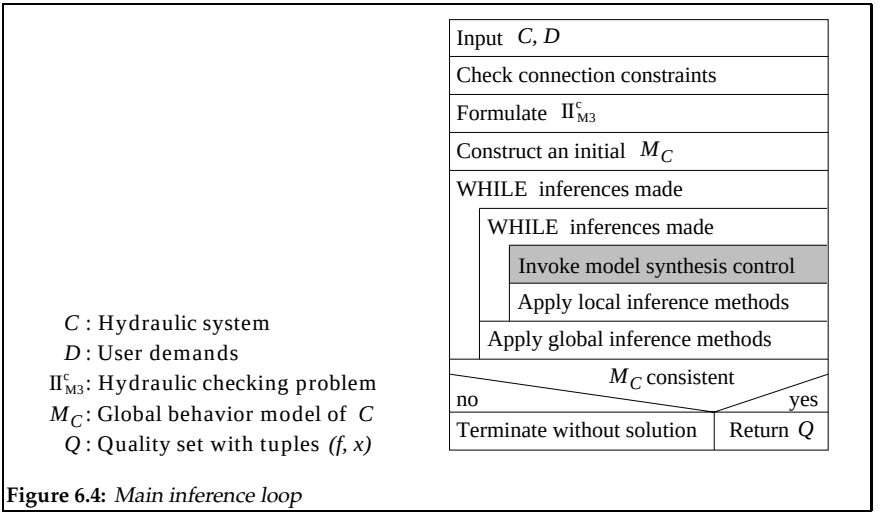
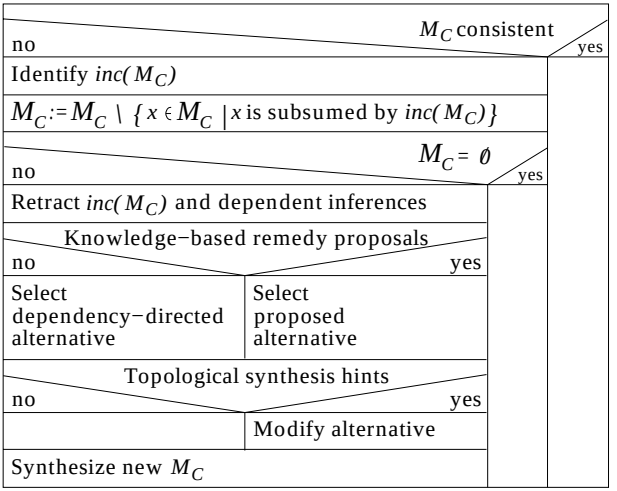


Figure 6.4: *Main inference loop*

Remarks. Within the main inference loop we distinguish local and global

inference methods. The former class comprises the methods for local value propagation, rule inference, and algebraic transformation; the methods of the latter class handle different types of equation systems. Section 5.4 gave an overview of these methods. Since local inference is causal and often more efficient as compared to global inference, it is applied before any global inference method is tried.

To ensure early pruning while exploring $\mathcal{M}_C$, the model synthesis control is invoked after each inference step.



M_C: Global behavior model
inc(M_C): Assumptions responsible for inconsistency in M_C
$\mathcal{M}_C$: Synthesis search space

Figure 6.5: *Model synthesis control*

Remarks. The set $\{x \in \mathcal{M}_C | x$ is subsumed by $inc(M_C)\}$ in figure 6.5 is comprised of those global behavior models of C whose assumptions contain the set $inc(M_C)$.

Note that the alternative selection and modification controls model synthesis by means of dependency-directed backtracking, knowledge-based backtracking, and the evaluation of topology constraints.

The identification of the assumptions $inc(M_C)$, which are responsible for an inconsistency in M_C, as well as the retraction of all consequences involved, requires a fairly sophisticated dependency recording.

Dependency Recording

The constraint network established by Π^{c}_{M3} consists of two kinds of nodes: nodes referring to functionalities (parameters) and nodes referring to behavior constraints. A constraint node b and a functionality node f will be linked by an edge, if f stands in the relation defined by b. Initially, each functionality is labeled either with f's alternative states or with *Unknown* (cf. figure 6.6).

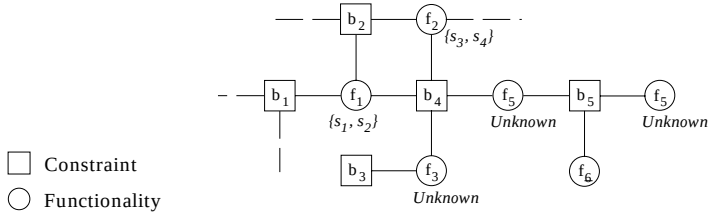


Figure 6.6: *The constraint network defined by the checking problem*

The value sets of those nodes f labeled *Unknown* are considered to be restricted by v_f. During the constraint satisfaction process, constraints are evaluated step by step and value sets are cut down to those values that match (fulfill) all actually triggered constraints. If a non-empty value set is assigned to each functionality, and each tuple $(x_1, x_2, \ldots, x_n)$, induced by these value sets, fulfills all constraints $b \in B$, the constraint satisfaction problem will be solved.

If a contradiction occurs in the course of constraint processing, the responsible nodes must be determined. These nodes, including all their consequences, must be retracted. Then, an alternative value assignment for the nodes that caused the contradiction can be selected, and inference can be continued.

The dependency recording concept is tailored to both the inference mechanisms and the constraints. In *ardeco* we distinguish between three types of dependency links:

- *Constraint Dependency.* Constraints can depend directly on other constraints: the so-called model selection constraints. If such a constraint is fulfilled, the dependent constraints are "active"; otherwise, they are "inactive". Throughout constraint processing new links are introduced in the constraint network that indicate both the inference (= supports) and the retraction (= depends) direction (cf. figure 6.7).

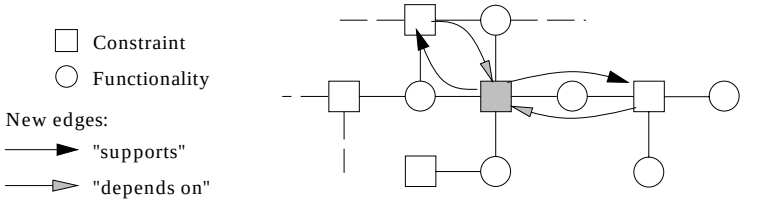


Figure 6.7: *Dependency recording between constraints*

- *Local Value Dependency.* The constraints processed during local propagation define a cause-effect chain between the nodes of the network. These relationships are recorded by the introduction of support links and by node labeling. Therefore, the root nodes of an inconsistency can be determined immediately, and their consequences can be traced for disbelief propagation purposes [52] (cf. figure 6.8).

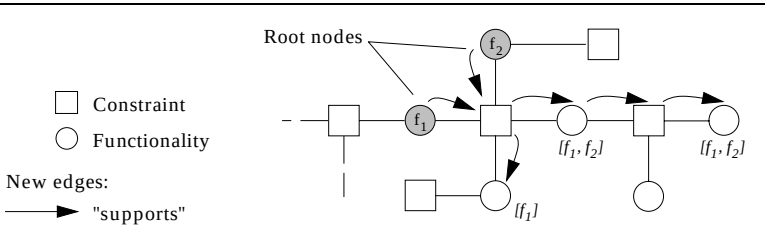


Figure 6.8: *Dependency recording during local propagation*

- *Global Value Dependency.* Constraints that cannot be treated by local propagation are called global. Global constraints establish cyclic dependencies between functionalities and constraints. Retracting one node of such a strong connected component results in the retraction of all nodes involved. In order to avoid labeling effort in $O(n \cdot m)$, with n, m specifying the number of functionalities and constraints respectively, not all dependency links are installed in the graph: only those that are necessary to instantiate a strongly connected component.

This dependency recording concept is an integrated part of each inference method in ARTdeco. It forms the base for a *dependency-directed* [65] and a *knowledge-based* backtracking. In either strategy the setting back to arbitrary points of the inference process is necessary. The former sets back to the root nodes of a contradiction where a new value assignment (= new alternative) is chosen chronologically. The latter provides deeper information about

which of the alternatives of the root nodes should be modified. In both cases all inferences that are no longer supported must be retracted, i.e., the constraint network has to be re-labeled.

Discussion

There is a lot of research related to "constraint satisfaction problems" [14], [17], [20], [23], [25], but only a small part of this research actually is contributed to hydraulic configuration. The term "constraint satisfaction problem" is somewhat misleading here since it is a label used for very different problem classes: One part of these problems is tackled by some kind of constraint propagation, while another part needs some kind of inherently global inference. E.g. Davis mentions six different categories of constraint propagation that are distinguished by the type of information which is updated [14].

An important category of constraint satisfaction problems are the so-called *label inference* problems. These problems deal with a network of nodes, each labeled with a set of possible values, and constraints that are used to restrict the value sets. Related to such problems, and certainly useful, are the terms "node consistency", "arc consistency", and "path consistency", which define different levels of local consistency [20], [47].

Note that in our constraint satisfaction problem, local consistency is not crucial. Most of the constraints define underdetermined relations on infinite sets. I.e., filtering value sets in its classical sense is hardly possible—but, filtering in the form of value range propagation is useful and partly necessary: value range information is exploited during the numerical subjobs and the constraint selection process.

Rather than "label inference" the following types of inference are employed to tackle the hydraulic checking problem:

- *Constraint Inference.* Constraint inference denotes a process where new constraints are inferred and added to the network.
- *Value Inference.* In the course of value inference, initially labeled nodes (the assumptions) along with the constraints are used to infer values of unlabeled nodes.

To make things more difficult, our constraint satisfaction problem Π^{c}_{M3} is inhomogeneous, i.e., very different types of constraints are employed. Thus, it cannot be tackled by a single method but needs a global con-

trol mechanism that both combines all required computation methods and maintains dependencies.

The dependency management in artdeco adopts concepts from Doyle's justification-based truth maintenance system (JTMS) [18] and from deKleer's assumption-based truth maintenance system (ATMS) [15]. Employing the classical ATMS-based dependency management would not be useful for performance reasons here: (*i*) Label-inferencing and updating all combinations of assumptions is not necessary, and (*ii*) maintaining ATMS-data structures constitutes an overhead as compared to recording the cause-effect dependencies during local propagation.

6.3 Knowledge Acquisition

artdeco operationalizes the component model of section 5.3. This component model defines all artdeco object classes, the structure of these classes, the building block model, and the syntax of the behavior descriptions. These concepts are an integral part of artdeco's philosophy. They are intended to simplify the creation and the processing of new components and cannot be modified.

Most of these concepts are kept transparent. From the users' point of view, artdeco objects represent a data structure that defines physical and graphic information (cf. figure 6.9).

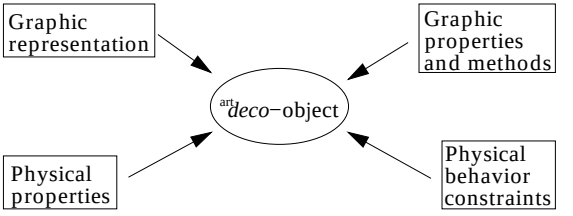


Figure 6.9: *The user's view of artdeco objects*

Physical and graphic information needs to be "synchronized". More precisely: If a connection line between two objects is drawn, it has to be ensured that there is also a physical correspondence between these objects. This correspondence is established by the gate concept; gates are designated areas of an objects graphic representation. They define how

external physical parameters can be referred and where components can be connected graphically.

The following EBNF-notation describes those parts of an *artdeco*-object that are user-definable:

⟨component⟩ ⟶ ⟨name⟩⟨physical-representation⟩⟨graphic-representation⟩
⟨physical-representation⟩ ⟶
    {GATE.⟨number⟩}$^*_1$ {⟨functionality⟩}$^*_0$ {⟨behavior-constraint⟩}$^*_0$
⟨functionality⟩ ⟶ ⟨name⟩⟨value⟩⟨default⟩⟨alternatives⟩
⟨name⟩ ⟶ ⟨symbol⟩
⟨default⟩ ⟶ ⟨value⟩
⟨alternatives⟩ ⟶ ({⟨value⟩}$^*_0$)
⟨value⟩ ⟶ ⟨symbol⟩ | ⟨number⟩

Remarks. The term ⟨behavior-constraint⟩ denotes an expression in *artdeco*'s behavior description language; ⟨graphic-representation⟩ denotes a collection of graphic primitives with different mouse-sensitive regions.

If a user wants to create a new component, he has to provide a set of behavior constraints as well as a graphic description with designated gates. *artdeco* takes this information, instantiates the necessary objects, updates the component model, and converts the behavior description into an internal form.

Strategy Language

artdeco's global inference strategy can be redefined. More exactly: *artdeco* provides a set of atomic inference techniques coping with different types of constraints like symbolic relations, equation systems, etc. Using some kind of BNF-syntax, these deduction techniques can be composed easily to a new, individual inference strategy. This is useful for adapting the inference process to the type of constraints given or to particularities of the domain.

The EBNF-notation below defines all productions that form a valid strategy for *artdeco*'s inference process. The semantics of the $\oplus$-symbol is as follows. The preceding deduction technique will be repeated until no further inference can be drawn. Note that this situation is always reached after a finite number of steps, since the number of constraints, variables, and alternatives is finite, and cyclic dependencies are detected.

⟨control-strategy⟩ ⟶ $\{\langle$control-strategy$\rangle\}_0^*$ | (⟨control-strategy⟩)$\oplus$ |
⟨local-inference⟩ | ⟨global-inference⟩
⟨local-inference⟩ ⟶ LOCAL-NUMERIC-PROPAGATION |
LOCAL-SYMBOLIC-PROPAGATION |
CONDITIONAL-CONSTRAINT-PROPAGATION |
DEMAND-TEST
⟨global-inference⟩ ⟶ SOLVE-LINEAR-EQUATION-SYSTEM |
SOLVE-NON-LINEAR-EQUATION-SYSTEM |
SOLVE-DIFFERENTIAL-EQUATION-SYSTEM

Behavior Description Language

The behavior description language establishes the interface to component behavior. Via this language interface one is able to modify and to maintain behavior descriptions of existing artdeco objects as well as to define the behavior of new ones.

artdeco's behavior description language is an implementation of the constraint language presented in section 5.3. Since the typical hydraulic engineer has no programming skill, it is kept as simple as possible and free of programming language details. Some of its characteristics are as follows:

- The language allows the formulation of (mixed) numerical and symbolic relations.
- The language is tailored to the connection philosophy of the building block model, which is defined on page 99. I.e., using the keywords [SELF] and [GATE], components may refer to their own functionalities as well as to information types at their gates.
- Parameters (variables) need not to be typed.

Usually, behavior constraints are specified in an *external* form, which in turn is converted into an internal representation that can be processed more efficiently. A detailed description of the constraint language and the converter can be found in [29].

6.4 Realization

Figure 6.10 gives a structural overview of artdeco. The important modules are briefly described below.

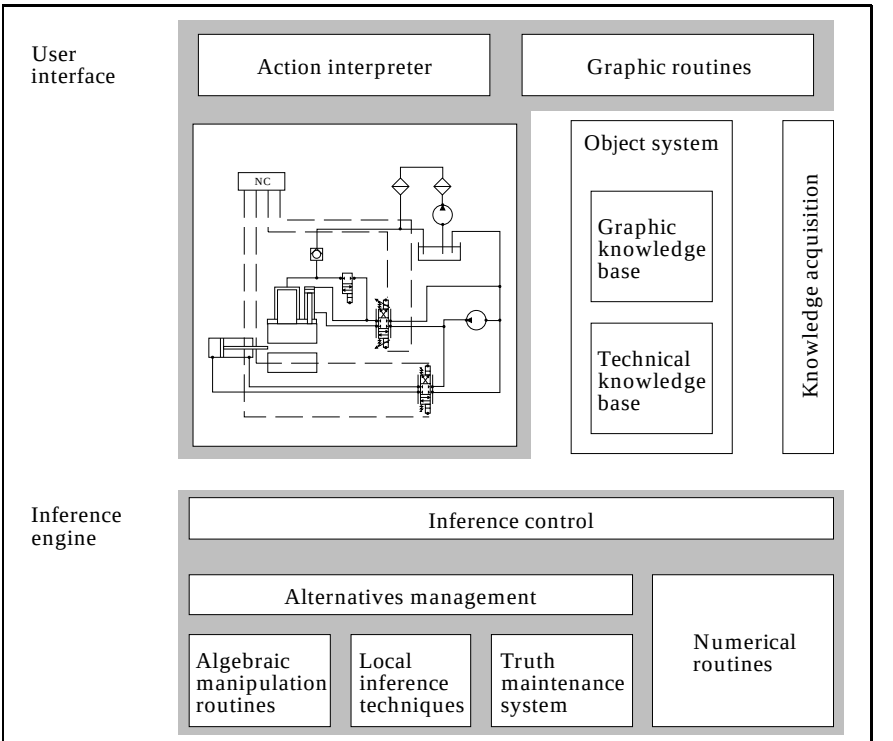


Figure 6.10: *Modules of* art*deco*

User Interface. The user interface consists of three main parts: (*i*) The front end realizing the circuit drawing area where a user manipulates graphic symbols and dialog boxes, (*ii*) an action interpreter taking a user's actions (mouse drag, double-click, keyboard input, etc.) and, dependent on the context, decides whether an action is valid or not, and (*iii*) a module providing graphic routines for a CAD-like handling of circuit diagrams.

Knowledge Bases. art*deco* provides a graphic and a technical knowledge base containing classes from which all user-visible objects are instantiated. The graphic classes predefine the eligible manipulation methods; the technical knowledge base defines the basic structure of important hydraulic component classes. The component catalog is built on top of these knowledge bases and can be extended with the aid of the acquisition module.

Knowledge Acquisition Module. The knowledge acquisition module provides

the language for behavior descriptions and an interface for the import of new graphic descriptions. The module checks new descriptions with respect to different syntactical and semantic aspects, instantiates the necessary objects in the knowledge bases, and makes the new components available in the component catalog.

Inference Engine. The inference engine provides several local and global inference techniques that are invoked by a global inference control. The inference control can be imagined as an interpreter that takes a hydraulic circuit along with the technical knowledge base (= Π^{c}_{M3}) as input and evaluates the global control strategy in order to find a solution of Π^{c}_{M3}.

Developmental Issues

There are two extreme positions of how a system may be developed that solves hydraulic instances of Π^{c}_{M3}:

1. *Tool-based.* By this approach we designate the strategy of selecting and combining tools where each solves a particular job of the entire problem. MAPLE or MATHEMATICA, for instance, are employed to do the algebraic manipulation and numerical computation jobs. Flexible object-oriented representation of data, rule processing, local propagation algorithms, and truth maintenance mechanisms are realized with aid of a powerful knowledge engineering tool. A CAD system establishes the front end. All tools are controlled by a command language, e.g. by TCL/TK [59]. Additionally, we need algorithms that filter the graphic descriptions and formulate instances of Π^{c}_{M3} which can be processed by the other tools.
2. *Language-based.* We will designate an approach language-based, if all concepts and algorithms are developed from scratch using one or more programming languages.

At first glance, the favorable developing approach seems to be closer to (1) than to (2). Rather for the following reasons the opposite is true:

- A circuit diagram given in a CAD system needs to be translated into single object representations that are both related one-to-one to hydraulic components and supplied with technical parameters and behavior descriptions.
- The integration of domain knowledge into a CAD system is difficult. But such an integration is exactly that what we need here. User

decisions that are not permitted should be detected as early as possible in order to avoid superfluous simulation effort.

- Since none of the tools could perform the entire constraint processing, a common constraint representation needs to be developed. This representation must be supplied with a truth maintenance mechanism and a global inference control that triggers the computation jobs.
- Generic numerical routines do not exploit physical restrictions of the domain. As a consequence, they may be less efficient than specially adapted algorithms. Also note that a numerically correct solution does not need to be physically correct.
- Rule processing and constraint inference have to be developed and integrated within the common constraint representation. Since knowledge acquisition is a heterogeneous task here, its operationalization would benefit from the language-based approach too.

Tackling all these problems is a demanding job that cannot be done in a single step since new concepts need to be developed and evaluated. Moreover, users should participate in the development process as early as possible.

We addressed this situation by dividing the development process of *art*$^{\text{inf}}$*deco* into two stages:

1. *Prototype Stage.* In the first place, we had to get a clear idea of how configuration in hydraulics could be supported. Research related to graphic problem formulation went hand in hand with research related to the necessary inference types, the expressiveness of a component language, adequate data structures, and the interplay of different inference mechanisms.
2. *Reimplementation Stage.* The reimplementation stage was *not* a duplication of the prototype stage. Rather, research has been concentrating on the design of efficient concepts and algorithms: From a user's point of view, a configuration problem cannot be "solved in principle" but needs to be solved in an acceptable time.

Throughout the prototype stage we used the knowledge engineering environment KEE to realize our ideas [30]. At the end of this stage, *art*$^{\text{inf}}$*deco*'s philosophy, its architecture, and a large part of the necessary methods had been developed or evaluated. The algorithms (local deduction techniques,

algebraic routines, graphic routines, the acquisition module) were written in COMMON LISP, the knowledge bases were built on top of the KEE object system, and the user interface was based on the KEE picture system. Aside from improving and developing our concepts, the KEE/LISP version of *art deco* served as a realistic communication base between users and developers.

When we started reimplementing *art deco*, we refrained from the employment of particular shells or knowledge engineering tools. Tools often restrict the portability and usually lead to a loss of performance.

art deco's philosophy requires a tight combination of the inference process on the one hand and the user interaction on the other. Thus, we developed a small graphics kernel that can be ported easily to other platforms. On top of this graphics kernel, we built a "semantic" graphics layer that provides powerful graphic commands *related to the application*. This semantic graphics layer interprets user actions and manipulates the technical and the graphic knowledge base. For efficiency and maintenance reasons a specialized object system for the representation of knowledge bases was also developed.

If an inference process is invoked, *art deco* constructs a constraint satisfaction problem using the actual instantiations of the technical and the graphic knowledge base. This job is then passed to the inference control that employs local and global inference techniques, truth maintenance, and an alternatives management to solve the problem. Since these techniques need a coordinated interplay, both a generic constraint representation, wherein all constraints can be formulated, and tailored constraint processing methods have been developed.

The actual version of *art deco* is realized in COMMON LISP and C/C++. The object system, the graphic interface, and the numerical routines are written in C/C++; the other modules of the inference engine are written in COMMON LISP. Parts of the knowledge acquisition module were developed with the UNIX tools LEX and YACC.

Summary & Conclusion

The contributions of this thesis to the area of configuration were as follows.

1. *Models of Configuration.* In relevant literature on the subject, configuration problems are often classified vaguely or by procedural aspects of particular configuration methods. Such a classification can be misleading, since it neglects that only a small part of a configuration problem can be solved domain-independently, i.e., by a generic approach. However, solving a configuration problem usually requires a thorough investigation of the real task and the related domain.

   The thesis in hand presented the *component model* view of configuration problems and systems. This approach moves the description of configuration objects into the center and hence, it is both closer to domain-dependent problems and provides a realistic view on a problem's complexity.

   Two main classes of component models were distinguished within the presented classification scheme: the structure-based models and the function-based models. The former class comprises associative, compositional, and taxonomic models; the latter comprises property-based and behavior-based models. For three models, relevant from the configurational standpoint, a clear and precise formalization was developed. Based on this formal framework, it could be shown that the purely property-based component model M1 and the model M2, which additionally allows the formulation of structural knowledge, are equivalent.

Structural component models establish a global view on the system to be configured while functional component models rely upon local connections only. This conceptual difference lets functional component models be

superior to structural models regarding knowledge acquisition, flexibility, and extendibility. Functional component models played a special role in the second part of this thesis:

2. *Property-based Configuration.* The advantages and drawbacks of property-based configuration were discussed. For a particular property-based description, the resource-based description, the configuration system MOKON was presented. It was shown in which way resource-based descriptions can be exploited to improve the performance of the configuration process and to support knowledge acquisition.

3. *Behavior-based Configuration.* The configuration of hydraulic systems was introduced as a sophisticated configuration/design task that requires a deep functional understanding of the domain. It was demonstrated how a behavior-based checking of such complex systems can be automated—more precisely:

   - We developed a component model and a processing approach for arbitrarily structured hydraulic systems. The component model enables a modular composition of technical systems and the formulation of numerical as well as symbolic relations; the processing approach combines model synthesis, different inference methods, knowledge-based and dependency-directed backtracking, and efficient truth maintenance concepts.
   - Apart from knowledge processing, user interaction and knowledge acquisition were also identified as demanding problems of the configuration/design process in hydraulics. To address these problems concepts of an almost entirely graphic problem specification and a behavior language tailored to hydraulics were developed.
   - New concepts that improve the processing of behavior descriptions in hydraulics were developed.
   - Most of the above concepts have been operationalized. We presented the system *ardeco* that, based on a circuit diagram, automates model selection and model synthesis in order to perform the checking of the related hydraulic system. The problem specification philosophy along with the integrated inference concepts as realized by *ardeco* shows a new quality in supporting hydraulic circuit design.

Conclusion

Can configuration technology be scaled up to manage complex design tasks? Yes, it can. But not due to the existence of a generic design algorithm.

In the course of our work we made the following observations:

1. Support for complex design tasks needs a profound analysis of the domain in order to strictly differentiate between an expert's strong and weak points respectively. Put another way, the efficiency of a design support depends on "what is left" to the user in the end.

   E.g., remember the design of hydraulic circuits. Even if we had a configuration system that could do the creative synthesis step but left the analysis step to the human expert, it would not be of much help.

2. The base of each configuration system for design support is the domain theory associated with the problem to be solved. I.e., there is no hope of finding a short cut to a solution e.g. by employing "intelligent" or knowledge-based techniques.

3. If there is intelligence behind a knowledge-based system for design support, it is the way the (design) problem space is explored. An intelligent problem space exploration is, for the most part, the result of an adequate selection, combination, and customization of existing algorithms and technologies. Exactly this is the challenge.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Massachusetts, 1983.

[2] M. Bauer, B. Stein, and J. Weiner. Problemklassen in Expertensystemen. *KI*, 3:13–18, Sept. 1991.

[3] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9:171–216, 1985.

[4] A. Brinkop, N. Laudwein, and R. Maassen. Routine Design for Mechanical Engineering. In *IAAI '94 Proc. of the Sixth Annual Conference on Innovative Applications of AI, Seattle, August 1-3*, 1994.

[5] D. Brown and B. Chandrasekaran. An Approach to Expert Systems for Mechanical Design. In *Trends and Applications '83*. IEEE Computer Society, NBS, Gaithersburg, MD, 1983.

[6] D. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.

[7] T. Bylander and B. Chandrasekaran. Generic Tasks for Knowledge-based Reasoning: the "right" Level of Abstraction for Knowledge Acquisition. *Int. J. Man-Machine Studies*, 26:231–243, 1987.

[8] B. Chandrasekaran. Generic Tasks as Building Blocks for Knowledge-based Systems: the Diagnosis and Routine Design Examples. *The Knowledge Engineering Review*, 3(3):183–219, Sept. 1988.

[9] B. Chandrasekaran. Design Problem Solving. *AI Magazine*, 11:59–71, 1990.

[10] B. Chandrasekaran and R. Milne. Reasoning About Structure, Behavior, and Function. *SIGART Newsletter*, Juli 85(93):4–59, 1985.

[11] W. J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[12] S. Cook. The Complexity of Theorem-Proving. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158. Association for Computing Machinery, 1971.

[13] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—An Approach to Domain-independent Construction. Technical Report 21, BMFT Verbundsprojekt, Universität Hamburg, FB Informatik, Mar. 1989.

[14] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.

[15] J. de Kleer. Problem Solving with the ATMS. *Artificial Intelligence*, 28:197–224, 1986.

[16] J. de Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24:7–83, 1984.

[17] R. Dechter and J. Pearl. Network-based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, pages 1–38, 1988.

[18] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[19] L. Eshelman. MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic Publishers, Boston, 1988.

[20] E. C. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11):958–966, Nov. 1978.

[21] U. Gappa. CLASSIKA: A Knowledge Acquisition System Facilitating the Formalization of Advanced Aspects in Heuristic Classification. In J. H. Boose, B. R. Gaines, and M. Linster, editors, *Proceedings of EKAW '88*, GMD-Studie 143, pages 31/1–31/17, Sankt Augustin, 1988. GMD.

[22] J. S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.

[23] J. Gosling. *Algebraic Constraints*. Dissertation, Carnegie-Mellon University, Department of Computer Science, May 1983.

[24] A. Günter. *Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen*. Dissertation, Universität Hamburg, Fachbereich Informatik, 1992.

[25] H.-W. Güsgen. CONSAT—*A System for Constraint Satisfaction*. Dissertation, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, Nov. 1987.

[26] B. Hayes-Roth. The Blackboard Architecture: A General Framework for Problem Solving. Heuristic Programming Project HPP–83–30, Stanford University, Computer Science Department, Heuristic Programming Projekt, May 1983.

[27] M. Heinrich and E. W. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems for Modular Components. In *Proc. CAIA '91*, pages 257–264, 1991.

[28] M. Hoffmann. Algorithmen zur Verarbeitung von topologischen Informationen in Netzwerken. Diploma thesis, Gerhard-Mercator-Universität - GH Duisburg, Fachbereich Mathematik / Informatik, Sept. 1993.

[29] U. Husemeyer. Entwurf und Realisierung einer Verhaltensbeschreibungssprache für technische Expertensysteme. Diploma thesis, University of Paderborn, Department of Mathematics and Computer Science, 1995.

[30] IntelliCorp. KEE *User's Guide*. IntelliCorp, Inc, 1975 El Camino Real West, California, 1988.

[31] W. Karbach and M. Linster. *Wissensakquisition für Expertensysteme*. Carl Hanser Verlag, 1990.

[32] W. Karbach, M. Linster, and A. Voß. Models of Problem Solving: One Label—One Idea? In B. Wielinga and J. Boose, editors, *Proceedings of EKA '90*, pages 173–189. IOS Press, Amsterdam, 1990.

[33] W. Karbach, X. Tong, and A. Voß. Closing the Knowledge Acquisition Gap: From KADS Models of Expertise to ZDEST-2 Expert Systems. In J. H. Boose, B. R. Gaines, and M. Linster, editors, *Proceedings of EKAW '88*, GMD-Studie 143, pages 31/1–31/17, Sankt Augustin, 1988. GMD.

[34] A. Kecskeméthy. MOBILE—An Objectoriented Tool-Set for the Efficient Modeling of Mechatronic Systems. In M. Hiller and B. Fink, editors, *Second Conference on Mechatronics and Robotics*. IMECH, Institut für Mechatronic, Moers, IMECH, 1993.

[35] J. Kippe. Komponentenorientierte Repräsentation technischer Systeme. In H. W. Früchtenicht, editor, *Technische Expertensysteme: Wissensrepräsentation und Schlußfolgerungsverfahren*. R. Oldenbourg Verlag, München, Wien, 1988.

[36] H. Kleine Büning, D. Curatolo, and B. Stein. Configuration Based on Simplified Functional Models. Technical Report tr-ri-94-155, Universität-GH Paderborn, Fachbereich Informatik, 1994.

[37] H. Kleine Büning, D. Curatolo, and B. Stein. Knowledge-Based Support within Configuration and Design Tasks. In *Proc. ESDA '94, London*, pages 435–441, 1994.

[38] H. Kleine Büning, N. Kleinjohann, and S. Schmitgen. Wissensbasierte Produktkonfiguration. *VDI–Z*, 132(10):161–163, Oct. 1990.

[39] H. Kleine Büning and B. Stein. Supporting the Configuration of Technical Systems. In M. Hiller and B. Fink, editors, *Second Conference on Mechatronics and Robotics*. IMECH, Institut für Mechatronic, Moers, IMECH, 1993.

[40] B. Kuipers. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence*, 24:169–203, 1984.

[41] T. Laußermair and K. Starkmann. Konfigurierung basierend auf einem Bilanzverfahren. In *6. Workshop "Planen und Konfigurieren", München*, FORWISS, FR-1992-001, 1992.

[42] R. Lemmen. Checking the Static and Dynamic Behaviour of a Hydraulic System. In *Proceedings of the 1th Asian Control Conference, Tokyo*, 1994.

[43] R. Lemmen. *Zur automatisierten Modellerstellung, Konfigurationsprüfung und Diagnose hydraulischer Anlagen mit dem Beispiel tankdruckangehobener Differentialzylinderantriebe*. Number 503 in Fortschrittsberichte VDI, Reihe 8: Mess-, Steuer- und Regelungstechnik. VDI-Verlag, Düsseldorf, 1995.

[44] R. Lemmen and B. Stein. Wissensbasierte Konfigurationsprüfung hydraulischer Anlagen. *at – Automatisierungstechnik*, 3, 1994.

[45] M. Linster. Using the Operational Modeling Language OMOS to Represent KADS Conceptual Models. In J. B. G. Schreiber, B. Wielinga, editor, KADS*: Knowledge Acquisition and Design Structuring*. Academic Press, 1992.

[46] M. Linster, W. Karbach, A. Voß, and J. Walther. An Analysis of the Role of Operational Modeling Languages in the Developement of Knowledge-based Systems. Forschungsbereich Künstliche Intelligenz, FABEL Report No. 3, GMD, Sankt Augustin, 1993.

[47] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 24:169–203, 1994.

[48] M. L. Maher. Process Models for Design Synthesis. *AI Magazine*, pages 49–58, 1990.

[49] S. Marcus. *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, 1988.

[50] S. Marcus and J. McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1989.

[51] S. Marcus, J. Stout, and J. McDermott. VT: An Expert Elevator Designer that Uses Knowledge-based Backtracking. *AI Magazine*, pages 95–112, 1988.

[52] J. P. Martins and S. C. Shapiro. A Model for Belief Revision. *Artificial Intelligence*, 35:25–79, 1988.

[53] Math Works Inc. SIMULINK *User's Guide*. Math Works Inc., Nattik, Massachusetts, 1992.

[54] J. McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*, 19:39–88, 1982.

[55] J. McDermott. Preliminary Steps Towards an Taxonomy of Problem-Solving Methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–266. Kluwer Academic Press, 1988.

[56] O. Najmann and B. Stein. A Theoretical Framework of Configuration. In *Proc. IEAAIE '92*, Paderborn, 1992.

[57] Y. Nakashima and T. Baba. OHCS: Hydraulic Circuit Design Assistant. In *First Annual Conference on Innovative Applications of Artificial Intelligence*, pages 225–236, Stanford, 1989.

[58] D. Norman. Some Observations on Mental Models. In A. Genter, editor, *Mental Models*. Hillsdale, New Yersey, 1983.

[59] J. K. Ousterhout. TCL: An Embeddable Command Language. In *USENIX Conference*, 1990.

[60] M. Piechnick and A. Feuser. MOSIHS – Programmsystem zur Simulation komplexer elektrohydraulischer Systeme. In *AFK, Aachener Fluidtechnisches Kolloquium*. Mannesmann Rexroth GmbH, Lohr, Germany, 1994.

[61] F. Puppe. *Problemlösungsmethoden für Expertensysteme*. Springer-Verlag, 1990.

[62] M. M. Richter. *Prinzipien der Künstlichen Intelligenz*. B. G. Teubner, Stuttgart, 1989.

[63] H. R. Schwarz. *Numerische Mathematik*. B. G. Teubner, Stuttgart, 1986.

[64] E. Soloway, J. Bachant, and K. Jensen. Assessing the Maintainability of XCON-IN-RIME: Coping with the Problems of a VERY Large Rule-Base. In *Proceedings AAAI—Sixth National Conference on Artificial Intelligence*. Morgan Kaufmann, July 1987.

[65] R. M. Stallman and G. J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9:135–196, 1977.

[66] L. Steels. Components of Expertise. *AI Magazine*, 11(2):28–49, 1990.

[67] B. Stein and R. Lemmen. ARTDECO: A System which Assists the Checking of Hydraulic Circuits. In *Workshop for Model-based Reasoning, ECAI '92*, 1992.

[68] B. Stein and J. Weiner. Model-based Configuration. In *OEGAI '91, Workshop for Model-based Reasoning*, 1991.

[69] B. Stein and J. Weiner. MOKON: Eine modellbasierte Entwicklungsplattform zur Konfiguration technischer Anlagen. In *5. Workshop "Planen und Konfigurieren", Hamburg*, LKI-M-1/91, 1991.

[70] P. Struß. Assumption-based Reasoning about Device Models. In H. Früchtenicht, editor, *Wissensrepräsentation und Schlußfolgerungsverfahren*. Oldenbourg Verlag, München, 1988.

[71] M. Suermann. Wissensbasierte Modellbildung und Simulation von hydraulischen Schaltkreisen. Diploma thesis, Universität-GH Paderborn, FB 17 Mathematik / Informatik, 1994.

[72] I. Syska. *Modulare Problemlösungsarchitekturen für Konstruktionssysteme*. Dissertation, Universität Hamburg, Fachbereich Informatik, 1992.

[73] I. Syska, A. Günter, R. Cunis, H. Peters, and H. Bode. Solving Construction Tasks with a Cooperating Constraint System. In B. Kelly and A. Rector, editors, *Research and Development in Expert Systems V*, Proc. of Expert Systems '88, Brighton. Cambridge University Press, 1989.

[74] W. Tank. *Modellierung von Expertise über Konfigurierungsaufgaben*. Dissertation, Universität (TU) Berlin, Fachbereich Informatik, 1992.

[75] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, June 1972.

[76] C. Tong. Towards an Engineering Science of Knowledge-based Design. *Artificial Intelligence in Engineering*, 2(3):133–166, 1987.

[77] G. Tseitin. On the Complexity of Derivations in Propositional Calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic*, pages 466–483. Springer Verlag, 1983.

[78] S. Twine. Towards a Knowledge Engineering Procedure. In B. Kelly and A. Rector, editors, *Research and Development in Expert Systems V*, Proc. of Expert Systems '88, Brighton. Cambridge University Press, 1989.

[79] H. Vo. *Representing and Analyzing Causal, Temporal, and Relations of Devices*. Dissertation, Universität Kaiserslautern, Fachbereich Informatik, 1986.

[80] A. Voß and W. Karbach. MODEL-K: KADS Grows Legs. In G. Schreiber, B. Wielinga, and J. Breuker, editors, KADS*: Knowledge Acquisition and Design Structuring*. Academic Press, 1992.

[81] J. Weiner. *Aspekte der Konfigurierung technischer Anlagen*. Dissertation, Gerhard-Mercator-Universität - GH Duisburg, FB 11 Mathematik / Informatik, 1991.

[82] B. Wielinga, W. V. de Velde, G. Schreiber, and H. Akkermans. Towards a Unification of Knowledge Modeling Approaches. Technical Report Esprit Project P5248 KADS-II, University of Amsterdam, University of Brussels, 1991.

Bibliographical Note

Benno Maria Stein was born in Euskirchen in 1964. He was educated at Krefeld. From 1984 until 1988 he studied Industrial Engineering at the University of Karlsruhe. He was a trainee at IBM and finished his diploma thesis in 1989 at the IBM CIM CENTER, Böblingen. From 1989 until now he has been a doctoral student of Computer Science, first at the universities of Bochum and Duisburg, and since 1991 at the University of Paderborn.