Model Construction in Analysis and Synthesis Tasks

Habilitation Thesis

Benno Maria Stein

Dept of Mathematics and Computer Science University of Paderborn

Contents

Preface

This thesis is on the automation of knowledge-intensive tasks.

The term "task" can be further narrowed with respect to analysis tasks and synthesis tasks in engineering domains. To the former we count classification problems, simulation problems, or diagnosis problems; to the latter we count configuration problems, design problems, or planning problems. Each of these problems may entail problems in turn, relating to knowledge acquisition, solution assessment, or hypothetical reasoning—to mention only a few. We call the tasks knowledge-intensive because either a problem solving method is not at hand and must be constructed at first, or, we do not know which of the available methods should actually be applied.

The term "automation" comprises a wide spectrum of interpretations. Ideally, task automation means to us that we can abandon the human engineering factor in the course of problem solving. However, in practice, a computer-based automation is restricted to selected aspects of the task in question: simulation, decision support, expert critiquing, model construction, search in synthesis spaces, visualization, etc. The focus of the thesis in hand lies on model construction.

Automating engineering tasks means working in an interface area: Even if we restricted ourselves to a single, particular diagnosis or design problem, knowledge and methods from various fields would have to be combined in a skillful way.

Of course, this situation makes the automation of knowledge-intensive tasks both an exciting challenge and an artistic discipline. However, this situation also makes a text that is concerned with this interface area vulnerable from positions of the fields that it touches: Engineering design (207), method engineering (285), systems engineering and design (305, 310), object-oriented modeling and simulation (38), software engineering (88), artificial intelligence (230, 203), knowledge-based methods for analysis and synthesis (214, 257). Starting from a pure engineering field and going to the field of artificial intelligence, own fundamentals and theories have been developed which cannot be integrated to their full extent, or whose latest developments may not be adopted within an interdisciplinary text.

This thesis is written from the position of a computer scientist and artificial intelligence researcher, however, with the intention to come—in the intersecting areas—as close as possible to engineering terminology and way of thinking. The thesis may be useful for computer scientists who are concerned with interdisciplinary problem

solving, say, who work in a close cooperation with engineers or domain experts on the automation of analysis and synthesis tasks.

The Starting Point Our starting point is characterized as follows. We are given either a system, S, or a set of systems, $\mathcal{S}$, along with a shortcoming of information about S or $\mathcal{S}$. From the viewpoint of a domain expert, the underlying models of S and $\mathcal{S}$ are well defined, and the question that is to be answered can be formulated in a definite way—for example:

- Which component is broken in S? (diagnosis $\sim$ analysis)
- How does S react on the input u? (analysis)
- Does a system with the desired functionality exist in $\mathcal{S}$? (synthesis)

If these analysis and synthesis questions shall be answered automatically, both adequate algorithmic models along with the problem solving expertise of the human problem solver must be operationalized on a computer. In this context, the construction of an adequate model often establishes the actual challenge when tackling the problem. That is the place where this thesis sets in.

Contributions

Model construction (model creation, model formulation, model finding, model building) is an artistic discipline, which highly depends on the reasoning job to be done. Model construction can be supported by means of a computer, and we investigate selected aspects from this field; the theoretical and conceptual contributions of this thesis can be grouped as follows.

- *Classification of Existing Work to Automated Modeling.* We classify existing approaches to model construction with respect to their position in the model hierarchy. Nearly all of the existing methods support a top-down procedure of the human modeler; they can be characterized as being either structure-defining (top), structure-filling (middle), or structure propagating (down).
- *Model Construction from a Metamodel Perspective.* Domain experts or knowledge engineers seldom start from scratch when constructing a new model; instead, they develop an appropriate model by modifying an existing one. Following this observation, we analyzed the projects on which we have worked (or are still working) from a metamodel perspective.[1] We classify the found model construction principles as follows: Model refinement, model simplification, model compilation, model reformulation, and model envisioning.

[1] A metamodel is a model of the modeling process itself; it is closely related to metamodeling: *"By metamodeling, we imply a process of design carried out at the metalevel, by which we define how the process of modeling (at the lower level) is to be carried out."* (Gigch, 93, pg. 92).

While model simplification and model refinement establish rather known principles to tackle analysis and synthesis problems in technical domains, the other principles have not been characterized in a generic form. In this context, we introduce model envisioning as a problem solving method which implies no constructional mission—we use it as a collective term for methods that prepare structure models in a graphical way in order to provide insights or to simplify the access when dealing with large systems.

- *Modeling Knowledge on Structure.* The aforementioned model construction principles can be subsumed under the term "system morphism", which was employed by Wymore and extended by Zeigler et al. (305, 310). With the introduction of so-called design graph grammars we make an instrument available to describe system morphisms with the focus on structure models.

  Design graph grammars are an advancement of classical graph grammar approaches and have been developed with a close look at expressiveness and applicability in engineering domains. They serve as an efficient concept to create and to manipulate structure models of technical systems when working on synthesis tasks. In particular, they can be used as a formal tool to quantify the quality of automatically generated solutions for a synthesis problem.

  Since design graph grammars provide a precise semantics for the structural modification a model undergoes during its simplification or transformation, they are used throughout the case studies of Part II as a formal description methodology.

Aside from the above conceptual considerations, each of the case studies in Part II establishes a contribution on its own in the field of knowledge-based problem solving. In the style of a paper title from Falkenhainer and Forbus, the case studies could be summarized under the motto *"Constructing the Right Model for the Job"*.[2] The following results may be emphasized.

- Chapter A introduces the paradigm of functional abstraction—a generic procedure to tackle complex synthesis problems. Put it overstated, the paradigm says: At first, we construct a poor solution of a design problem, which then must be repaired.

- Section A.1 (model simplification) shows how the paradigm of functional abstraction is successfully applied in the fluidic domain. Besides, this section contributes some aspects to the field of case-based reasoning.

[2] *"Compositional Modeling: Finding the Right Model for the Job"* (71). The article presents concepts and strategies to automatically construct a "useful model" when given three things: A general domain theory, a structural description of a specific system, and a query about the system's behavior. Falkenhainer and Forbus'work addresses the problem of answering a query while minimizing extraneous detail.

- Section B.1 (model compilation) presents a preprocessing concept to speed up the method of resource-based configuration. This way it becomes possible to combine the user-friendly (local) resource-based modeling method with the efficiency of a global, variant-based configuration strategy.

- Section B.2 (model compilation) exploits the idea of fault models of the GDE+ (275) to generate tailored heuristic diagnosis models from first principles. Our approach overcomes some problems of model-based diagnosis approaches, which are caused by long interaction paths and feedback structures in the system of interest.

  Moreover, this section presents a domain-independent measure to assess the diagnosability of a technical system S. This measure relies on an information-theoretical interpretation of a database with simulation results of S.

- Section C.1 (model reformulation) shows how a synthesis scheme for the design of wave digital structures is operationalized on a computer. Our approach is based on the triconnected component algorithm of Hopcroft and Tarjan (111) and leads to a linear time procedure (in the number of the electrical circuit elements). It is optimum with respect to both the synthesized structures and the runtime.

- Section C.2 (model reformulation) addresses the construction of a similarity measure from a classified collection of objects. At heart, we pick up Richter's argumentation: Similarity knowledge is distributed over the containers vocabulary, similarity measure, case base, and solution transformation, where each container is able to contain all available knowledge (223).

- Chapter D demonstrates how model envisioning is applied as a powerful reasoning concept. We present three applications each of which exploits envisioning in a different connection: As model formulation aid for computer networks, to simplify the understanding and maintenance of knowledge bases, and for the functional analysis of fluidic systems.

- Section D.1 (model envisioning) introduces a new measure to detect "natural" structures in graphs. The measure is called Λ, it is based on a graph's connectivity, and a maximization of a graph's Λ-value leads to clustering results superior to existing graph clustering algorithms.

  Moreover, the heuristic algorithm MAJORCLUST is presented that maximizes the weighted partial connectivity Λ. Our experiments show that MAJORCLUST outperforms the well-known clustering approaches with respect to both runtime and clustering results.

Thesis Structure

The thesis is organized into two parts. Part I presents general and theoretical aspects of model construction; Part II is comprised of the case studies that are concerned with the operationalization of knowledge-intensive tasks.

Part I Framework Chapter 1 starts with a short discussion on models and model construction; in particular it relates the term "model", which is coined by the conception of system analysis, to the term "model space" that is to be seen in connection with synthesis tasks. Main contributions of this chapter are Section 1.2 and 1.3, which address the question of how model construction can be supported by means of a computer.

Chapter 2 presents a framework for the description of models for modular technical systems. The framework is powerful enough to become applied to a variety of domains and modeling scenarios; however, a great deal of attention has been paid to keep it as clear and simple as possible.

Our view to models and model construction is oriented at structure and behavior; and, by specifying the concepts of locality, causality, and no-function-in-structure, the interweaving of structure and behavior becomes quantifiable.

In particular, Section 2.4 introduces, isolated from a specific problem and a concrete domain, the five different model construction principles mentioned above.

Chapter 3 is devoted to design graph grammars. It starts with an introduction and motivation around structure model manipulation and then develops design graph grammars as an appropriate means for this objective.

The use of design graph grammars is connected to various theoretical issues, which are addressed in the last two sections of this chapter. They examine the relationship of design graph grammars to classical graph grammars and present issues concerning the application of graph grammars in structure analysis and system design.

Part II Case Studies The case studies presented in Part II have been selected from our research activities. Of course, they should not be regarded as a representative spectrum with respect to model construction in analysis and synthesis tasks. On the other hand, especially against the background that recipes for the operationalization of a model construction principle can hardly be stated, the realized implementations may serve as a pool of ideas when tackling new analysis and synthesis tasks.

To work out common ground and differences between the mentioned construction principles, the models in the case studies (that stem from different domains and applications) are formalized in a unique manner. In a way, this involves a description of a model under a structural point of view, under a behavioral point of view, or under both.

Moreover, each case study comes along with the same set up: It starts with a (hopefully) illustrative outline of the respective task. Then follow four subsections; the first of which contains a formal specification of the underlying models while the last subsection recapitulates the case study in brief outlines.

Clearly, different readers have more or less previous knowledge regarding a case study's underlying domain theory, problems, or methods. We tried to make up for this situation by a consistent problem formulation, which, however, may be too succinct or too long winded, as the case arises.

Instead of a Conclusion

Within each of the case studies a conclusion may be drawn against the background of the domain in question. Nevertheless, if we had to draw a *comprising conclusion* it would express the considered opinion that a powerful model construction support is possible. But, we cannot expect that there is a recipe for constructing adequate models automatically.

The success of a diagnosis or design approach depends on the underlying problem space—say, its size, and the way it is explored. A tractable problem space is in first place the result of adequate models, which in turn are the result of a skillful selection, combination, and customization of existing construction principles.[3]

[3]The statement is in accordance with Tolvanen's observations, who has investigated modeling tools for incremental method engineering (285).

Part I

Framework

1

Models and More

Numerous definitions, explanations, and introductions have been formulated about the term model. However, the reader may not be anxious that an exhaustive discussion of this term is given here. Instead, I will start with my favorite definition, which stems from Minsky, and bring in extensions, different interpretations, as well as additional explanations at the appropriate places.

> *"To an observer B, an object A^* is a model of an object A to the extent that B can use A^* to answer questions that interest him about A."*
>
> Minsky, 1965, pg. 45

In this thesis

- the interesting objects, A, are technical systems,
- the observer, B, is a domain expert who works on a problem solving task from the field of analysis, such as a diagnosis task, or synthesis, such as a design task,
- the questions are embedded in a ternary way; they relate to the technical system, to the problem solving task, and to the domain expert,
- the models, A^*, are exactly that what Minsky has pointed out above.

Before we start talking on model construction, we go back one step and clarify in the next section, 1.1, purpose and meaning of models. Main contributions of this chapter are Section 1.2 and 1.3, which discuss how the construction of models and model spaces can be supported, and which develop a classification scheme for the various types of model construction approaches. Finally, Section 1.4 outlines contact points and differences of our work in respect to developments from the field of Artificial Intelligence.

1.1 Motivation and Disambiguation

Models are the essential element within the human ability to reason about systems. A system, S, can be considered as a clipping of the world and has, as its salient property, a boundary. On account of this boundary, it can be stated for each object of the world whether or not it is part of S.[1] As quoted at the outset, the purpose of a model is to answer questions about a system of interest. The process of developing a model for a system is called model formation, model creation, or modeling.

In the following we shed light on this process and motivate why models can be used to answer questions about systems.

Our starting point is characterized as follows. We are given a system S along with a shortcoming of information about S. In the case that S represents a technical device this information may relate to an open question about the behavior of S, or about the probability that S will be break down when being run with overload, or simply—about the cost of S.

Answers to such questions can be found by performing tailored experiments with S. Such experiments comprise purposeful excitations of S while accurately observing its reactions or modifications, which are called behavior here. By interpreting a system's behavior, answers to the question of interest may be found (see Figure 1.1, which is derived from Goldschmidt (97, pg. 3)).

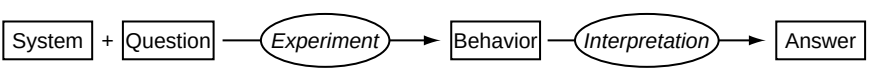


Figure 1.1. Experimentation as a concept to eliminate information deficits relating to a system.

Clearly, experimenting with a system is not possible if the open questions are of a hypothetical nature and relate to an, up to now, non-existing system. As well as that, even if a system S exists, several reasons may forbid experimentation with S. Some of them are: The experiment cannot be set up at all, the system's reactions are too slow and too fast respectively to become observed, the experiment is too expensive or too risky to be executed, or the experiment's influence onto the system cannot be accepted. Jeffers (124), Cellier (37, pg. 10), or Murray-Smith (190, pg. 13) present discussions of these problems.

Given such a situation, a way out is the creation of a model M from the system S and then to execute the experiment on M (see Figure 1.2). Performing an experiment on a model is called simulation since Korn and Wait (see (150) quoted in Cellier (37, pg. 6)).

As depicted in Figure 1.2, a model does not depend on the system as the only source of information but is formed purposefully, in close relation to the question

[1] This characterization of the term system is derived from Gaines's definition (87), which, like other definitions, can be found in the well written introduction of Cellier's book on continuous system modeling (37).

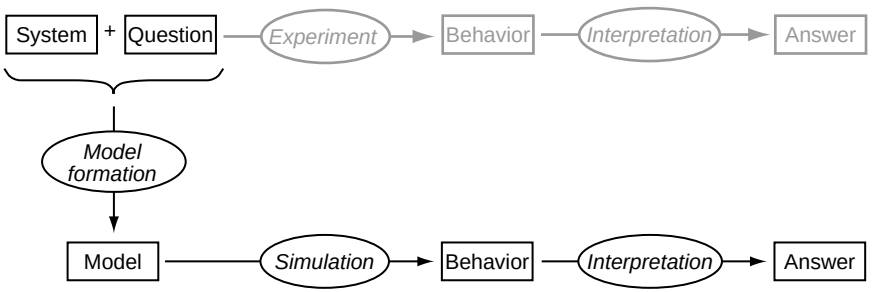


Figure 1.2. Simulation as a concept to eliminate information deficits relating a system.

of interest and/or the experiment to be performed on it. Cellier and Minsky count the experiment as a vital part of each model formation process.—Note however, that the interesting question may imply the instructions for the experiment, or the experimental frame as called by Zeigler (309, 310), and hence, the experiment may not be a mandatory source of information within the model formation process. Under the problem-solving view of Artificial Intelligence this is standard practice: The interesting questions (diagnosis or planning problems) prescribe the problem representation (the model) and yet the problem-solving method (the "simulation" algorithm) (214).

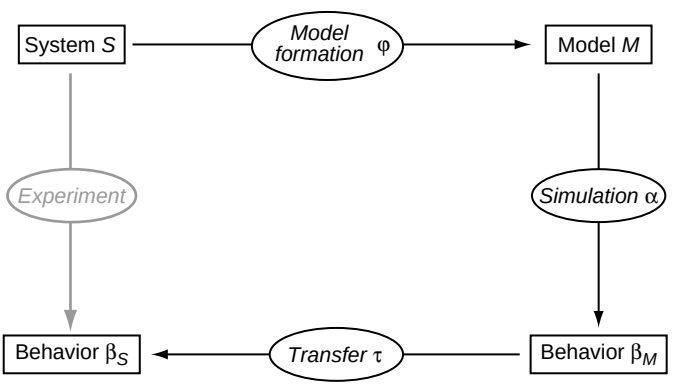


Figure 1.3. On the equivalence of system behavior, β_S, and model behavior, β_M.

Prerequisite for the use of models when reasoning about a system is that the functions depicted in Figure 1.3 can be instantiated such that the following relation holds.

$$\beta_S = \tau(\alpha(\varphi(S)))$$

where

β_S	behavior of the system S,
φ	formation of a model M for the system S,
α	simulation (analysis) of the model M,
τ	transfer of the model behavior β_M onto the system S.

Recall that the validity of the above relation cannot be of a generic nature for a system S; the functions τ, α, and φ are developed in compliance with the interesting question for S.

Creating Models of Technical Systems

Giving the gist of what Kowalk said, a system is a spatio-temporally closed and logically interconnected unit (152, pg. 27). A system, and in particular, a technical system consists of elements, called subsystems, and relations. The relations divide into internal relations, which connect subsystems, and external relations, which connect subsystems and objects beyond the system boundary, that is, objects from the environment.

Both subsystems and relations are characterized by attributes, called variables. Variables that do not change over time are called parameters; time-depending variables that are necessary to prescribe the system behavior are called states.

Variables that characterize external relations divide into input variables and output variables. Input variables are extraneous; they act from the environment on the system and modify the system's states. Input variables and state variables determine the system's output variables, which in turn affect the environment. Systems form modular building blocks: By unifying input and output variables of different systems, systems can be coupled to form a new system. The boundary of the new system is defined by the input and output variables that have not been unified.

Creating a model of a technical system S means to shape its underlying concept or idea—a process that is first of all mental, and that always involves three major steps.

(1) Identification of the system's boundary $\Rightarrow$ black-box model.

(2) Identification of the subsystems and relations of $S \Rightarrow$ structure model.

(3) Characterization of constraints between the variables of the subsystems and relations $\Rightarrow$ behavior model.

Identifying a system's boundary, Step 1, relates to the focusing. In the field of engineering, focusing is realized among others by the well-known Method of Sections. Identifying a system's structure, Step 2, relates to the modeling granularity. Within this step, the system's parameters along with its input, output, and state variables

are defined. Characterizing a system's behavior, Step 3, relates to the modeling accuracy or fidelity. Both modeling granularity and modeling fidelity define the modeling deepness, which is directly connected to the model's complexity. Goldschmidt writes in this connection:

> *"A model must render only those properties of a system that are necessary to answer the question of interest."*
>
> Goldschmidt, 1996, pg. 6

Note that a concise formulation of a question allows the use of a specialized model, which in turn is bound up with the risk that this model cannot be used to answer related questions. This tradeoff renders the creation of "adequate" models an artistic discipline.

The outlined model shaping steps happen in our mind, and a model at this stage is called a *mental model*. Gentner and Stevens argue that a mental model describes the way people understand some domain of knowledge (see 91, 17).

To communicate a mental model it must be given a representation. A mental model becomes representational as a physical or material model by craftsmanship. A mental model becomes representational as a symbolic or formal model by developing a prescription for the constraints, which are characterized in the third model formation step. Figure 1.4 classifies models with respect to their representational form.

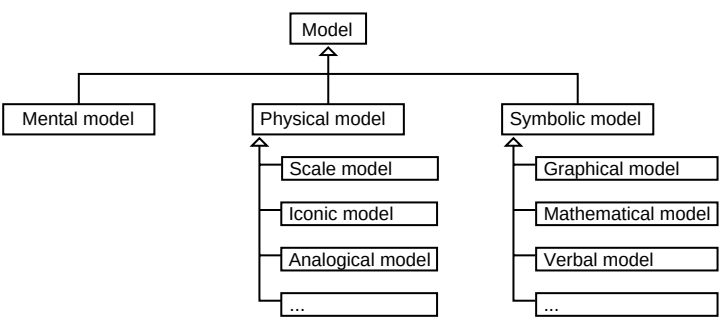


Figure 1.4. Classification of models with respect to their representational form.

Important physical models are scale models, iconic models, and analogical models. Scale models are investigated as part of an artificial environment, and scale-up knowledge or similarity rules are used to reason about the behavior of the system. Iconic models are used for illustrative purposes in first place. Analogical models exploit analogies between different scientific disciplines.

Important symbolic models are graphical models, mathematical models, and verbal models. Graphical models show subsystems along with their relations; they rep-

resent a structure model of the system. Examples are fluidic or electrical diagrams, signal-flow graphs, bond-graph diagrams, and mechanical structure sketches. Mathematical models comprise the total set of constraints that are defined over the system's variables. In connection with technical systems mathematical models are also called behavior models (39, 54). Verbal models describe a system by means of textual information.

Remarks. Using the representational form of a model establishes the most familiar approach to model classification. However, other approaches to classification exist; they rely upon the model purpose and differentiate between explanation models, optimization models, and forecasting models (121, 206), upon the model accuracy and differentiate between quantitative models and qualitative models (23, 43), upon the number of inputs and outputs, or upon the type of the behavior specification (253, pg. 147). The unified modeling perspective that is presented in Section 2.1 is grounded on the distinction between structural and behavioral information contained in models.

1.2 Model Construction (I)

We use the term model construction as a collective term for all kinds of processes where a given model is transformed into another model. Model construction takes very different forms. The most common situation where we encounter model construction is the transformation of an abstract model, which is close to the human understanding, into a computer model, that is to say, a program. The execution of this program represents an experiment at a symbolic model, and hence is a simulation.[2]

A model construction process that maps a model M onto a less abstract model is also called top-down model construction, model synthesis, or model instantiation (151); it is strongly related to system analysis (143). Model construction processes that work bottom-up are not treated here; they play an important role in reverse engineering and identification tasks (cf. 310, pg. 12).

The transformation of a mental model into a computer model usually happens in several steps wherein intermediate models are constructed: a structure model, a behavior model, and an algorithmic model (see Figure 1.5, which shows a modified version of Wallaschek's presentation (293)).

Typically, the human designer materializes his mental model as a textual or graphical structure model, for instance as a drawing or a diagram representation of the system S. The structure model, which defines subsystems and relations of S, becomes a behavior model of S by specifying a behavior prescription for each subsystem. Since behavior descriptions are often defined in a mathematical form, behavior

[2] In Section 2.3, Page 40, a precise and more generic specification of the term "simulation" in connection with mathematical models is given.

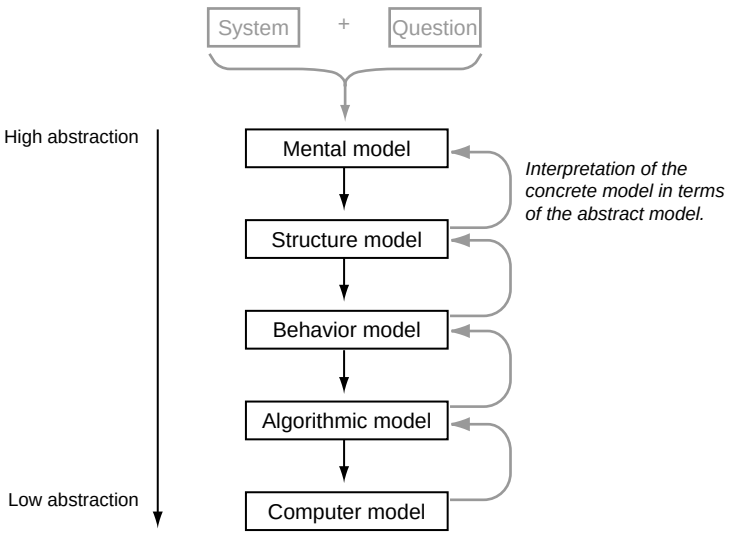


Figure 1.5. A hierarchy of models. Top-down model construction means to map abstract models onto less abstract models. Final objective is the operationalization of the mental model in the form of a computer program.

models are also designated as mathematical models (293). In the majority of cases the behavior model must be prepared with respect to the simulation algorithms employed. This step includes the introduction of a causal order for the behavior equations, the normalization of behavior equations, or other algebraic operations. The specification of an algorithmic model within a computer language or a simulation language like ACSL (185) yields a computer model.

Remarks. Moving down the model hierarchy means going in structure-to-behavior direction and is a process of increasing concretization. Model construction processes of this type resemble Zeigler et al.'s so-called association mappings (cf. 310, pg. 295). Note that model construction needs not to be a vertical process; the next but one subsection, starting on Page 11, shows that model construction can also be a mapping between models at the same level.

Model Construction Support

To support model construction many approaches have been developed. Most of them are top-down; they aim at a reduction of the distance between abstract models and less abstract models. Ideally, a mental model should be translated automatically into a computer program.

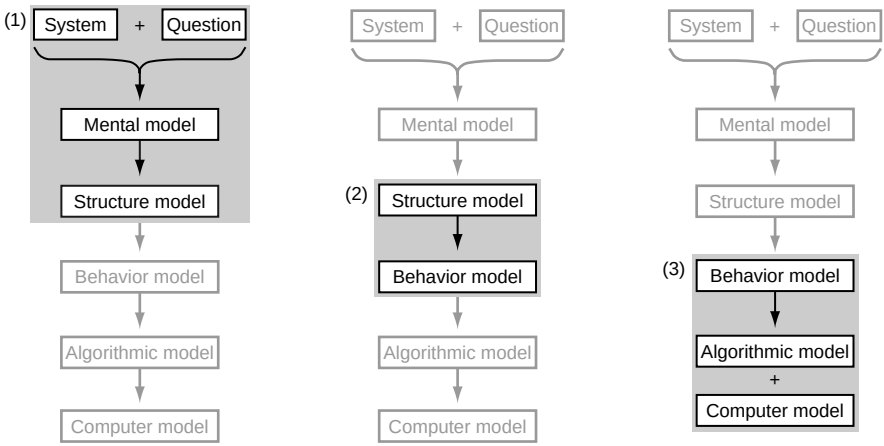


Figure 1.6. The shaded areas indicate the three major places at which top-down model construction is supported (from left to right): System boundaries and relevant phenomena, model deepness and model coherence, model synthesis and model processing.

Top-down approaches in the field of model construction belong to one of the following three classes, which are also illustrated in Figure 1.6.

(1) *System + Question → Mental Model → Structure Model.* Support at this place relates to the model construction steps 1 and 2, stated on Page 6. Creating a mental model includes the problem of phenomena selection, which in turn is closely connected with the problem of identifying adequate system boundaries. Note that no sharp line may be drawn between mental models and graphical structure models (293).

Known representatives are: The model fragment idea for the organization of phenomena by Nayak and Fikes et al. (193, 194, 195, 75), the reasoning on hypothetical scenarios for the determination of system boundaries by Rickel and Porter (225), the CFRL paradigm from Iwasaki and Levy that maps from function onto behavior (118, 119, 287), the approaches to ontological reasoning from Sasajima et al. (235) or Liu and Farley (163), the case-based design approaches that map from demand specifications onto technical devices (96, 154, 169).

(2) *Structure Model → Behavior Model.* Support at this place relates to the model construction steps 2 and 3, stated on Page 6. Creating a behavior model means to select and compose local behavior models from a given domain theory according to a structural description of a system.[3]

Known representatives are: The model composition approach developed by Falkenhainer and Forbus and its variants (71, 27), the graphs of models

[3] In Section 2.1, Page 31, a definition of local behavior models is given.

paradigm from Addanki et al. (5, 191), the use of model hierarchies in the field of diagnosis (4, 40, 276), the selection of local behavior models with respect to the desired accuracy for predefined phenomena (279).

(3) *Behavior Model → Algorithmic Model + Computer Model.* Support at this place relates to one of the following tasks: Synthesis of an algorithmic model from a given set of local behavior models, generation of normal forms required for numerical algorithms, behavior model processing.

Known representatives are: Continuity and compatibility constraint introduction as realized in FLUIDSIM and standardized in MODELICA (65, 38), task-oriented selection of efficient numerical procedures (47), symbolic manipulation of equation systems such as BLT-partitioning or automatic tearing (164), minimum coordinate determination in kinematics computations, coherent synthesis of local behavior models (120, 101, 160, 259), automation of Jordain's principle as realized in AUTOLEV (129, 2).

Remarks. The operationalization of model construction knowledge at the mental model level, (1), does not hit the mark in many modeling problems. Especially when diagnosing or configuring an existing system, one is rather concerned with aspects of adequacy and efficiency. Phenomena selection is a worthwhile objective, though, but it is usually too challenging to be handled by a computer at present.

Model Construction Based on Source Models

Let a system, S, and a question about S be given. The contributions of this thesis ground on the observation that in many cases some well-tried model M of S exists, which is, possibly, a remnant from an earlier problem solving task.

The thesis investigates in which ways from an existent model M, called the *source model*, a new model M' can be constructed that is suitable to solve a particular analysis or synthesis problem (see Figure 1.7). For instance, the structure information that can be derived from a local behavior model may form the base when tackling a related configuration problem, or, by evaluating a complex behavior model at selected points in time a heuristic model for diagnosis or control purposes can be derived.

We have organized the model construction methods that have been applied in this thesis within the following groups.[4]

- *Model Refinement.* Adapting a model by filling gaps.
- *Model Simplification.* Coarsening a model by stripping off unused parts.
- *Model Compilation.* Making a model more efficient by introducing shortcuts.
- *Model Reformulation.* Switching to another modeling paradigm.
- *Model Envisioning.* Providing insights by rendering the model structure.

[4] In Section 2.4, Page 42, this classification is introduced in greater detail.

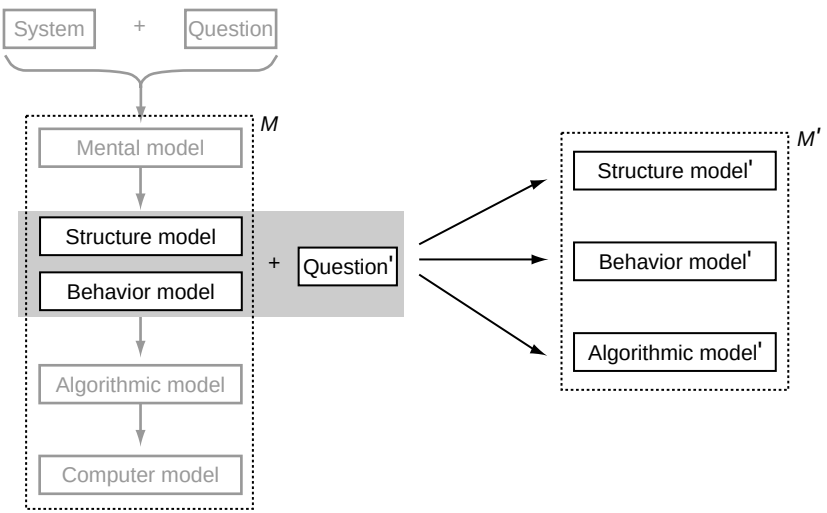


Figure 1.7. Utilization of a source model to construct an adequate model regarding some question. Question′ designates an analysis or synthesis task; the shaded area indicates the underlying knowledge source.

Remarks. (1) While model construction support as depicted in Figure 1.6 means moving down the model abstraction hierarchy, does model construction that is based on source models establish a horizontal mapping in most cases. Model construction processes of this type resemble Zeigler et al.'s so-called morphisms (308, 310). (2) Our observations are made from a perspective which is also designated as "metamodeling" (93, 285): The focus is on the modeling process itself.

If a source model, M, guides model construction, M establishes the primary knowledge source, say, pattern when building the new model M'. Anyway, additional knowledge sources, which give answers to the following questions, are inevitable.

- *Model Refinement.* What is to be filled into the gaps?
- *Model Simplification.* Which parts shall be stripped off?
- *Model Compilation.* Where is a hidden shortcut?
- *Model Reformulation.* How does the migration rule looks like?
- *Model Envisioning.* Whereby is the model structure characterized?

The nature of the additional knowledge sources unveils that instances of these model construction approaches are much more specialized with respect to the domain and to the question of interest then are the top-down approaches of the previous subsection. This in turn means that we cannot expect recipes for horizontal

model construction: Model compilation, for instance, designates a construction *principle* rather than a construction *method*.

Most of the case studies presented in Part II of this thesis contain new ideas for model construction, which pertain either to conceptual aspects or to the realized algorithms. These solutions are organized according to the above classification scheme and may serve as a pool of ideas when tackling new analysis and synthesis tasks.

How Model Construction is Operationalized

Recall the three main places where top-down model construction can be supported, shown in Figure 1.6. Automation in this connection means to derive

(1) necessary phenomena from the question of interest,
structural information from necessary phenomena,
structural information from the desired function,

(2) behavioral information from necessary phenomena,
behavioral information from functional requirements,
behavioral information from structural information,

(3) algorithmic information from behavioral information.

Following this view, top-down model construction methods can be characterized as

(1) *structure-defining* methods,

(2) *structure-filling* methods, and

(3) *structure-propagating* methods.

This classification says a lot about how these methods work. Structure-propagating methods, (3), is a collective term for mathematical procedures that form a global algorithmic model by "propagating" the constraining implications of the local behavior models along a given structure. Since structure-propagating methods operate on the mathematical model level, by far the largest part of them is domain-independent and task-independent.

Structure-defining methods, (1), as well as structure-filling methods, (2), are based on libraries of device models, component models, or model fragments. The models in the libraries describe systems, building blocks, or phenomena from a particular domain.[5]

[5]Depending on the context, these libraries are also called case-base, taxonomy, or ontology (166, 5, 296).

Example. A model of an analog lowpass filter is a device model from the electrical engineering domain. A resistor of this filter is a component, and one of its model fragments is given by Ohm's law, $v(t) = R \cdot i(t)$; another model fragment may take the resistivity depending on the temperature, T, into account. In this case the model fragment consists of two additional equations, $R = \rho \cdot l/A$ and $\rho = \rho_0(1 + \alpha(T - T_0))$, where l, A, ρ_0, and α designate the resistor length, the cross-sectional area, the resistivity at T_0, and the coefficient of the resistivity respectively.

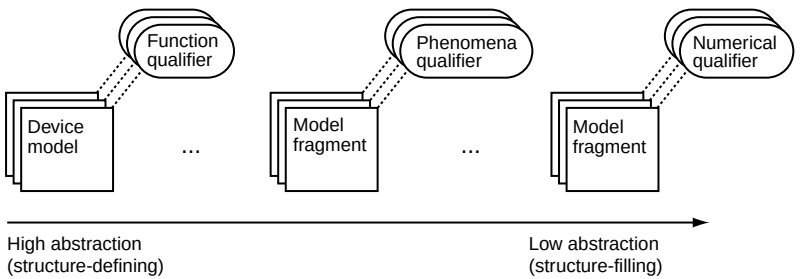


Figure 1.8. Libraries with different kinds of models and qualifiers. Device models (left), as used in case-based design, define the structure of an entire system; depending on the qualifier, model fragments can play the role of both structure-defining (middle) and structure-filling (right).

The models in the libraries are tagged by qualifiers that encode knowledge about their applicability (see Figure 1.8). Generally speaking, the job of structure-defining and structure-filling methods is the selection of models from libraries by processing the qualifier constraints of the chosen models. The qualifiers can be of very different forms:

- In case-based design the qualifiers encode an abstracted functional description of devices. The job of a case-based reasoning system is to find for a given functional demand the best fitting case from the device model library (76, 154, 215, 168). Qualifier processing here relates to the computation of similarity measures and to case retrieval (82, 218).

- Model fragment qualifiers as used by Nayak (193, 195) or by Iwasaki and Levy (119) encode relations amongst phenomena, typically formulated within propositional logics. Given two model fragments, m_1, m_2, the contradictory relation states whether m_1 and m_2 go along with each other; the approximation relation, on the other hand, states whether m_1 is a more approximate description of some phenomena compared to m_2. Additionally, coherence constraints and precondition constraints are used to define domain-dependent modeling restrictions. Qualifier processing here aims at the synthesis of device models that are causal, coherent, and minimum. This synthesis problem is in NP (195, pg. 69).

- Model fragment qualifiers are also employed to encode mathematical properties or hints respecting behavior processing. This includes knowledge about signal ranges, model linearity, model stiffness, system orders, numerical conditioning, steady states, oscillation behavior, damping, processing difficulty, and processing performance. Pos et al. (212) use this kind of knowledge under the name "modeling assumptions". Qualifier processing here means assumption management.

Remarks. Aside from tagging models absolutely, by means of qualifiers, also a relative classification of models with respect to their applicability is reasonable. Case-based design does this in a natural way: A similarity value can be computed for any two device models of a case library, thus defining an order between model candidates for a demanded functionality. Addanki et al.'s approach points in a similar direction: He proposes libraries that organize models and model fragments as complete graphs. The edges, which link the models, are grouped into priority classes and can be interpreted as a model distance measure (cf. 5, pg. 1433).

Methods for model construction based on source models, which realize the horizontal mapping shown in Figure 1.7, transform a structure model, M_S, into another structure model, M'_S, and a behavior model, M_B, into another behavior model, M'_B. Anticipating the notation of the next chapter, we establish this form of model construction as two functions, γ_S, γ_B, which map from the source model onto a new model:

$$M_S \xrightarrow{\gamma_S} M'_S, \quad M_B \xrightarrow{\gamma_B} M'_B$$

Note that the functions γ_S and γ_B can describe both a slight modification and a fundamental reformulation of the source model.

As opposed to structure-defining and structure-filling methods, the functions γ_S and γ_B cannot be classified in a uniform way. This is in the nature of things. Nevertheless, we have developed a formalism that enables us to specify a wide range of structure model mappings γ_S, the so-called *design graph grammars*.

Design graph grammars possess the flexibility to model even very specific kinds of domain knowledge while still providing a broadly understood semantics. By applying graph transformation rules

- a structure model can be analyzed,
- an incompletely and coarsely defined structure model can be completed and refined, and
- even the reformulation of a structure model with respect to another paradigm becomes possible.

Figure 1.9 gives such a reformulation example from the field of theoretical communications engineering: A series-parallel decomposition of an electrical circuit is

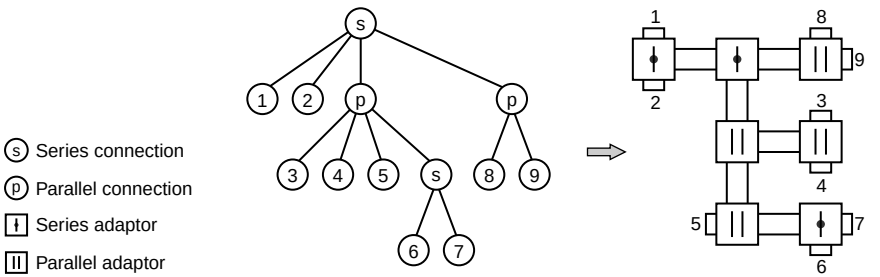


Figure 1.9. A series-parallel decomposition tree (left-hand side) is transformed into a so-called adaptor structure (right-hand side).

transformed into an adaptor structure.[6] The related design graph grammars consists of two structure-modifying rules. Design graph grammars are introduced in Chapter 3.

Remarks. The largest group of commercial tools that support model construction concentrate on structure filling; moreover, they are restricted to situations where the model deepness has been predefined by the human designer.

An important role for the effectiveness of such tools comes up to the user interface. It can provide powerful graphical support and be close to the mental model of the human designer, for example as realized within FLUIDSIM: Based on CAD drawings of even large and complex electro-fluidic systems, FLUIDSIM generates the related algorithmic models without any human support (262). AUTOLEV (236) and the IMECH toolbox (11) are tools for textually describing mechanical multibody systems. Models in AUTOLEV are formulated within a proprietary mathematical language, models in the IMECH toolbox are formulated by instantiating C++ objects. A comparison of these tools can be found in (97).

Because of its popularity the SIMULINK toolbox is worth to be noted here. Although SIMULINK comes along with a fully-fletched graphical interface, it does not provide model construction support at one of the mentioned places. Working with SIMULINK means to specify algorithmic models manually, by drawing block diagrams.

[6] Section C.1 presents this instance of a model reformulation task in greater detail.

1.3 Synthesis Tasks

The given exposition respecting "model formation" and "model construction" is perfectly adequate when treating analysis tasks, like diagnosis or simulation. Within such tasks the questions of interest relate to a single system. This is not the case for synthesis tasks and its most prominent representative: system design. Here we are confronted with questions that relate to a *set* of systems.

This section addresses this shortcoming. It recapitulates the concepts of the preceding sections and interprets them from the standpoint of system design.

Motivation and Disambiguation

Our starting point is characterized as follows. We are given a set of systems, $\mathcal{S}$, called the system space, along with an open question. The question may ask whether a system with a desired functionality exists, say, is member of $\mathcal{S}$, or how much a system with a desired functionality at least costs.

Answers to such questions can be found by designing the desired system and analyzing its functionality (see Figure 1.10).

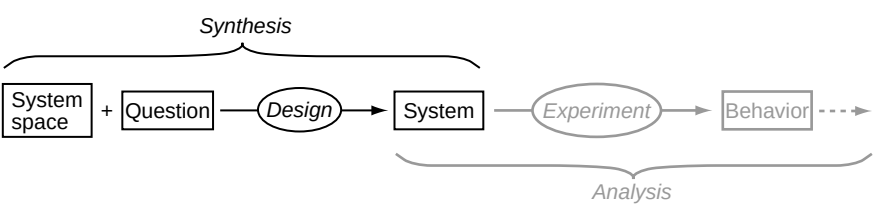


Figure 1.10. A design task defines a set of systems, called the system space, as well as a question that asks whether a system of a certain functionality does exist. An answer can be found by designing and analyzing the system.

Clearly, designing a system solely for answering an open question cannot be accepted in the very most cases. A way out is the creation of a bijective function, φ, that maps each system $s \in \mathcal{S}$ onto a model in a *model space* (see Figure 1.11). Usually the model space is described intensionally, by means of a finite number of combinable objects along with operations that prescribe how objects can be connected to each other. Put in a nutshell, design problems are solved by developing a systematic search strategy that turns the model space into a search space.

Note that the prerequisite for the use of models when reasoning about systems, which is illustrated in Figure 1.3 on Page 5, remains untouched. We claim that functions τ and α can be instantiated such that the following relation holds.

$$\beta_S = \tau(\alpha(\varphi(S)))$$

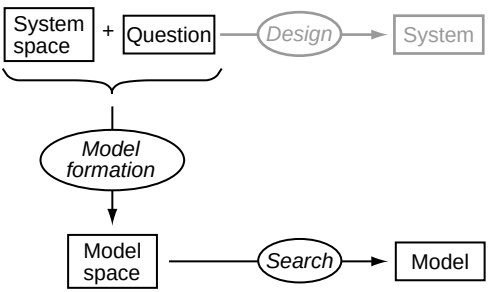


Figure 1.11. Design problems are solved by means of a systematic search in the model space—provided that a bijective mapping from the system space onto the model space can be stated.

where

β_S	behavior of the system S,
φ	mapping between system space and model space,
α	simulation (analysis) of the model M,
τ	transfer of the model behavior β_M onto the system S.

Creating a model space means to capture the underlying concepts of the interesting system space—a process that, again, is first of all mental, and that involves three major steps.

(1) Identification of possible system building blocks ⇒ subsystems.

(2) Identification of system construction principles ⇒ system space.

(3) Mapping of the system building blocks and construction principles onto objects and connection operations ⇒ model space.

Identifying the possible system building blocks, Step 1, relates to *destructuring*. Identifying the construction principles, Step 2 requires a deep understanding of the domain and the allowed design principles. The definition of a mapping φ, Step 3, is a challenging model formation problem. Aside from modeling system building blocks also the synthesis operations upon these building blocks have to be translated in the model world.

The outlined process happens in our mind, analog to the model creation process on Page 6, and a model space at this stage is called a *mental model space*. To communicate a mental model space it must become representational, as a physical or a symbolic model space. Figure 1.12 classifies model spaces with respect to their representational form.

The different model spaces develop from the different model representations in a natural way, by providing a facility to work on *sets* of models. A CAE-system, for instance, provides a facility to work on sets of graphical models, while a virtual prototyping system provides a facility to work on sets of behavior models.

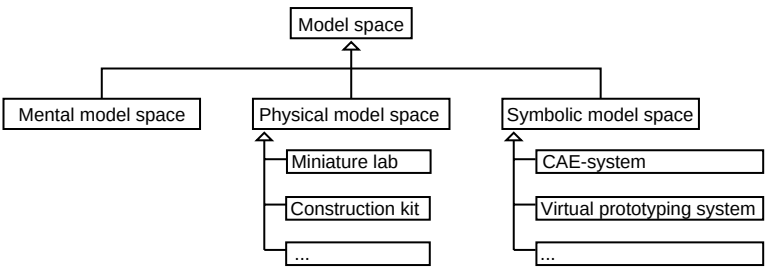


Figure 1.12. Classification of models spaces with respect to their representational form.

Model Space Construction

Note that a model space is no end it itself; it serves as a collection of candidate models from which one model has to be selected. This model represents the solution of the design task. Also note that the selection process may happen at any level of abstraction, at the level of the mental model space, at the level of the structure model space, or at the level of the behavior model space. If, for instance, the solution is searched in the mental model space, the subjacent levels will never come into being.

The construction of a model space for a given system space is a creative job. It requires both experience and profound technical skills in the respective domain and is done by a human designer.

Depending on the model space structure also the search within a model space can be creative job—however, exactly at this place computer assistance is brought into play. Prerequisite is that the mental model space is transformed into an algorithmic model space. A computer program that encodes an algorithmic model space along with an efficient search strategy is called design system, configuration system, expert system for design, or expert system for configuration (257, 117).

The model space grows exponentially in the number of objects and connection operations. The human designer, who constructs the model space, will hence shape the models in the model space as simple as possible. The consequence is not surprising: Design problems are often tackled at the level of structure models or at the level of simplified behavior models (34, 92, 242).

Computer systems for configuration or design are distinguished by the type of constraints that the knowledge engineer can use to materialize his mental model space. They transform either a structure model space, a behavior model space, or a combination of both into an algorithmic model space (see Figure 1.13).

(1) *Structure Model Space → Algorithmic Model Space.* Enable the task-adequate formulation of structure constraints. Structure constraints specify global synthesis knowledge and fall into the following classes: Compositional "part-of" constraints, which define the skeleton of the system to be designed (cf. 213,

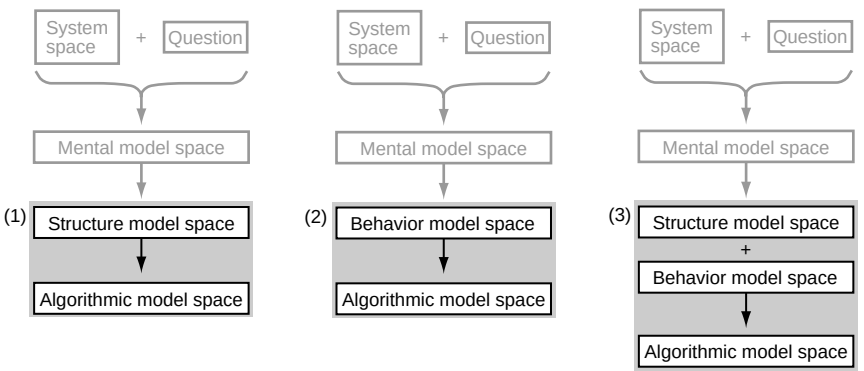


Figure 1.13. The shaded areas indicate the starting points from which model space construction is realized within design systems.

pg. 137); taxonomic "is-a" constraints, which are used to formulate functional hierarchies (187); spatial constraints, which define arrangement and place restrictions (239); and associative constraints, which is a collective term for other structure constraints between submodels. The design graph grammar approach developed by Stein and Schulz (268) provides an operational semantics to define all types of structure constraints.

(2) *Behavior Model Space → Algorithmic Model Space.* Enable the task-adequate formulation of behavior constraints. Behavior constraints that can be mastered in design tasks are resource-based constraints (105, 272) and local value propagation schemes (102, 116). More complex behavior constraints are conceivable but must be specialized with respect to a concrete task, as realized for instance in systems for industrial mixer design or elevator design (33, 172).

(3) *Structure Model Space + Behavior Model Space → Algorithmic Model Space.* Enable the formulation of structure models in combination with simple behavior constraints, as realized for instance in the configuration platform PLAKON (46). Also case-based design, if implemented according to the case-adaptation philosophy, is a representative of such a hybrid approach (95, 110).

Remarks. Overviews on design systems and their underlying theory can be found in (117, 256, 189) and in the annual workshop proceedings of the GI expert group 1.5.3, "Planning and Configuration".

Recall that the main focus of this thesis is model construction. Model construction and model *space* construction are strongly related to each other: The horizontal model construction principles, presented on Page 11, establish a natural and powerful approach to model space construction. When working on a synthesis task, the

application of these principles to a given source model will have the following impacts on the respective model space.

- *Model Simplification.* The model space becomes smaller, say, tractable.
- *Model Compilation.* Search in the model space is speeded up.
- *Model Reformulation.* The model space is mapped onto another model space.

The case studies of the Sections A, B, and C exploit these effects.

1.4 The AI Point of View

Research in Artificial Intelligence (AI) produced strategies and methods to solve analysis and synthesis tasks. To which extent answer these developments questions of modeling and model construction?

This section comments contact points and differences between AI developments and our work. In particular, we will discuss the concept of problem solving methods and sketch out their basic architecture. It is not intended to present an in-depth discussion on all related AI developments; this clearly goes beyond the scope of this section and, in fact, is not necessary to motivate the contributions of our model construction ideas.

Problem Classes and Problem Solving Methods

Problem classes provide a scheme for the characterization of knowledge-intensive tasks. The idea of a classification scheme was firstly introduced by Hayes-Roth et al. (104) and results from the following wishful thinking: When given a particular problem Π, a solution for Π could be easily constructed if we are given a look-up-table that associates problems with methods solving them. The latter are called *problem solving methods*.

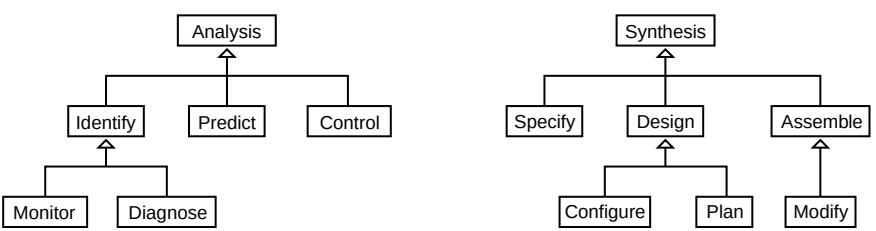


Figure 1.14. The hierarchy of problem classes according to Clancey (41).

Figure 1.14 shows Clancey's hierarchy of problem classes (41), which is a substrate of the work of Hayes-Roth et al. (104). Clancey pursued 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, 1985, pg. 315

Remarks. Other, less popular criteria for the classification of problems have been developed. They are based on generic problem solving methods (20), on the type of the model realized within a system (257), or on the domain.

The term problem solving method designates a procedure, a concept, or an algorithm that is employed to tackle a clearly defined problem. Bauer et al. distinguish between three types of problem solving methods (20): Methods that specify a conceptual procedure for knowledge processing like HEURISTIC CLASSIFICATION or HYPOTHESIZE AND TEST, role-limiting-methods (67, 176) that "merely" need to be filled with domain knowledge like PROPOSE AND REVISE or COVER AND DIFFERENTIATE, and methods that realize basic knowledge processing techniques like FORWARD CHAINING or FUZZY INFERENCE.

Additionally, a distinction between "strong" and "weak" is often made (176, 213). These attributes do not characterize a method's efficiency but indicate its range of application. A weak problem solving method is less specialized and hence, it can be used for a wide range of knowledge processing tasks. A strong problem solving method, on the other hand, is tailored towards a problem-dependent task. Due to its specialization its range of application is narrow, but it can provide far-reaching knowledge acquisition support.

Remarks. To put it overstated, if the problem-solving-method idea worked for all kinds of knowledge-intensive tasks, model construction would be a thing of the past. Obviously this is not the case, and despite of this fact—or just because of it—it makes sense to shed light on the architecture of problem solving methods.

Problem Solving Methods and Model Construction

One of the furthest developed lattices of problem solving methods comes with the KADS models of interpretation (32, 130, 292). KADS, which is an abbreviation of "Knowledge Acquisition and Design Structuring", has been developed from 1985 to 1989; it provides a methodology for the model-based design of knowledge-based systems (300). A central role within the KADS methodology play the so-called conceptual models, which are used to specify problem solving expertise within four layers: strategic, task, inference, and domain. The upper three layers form the domain-independent interpretation model.

Interpretation models assign roles to problem solving knowledge (called meta-classes) and an inference structure between them (called knowledge sources). For

instance, the inference structure for the problem solving method HEURISTIC DIAGNOSIS, which actually is the underlying diagnosis approach of MYCIN (41), looks as depicted in Figure 1.15.

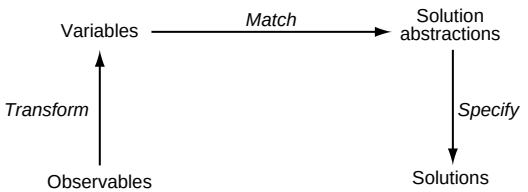


Figure 1.15. The inference structure of the problem solving method HEURISTIC DIAGNOSIS.

The metaclasses of the inference structure are: observables, variables, solution abstractions, and solutions. The knowledge sources are: transform, match, and specify. Each knowledge source is characterized by its inferential semantics, its input and output (the incident nodes in the inference structure graph), the method that realizes the inference, and maybe support knowledge. To get an idea of the explicitness of interpretation models, the inferential semantics of the knowledge sources employed in HEURISTIC DIAGNOSIS is specified in Table 1.1.

Knowledge source	Inferential semantics
Transform	Data abstraction: System data are transformed into variables
Match	Abstracted data are matched by direct association
Specify	Solution abstractions are refined into specific solutions

Table 1.1. The inferential semantics of the knowledge sources in HEURISTIC DIAGNOSIS.

Of course more complex, but with the same constructional means, interpretation models for various knowledge-intensive tasks have been developed as part of the KADS project. They cover tasks for all problem classes shown in Figure 1.14.

Another substantial collection of problem solving methods has been compiled by Puppe (213). Compared to the KADS models of interpretation, his specification of problem solving methods is at a less abstract level; it is grounded on the three pillars knowledge representation, knowledge manipulation, and knowledge acquisition.

Regarding the method HEURISTIC DIAGNOSIS the elements of the knowledge representation component are: symptoms, symptom abstractions, solution classes, solutions, and rules. Puppe describes these elements verbosely and pretty close to the level of data types. Placed at this level, an algorithmic specification of the desired inference task forms the gist of the knowledge manipulation component.

Remarks. Problem solving methods can unfold a high inferential power on knowledge intensive tasks. With respect to inference and modeling flexibility, they represent an

advancement of the classical expert system paradigm. They specify the involved kinds of knowledge and clearly assign roles that the kinds of knowledge play in the inference process. On that account, problem solving methods have the potential to bridge the gap between knowledge engineering and software engineering—a fact that has been proven time and again.

Straight out, one should not expect that the conception of well-defined problem solving methods will contribute a great deal to tasks such as model construction. Model construction requires, in the very first place, the determination of an adequate modeling level, and secondly, not less demanding, the formulation of a human's perception of the interesting system at this modeling level.

Both jobs allow of no solution pattern but require a sense of the underlying domain, experience, and creativity instead (296, 3). Model construction is by far more than knowledge acquisition and the assignment of knowledge roles. Problem solving methods, on the other hand, come into play if both an adequate modeling level is found and a model of the system (not of the problem solving process!) is already constructed. A glance at the above example shows that.

Remember that at model construction time one may be confronted with following and other situations.

- The model is incomplete with respect to certain system properties.
- The model of the system is too complex.
- The model is in a disadvantageous representation.
- The model cannot answer the interesting question.

The reality is by far too complex to solve problems of this type in a recipe-like manner, and the structure of the work in hand reflects this matter of fact: The model construction methods mentioned on Page 11 designate principles rather than methods; the case studies in Part II exemplify the operationalization of these principles.

2

A Framework for Model Construction

The previous chapter gave a discussion on types of models, the construction of models, and related aspects. In this chapter we concentrate on models of modular technical systems and draw up a framework for both the description and the construction of such models. The framework is powerful enough to become applied to a variety of domains and modeling scenarios; however, a great deal of attention has been paid to keep it as clear and simple as possible.

Our view to models and model construction is bivalent: it is oriented at structure and behavior (see Figure 2.1).

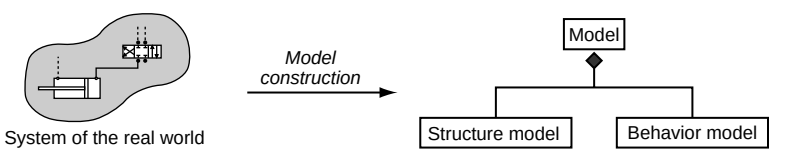


Figure 2.1. Bascially, model construction divides into structural and behavioral considerations.

A structure model renders the structural or topological setup of a system, a behavior model reproduces, in extracts, a system's behavior.[1] In this connection, model formation relates to both structural composition and behavior specification. Similarly, the construction of a new model from a given source model can take effects onto the source model's structure, by means of adding or removing model constituents, but also onto the source model's behavior, by changing the focus or the resolution of the interesting phenomena.

The next section provides for a formal basis of structure models and behavior models. Moreover, by specifying the concepts of locality, causality, and no-function-in-structure, the interweaving of structure and behavior becomes quantifiable (see Figure 2.2). Section 2.2 and 2.3 engage into different behavior model types and simulation approaches. Finally, Section 2.4 introduces, isolated from a specific problem and a concrete domain, five different concepts to model construction.

[1] In other places, the term structure is also used to specify non-topological aspects of a system.

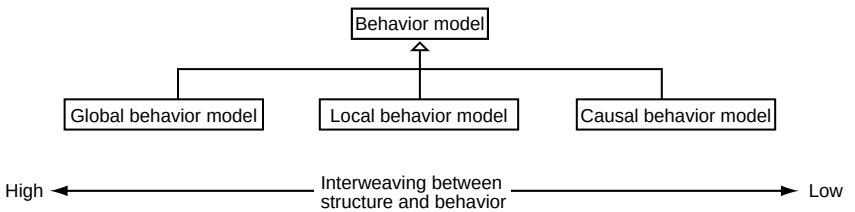


Figure 2.2. Depending on its degree of interweaving with the structure model, a behavior model is called global, local, or causal.

2.1 A Unified Modeling Perspective

Let S be a system of the real world. A model of S is a restriction of S with respect to a set of interesting *functionalities*, attributes, or properties. If the model is composed from submodels, which should be the normal case when dealing with modular technical systems, the functionalities are distributed amongst these submodels in some way.

Each such distribution of functionalities prescribes a frame wherein structure models and behavior models can be formed. This observation motivates our following plain—but clear and universal—definition of the term model (see 99 and also 31, pg. 47).

Definition 2.1 (Model, Graph of a Model) *A model is tuple $\langle F, \mathcal{M} \rangle$, where F is an arbitrary, finite set, called the set of interesting functionalities or properties. $\mathcal{M}$ is a finite collection of nonempty sets, $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$, $M_i \subseteq F, i = 1, \ldots, n$. The $M_i \in \mathcal{M}$ are called submodels, objects, or components.*

A multigraph $G = \langle V, E \rangle$, $V = \{1, \ldots, n\}$, is called a graph of the model $\langle F, \mathcal{M} \rangle$, if the following holds.

(1) $|M_v \cap M_w| \neq \emptyset \Rightarrow$ E contains $\{v, w\}$ with frequency k, $1 \leq k \leq |M_v \cap M_w|$

(2) G is called the complete graph of a model, if $|E| = \sum_{\substack{v=1,..,n \\ w=1,..,v}} |M_v \cap M_w|$.

Remarks. (1) The edges $E_{M_v} \subset E$ that are incident to $v \in V$ are called the terminals of submodel M_v. (2) An edge $e \in E$ may be directed; it then is written as an ordered set (v, w) with tail v and head w. (3) $|F|$ constitutes a relative measure for the models accuracy or fidelity; it corresponds to the term "resolution" as used by Zeigler et al. (see 310, pg. 330). (4) $|\mathcal{M}|$ constitutes a relative measure for the model's granularity and corresponds to Zeigler et al.'s term "size". (5) The "product" of fidelity and granularity, $|F| \times |\mathcal{M}|$, is called the *analytic complexity* of a model (see also 310).

Example. The definition is illustrated at the small electrical circuit of Figure 2.3, a series connection of a voltage source, a resistance, and a capacitance. Please note that

this example shall demonstrate the *syntax* of our model definitions in first place; an interpretation from an engineering point of view is allowed but not necessary.

- *Functionalities.*
  $F = \{R, C, e, i_1, i_2, i_3, \phi_1, \phi_2, \phi_3\}$.
- *Submodels.*
  $\mathcal{M} = \{M_1, M_2, M_3\}$, where $M_1 = \{R, i_1, i_2, \phi_1, \phi_2\}$, $M_2 = \{C, i_2, i_3, \phi_2, \phi_3\}$, and $M_3 = \{e, i_1, i_3, \phi_1, \phi_3\}$.
- *Graph.*
  $G = \langle V, E \rangle$, where $V = \{1, 2, 3\}$ and $E = \{\{1, 3\}, \{1, 3\}, \{1, 2\}, \{2, 3\}, \{2, 3\}\}$. The right-hand side of Figure 2.3 shows this graph of $\langle F, \mathcal{M} \rangle$.

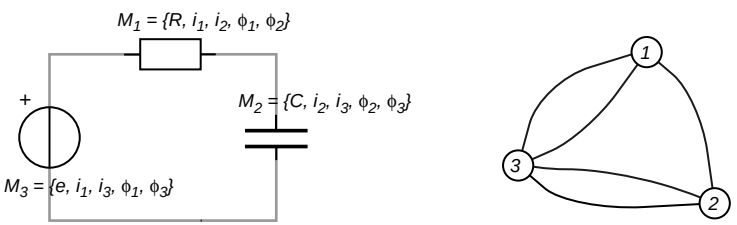


Figure 2.3. A model consisting of the three submodels M_1, M_2, and M_3 (left-hand side). The right-hand side shows an incomplete graph—out of 8 possible graphs—of this model.

Observe that each submodel of a model $\langle F, \mathcal{M} \rangle$ corresponds to a node in a graph G of $\langle F, \mathcal{M} \rangle$. Also note that each functionality that is shared amongst two submodels, M_v, M_w, defines some kind of *interaction path* between M_v and M_w. This way, G can be regarded as a structure description of a model. The graph G is a particular kind of *intersection graph*, and, in this connection, the set F is also called *host* (100).

Typically, both the submodels and the interaction paths between submodels are subject to *categorization*, which forms the basis to define equivalence classes amongst the submodels. For instance, all resistances in an electrical circuit are indistinguishable from a structural point of view; they have two terminals each of which is of type conductive.

Since labeling provides a sufficient means to define building blocks within a graph of a model, a categorization can be realized by means of labeling. The following definition introduces the concept of a *structure model* by restricting the labeling of a graph with respect to the building block metaphor: Two submodels of a model $\langle F, \mathcal{M} \rangle$ can get assigned the same label only if they provide the same interaction paths.

Definition 2.2 (Structure Model) *Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E \rangle$ be a graph of this model. Moreover, let Σ be a set of node labels and edge labels, and let $\sigma : V \cup E \to \Sigma$ be a labeling function.*

$\langle V, E, \sigma \rangle$ is called a structure model over $\langle F, \mathcal{M} \rangle$ if the edges that are incident to two equally labeled nodes are of equal number, orientation, and have equal labels—stated precisely: Let E_v^+ and E_v^- denote the edges $(u, v) \in E$ and $(v, u) \in E$ respectively; furthermore let $\Sigma_v^+ = \{\sigma(e) \mid e \in E_v^+\}$ and $\Sigma_v^- = \{\sigma(e) \mid e \in E_v^-\}$ be multisets comprising the labels associated with E_v^+ and E_v^-. Then the following holds.

(1) $\langle V, E, \sigma \rangle$ *is a structure model* $\quad \Leftrightarrow \quad \big(\sigma(v) = \sigma(w) \;\Rightarrow\; \Sigma_v^+ = \Sigma_w^+ \;\wedge\; \Sigma_v^- = \Sigma_w^-\big)$

Remarks. (1) A structure model over $\langle F, \mathcal{M} \rangle$ is designated with $S\langle F, \mathcal{M} \rangle$; likewise, the set of all structure models over $\langle F, \mathcal{M} \rangle$ is designated with $\mathcal{S}\langle F, \mathcal{M} \rangle$. (2) The labeling function, σ, divides the sets of nodes and edges into classes of equal physical properties. (3) If the edges $e \in E$ are undirected, E_v^+ and E_v^- collapse to a single set E_v.

Example. Figure 2.4 (left and middle) shows two graphs of the circuit model where the components have been abstracted towards electrical dipoles. Since the equally labeled nodes coincide in both number and labels of their incident edges, these graphs represent structure models.

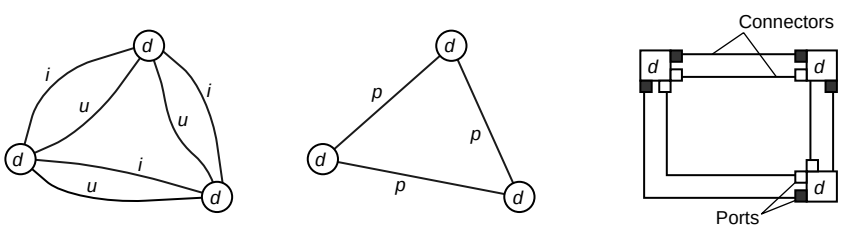


Figure 2.4. Two structure models of the circuit example (left and middle) and a port-and-connector model (right-hand side), which corresponds to the left structure model. Ports with an equal role are colored equally.

Structure models as introduced here form a theoretical basis for Stefik's so-called *port-and-connector models* (see 256, pg. 610). The ports on each submodel correspond to the different roles that submodels can play with respect to each other. Unlike the above structure model paradigm, submodels of the same type may provide different ports; moreover, Stefik restricts the port-and-connector methapor to a special abstraction level for configuration tasks.

A structure model can be represented as a port-and-connector model in a canonical manner: Between the nodes and edges of both representations a bijective mapping is defined. Moreover, two adjacent ports in the port-and-connector model get assigned the same role, which is prescribed by the edge label of the corresponding edge in the structure model (see Figure 2.4).

Structure models are a powerful means to define a model's composition space—without committing the level of abstraction at which a technical system is described.

Note that a structure model says nothing about the models type or purpose, whether it establishes a qualitative model, a dynamic behavior model, or some other model.

However, the functionalities of a model of a technical system constrain each other, and the underlying constraints are usually referred to as *behavior*. This point is addressed in the following definition.[2]

Definition 2.3 (Behavior Model) *Let $\langle F, \mathcal{M}\rangle$ be a model. A behavior model over $\langle F, \mathcal{M}\rangle$ is a tuple $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ whose elements are defined as follows.*

- *F_U, F_Z, F_Y are subsets of F, with $F_U \cap F_Z = \emptyset$. F_U, F_Z, and F_Y are called input variables, constraint variables, and output variables respectively. The functionality set $F_P = F \setminus (F_U \cup F_Z \cup F_Y)$ constitutes the set of component properties or component parameters. (If unmistakable the terms "functionality" and "variable" will be used as synonyms.)*

- *For each functionality $f \in F_U, F_Z, F_Y$, and F_P there is an arbitrary, possibly infinite set U_f, Z_f, Y_f, and P_f respectively, called the domain of f.*

  Moreover, for each $f \in F_U$ there is an additional domain, U_f^T, of partially defined functions in the parameter time, $U_f^T := \{u \mid u : T \to U_f, t \mapsto u(t)\}$. Depending on the model's time base, which may be continuous time, discrete time, or discrete event, T may be an interval from $\mathbf{R}^+$, an interval from $\mathbf{N}$, or a linearly ordered finite set.

  $\mathcal{V}$ comprises the domains of all $f \in F$. As a matter of convenience, the Cartesian products of the domains of the variables in F_U, F_Z, F_Y are designated with $\mathcal{U}, \mathcal{U}^T, \mathcal{Y}$, and $\mathcal{Z}$. E.g., $\mathcal{Y} := Y_{f_1} \times Y_{f_2} \times \ldots \times Y_{f_{|F_Y|}}$, $f_i \in F_Y$.

- *Δ is a function, it is called the global state prescription function. Δ declares a set of state variables, $F_X \subseteq F_Z$, and a state space, $\mathcal{X}$, which is the projection of $\mathcal{Z}$ with respect to F_X. Given a state vector $\mathbf{x} \in \mathcal{X}$, a vector of input functions $\mathbf{u}(t) \in \mathcal{U}^T$, and some point in time $t \in T$, Δ determines a constraint vector $\mathbf{z} \in \mathcal{Z}$ including a new state, say, $\Delta : \mathcal{X} \times \mathcal{U}^T \times T \to \mathcal{Z}$.*

- *Λ is a function, it is called the output function. The output function might be a function of constraint variables and input or only a function of constraint variables. Given a constraint vector $\mathbf{z} \in \mathcal{Z}$ and an input vector $\mathbf{u} \in \mathcal{U}$, Λ determines an output vector $\mathbf{y} \in \mathcal{Y}$, say, $\Lambda : \mathcal{Z} \times \mathcal{U} \to \mathcal{Y}$ or $\Lambda : \mathcal{Z} \to \mathcal{Y}$.*

[2] The behavior model definition developed in this place points at a similar direction as the work of Zeigler et al. in (310). Distinctions to Zeigler et al.'s approach include among others: State variables and constraint variables are clearly distinguished here, model behavior can be mapped intuitively onto a model's structure, and, the long winded dichotomy between system structure specification and system specification has been abandoned. Note that in this place, as well as in connection with the construction of models, the work of Wymore should be mentioned, whose clearly formulated ideas form the base for many successors (305).

Remarks. (1) A behavior model over $\langle F, \mathcal{M} \rangle$ is designated with $B\langle F, \mathcal{M} \rangle$; likewise, the set of all behavior models over $\langle F, \mathcal{M} \rangle$ is designated with $\mathcal{B}\langle F, \mathcal{M} \rangle$. (2) Δ is called state prescription function (in place of state *transition* function) since the state space $\mathcal{X}$ may be singleton or empty. (3) For an initial state, $\mathbf{x}^0 = (x_1, \ldots, x_{|F_X|})$, $\mathbf{x}^0 \in \mathcal{X}$, the set of all states that actually can be adopted by the model is called the model's admissible state space, $\mathcal{X}_{\Delta(\mathbf{x}^0,*)}$. The admissible state space restricts the domain and range of Δ related to an initial state. (4) The elements in $\mathcal{U}^T$ are called input trajectories or input signals. (5) The restriction of Δ with respect to some initial state $\mathbf{x}^0 \in \mathcal{X}$ and some input function $\mathbf{u}(t) \in \mathcal{U}^T$, say, $\Delta(\mathbf{x}^0, \mathbf{u}(t), t)$, defines a unique state trajectory along the time base T.

Example. We draw upon the electrical circuit from Figure 2.3, but introduce functionalities for the components' voltage drops. Input functionality is e, the voltage at the source; output functionalities are the voltage drops at the resistance and the capacitance, v_R, v_C.

- *Functionalities.*
  $F = \{R, C, e, i_1, i_2, i_3, \phi_1, \phi_2, \phi_3, v_R, v_C, \dot{v}_C\}$.

- *Submodels.*
  $\mathcal{M} = \{M_1, M_2, M_3\}$, where $M_1 = \{R, i_1, i_2, \phi_1, \phi_2, v_R\}$,
  $M_2 = \{C, i_2, i_3, \phi_2, \phi_3, v_C, \dot{v}_C\}$, and $M_3 = \{e, i_1, i_3, \phi_1, \phi_3\}$.

- *Functionality Roles.*
  $F_U = \{e\}$, $F_Z = \{i_1, i_2, i_3, \phi_1, \phi_2, \phi_3, v_C, \dot{v}_C\}$, $F_Y = \{v_R, v_C\}$, $F_P = \{R, C\}$.

- *Functionality Domains.*
  $U_e = \mathbf{R}$, U_e^T is the set of continuous functions, $T = \mathbf{R}^+$,
  $Z_f = \mathbf{R}$ with $f \in \{i_1, i_2, i_3, v_C, \dot{v}_C\}$, $Z_f = \mathbf{R}^+$ with $f \in \{\phi_1, \phi_2, \phi_3\}$,
  $Y_{v_R} = \mathbf{R}$, $Y_{v_C} = Z_{v_C}$, and $P_f = \mathbf{R}^+$ with $f \in \{R, C\}$.

- *State Prescription Function.*
  $F_X = \{v_C\} \subseteq F_Z$, $X_{v_C} = Z_{v_C}$, $\Delta : X_{v_C} \times U_e^T \times T \to \mathcal{Z}$ (defined implicitly),

$$\Delta = \begin{cases} \dot{v}_C &= C^{-1} \cdot i_2 \\ v_C &= \phi_2 - \phi_3 \\ R \cdot i_1 &= \phi_1 - \phi_2 \\ e(t) &= \phi_1 - \phi_3 \\ i_1 &= i_2 \\ i_2 &= i_3 \\ i_3 &= i_1 \\ \phi_3 &= 0 \end{cases}$$

- *Output Function.*
  $\Lambda : Z_{\phi_1} \times Z_{\phi_2} \times Z_{\phi_3} \to Y_{v_R} \times Y_{v_C}$,

$$\Lambda(\phi_1, \phi_2, \phi_3) : \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} = \begin{pmatrix} v_R \\ v_C \end{pmatrix}$$

Remarks. Of course the sets F and $\mathcal{M}$ and the state prescription function Δ could have been formulated differently, e. g., by defining the continuity conditions in a different way. However, modern modeling/simulation environments such as MODELICA or FLUIDSIM provide powerful means to derive continuity, compatibility, or connection constraints automatically.

The effort that is necessary to process a behavior model $B\langle F, \mathcal{M}\rangle$ is called *constraint complexity*. Beside the size of Δ in the number of variables and equations, the constraint complexity depends on several factors, numerical properties for instance: Numerical properties include the desired accuracy, the convergence behavior of Δ, and stability conditions during simulation.

An important postulation when working with behavior models is the locality principle. It claims that each submodel M has a private computation rule, Δ_M, that is complete and exclusive. Completeness means that all constraint variables of the submodel can be determined by Δ_M; exclusivity means that Δ_M affects solely the functionalities of the submodel. Exclusivity implies that an exchange of information between the submodels must happen via the explicit interaction paths. The locality principle hence imposes certain conditions on the global state prescription function Δ, which are captured by the subsequent definition.

If Δ is represented by a *set* of relations, Δ can be partitioned with respect to $\langle F, \mathcal{M}\rangle$.[3] Note, however, that a subset of these relations, $\Delta_M \subset \Delta$, forms a proper state prescription function only if the domain variables in Δ_M are elements from F_U and F_X, referred to as F_{U_M} and $F_{X_M} \neq \emptyset$, and if the range variables of Δ_M, F_{Z_M}, form a superset of F_{X_M}. I. e., $\Delta_M : \mathcal{X}_M \times \mathcal{U}_M^T \times T \rightarrow \mathcal{Z}_M$, where $\mathcal{X}_M$ and $\mathcal{Z}_M$ define the related state space and constraint domain; $\mathcal{U}_M^T$ is a projection of $\mathcal{U}^T$ respecting the input variables F_{U_M} of Δ_M. Typically, Δ_M does not establish a state prescription function but some other relation on $\mathcal{U}_M^T \times \mathcal{Z}_M$, where both $\mathcal{X}_M \subseteq \mathcal{Z}_M$ and $\mathcal{U}_M^T$ may be empty.

Definition 2.4 (Local Behavior Model) *Let $\langle F, \mathcal{M}\rangle$ be a model, and let $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ be a behavior model over $\langle F, \mathcal{M}\rangle$. $B\langle F, \mathcal{M}\rangle$ is called a local behavior model with respect to $\langle F, \mathcal{M}\rangle$, if for each submodel $M \in \mathcal{M}$ a local behavior relation $\Delta_M \subseteq \Delta$ can be specified such that the following conditions hold.*

(1) $\forall \delta \in \Delta \;\; \forall M \in \mathcal{M} : \; F_\delta \subseteq M \Leftrightarrow \delta \in \Delta_M$,
where F_δ comprises the variables that occur in δ.

(2) $\bigcup_{M \in \mathcal{M}} \Delta_M = \Delta$

Remarks. (1) A local behavior model is sometimes called a component model (137, 205). (2) There exists at most one decomposition of Δ into local behavior relations Δ_M, $M \in \mathcal{M}$, that fulfills the conditions of a local behavior model with respect to $B\langle F, \mathcal{M}\rangle$.

[3] Δ can be represented in a monolythic manner, e. g. in the form of a single look-up table.

Example. Again we draw upon the electrical circuit from Figure 2.3 and investigate the previously defined state prescription function, Δ, with respect to the partitioning conditions of Definition 2.4. Altogether, three sets of relations $\Delta_{M_v, v\in\{1,2,3\}}, \Delta_{M_v} \subseteq \Delta$ must be formed, which are shown in Table 2.1. Obviously, the state prescription function Δ fulfills the locality conditions.

v	$\Delta_{M_v} \subseteq \Delta$	Domain and variables of Δ_{M_v}
1	$\{R \cdot i_1 - \phi_1 + \phi_2 = 0,\ i_1 - i_2 = 0\}$	$P_R \times Z_{i_1} \times Z_{i_2} \times Z_{\phi_1} \times Z_{\phi_2}$, with $\{R, i_1, i_2, \phi_1, \phi_2\} \subseteq M_1$
2	$\{\dot{v}_C - C^{-1} \cdot i_2 = 0,\ v_C - \phi_2 + \phi_3 = 0,$ $\phi_3 = 0, i_2 - i_3 = 0\}$	$P_C \times X_{v_C} \times Z_{\dot{v}_C} \times Z_{i_2} \times Z_{i_3} \times Z_{\phi_2} \times Z_{\phi_3}$, with $\{C, v_C, \dot{v}_C, i_2, i_3, \phi_2, \phi_3\} \subseteq M_2$
3	$\{e - \phi_1 + \phi_3 = 0,\ \phi_3 = 0,\ i_1 - i_3 = 0\}$	$U_e \times Z_{i_1} \times Z_{i_3} \times Z_{\phi_1} \times Z_{\phi_3}$, with $\{e, i_1, i_3, \phi_1, \phi_3\} \subseteq M_3$

Table 2.1. Covering the global state prescription function, Δ, by three sets, $\Delta_{M_1}, \Delta_{M_2}, \Delta_{M_3}$, of submodel relations.

Another interesting property of behavior models is bound up with the concept of causality. By introducing a cause-effect direction for the local behavior relations, causality imposes additional restrictions on the locality property. In this connection we agree on the following notions: For a directed graph $G = \langle V, E\rangle$ of a model $\langle F, \mathcal{M}\rangle$ and an edge $(v, w) \in E$, a functionality $f \in M_v \cap M_w$ is called supplied functionality respecting v and demanded functionality respecting w. Based on the definitions already introduced, causality can be defined in an elegant manner.

Definition 2.5 (Causal Behavior Model) *Let $\langle F, \mathcal{M}\rangle$ be a model, and let $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ be a local behavior model over $\langle F, \mathcal{M}\rangle$ with the local behavior relations Δ_{M_v}, $v = 1, \ldots, |\mathcal{M}|$. A behavior model $B\langle F, \mathcal{M}\rangle$ is called causal if the following conditions hold.*

(1) Input or output functionalities are not supplied and demanded respectively: $\forall\, v, w \in \{1, \ldots, |\mathcal{M}|\}, v \neq w :\ (M_v \cap M_w) \cap (F_U \cup F_Y) = \emptyset$

The model $\langle F, \mathcal{M}\rangle$ has a directed, complete graph[4] *G such that*

(2) each local behavior relation Δ_{M_v} defines a function that has no supplied functionality amongst its domain variables and no demanded functionality amongst its range variables,

(3) G contains no cycles.

Remarks. (1) Within a causal behavior model no submodel depends on itself in a feedback manner. (2) Each causal behavior model defines a feedback-free signal flow

[4] See Definition 2.1.

graph. (3) Causality entails locality, i. e., locality establishes a necessary condition for causality.

Example. We analyze whether the local behavior model of our example, $B\langle F, \mathcal{M}\rangle$, is causal. Assertion: $B\langle F, \mathcal{M}\rangle$ does not establish a causal behavior model because condition (3) of Definition 2.5 can never be fulfilled. Idea: An acyclic, directed graph has at least one node v whose indegree is zero; such a node v cannot be found in any directed, complete graph[4] of $\langle F, \mathcal{M}\rangle$ in the example. Proof (indirect): If the indegree of a node v is zero, the relations in Δ_{M_v} contain no demanded functionalities. This implies that all constraint functionalities of M_v are determined by Δ_{M_v}, which in turn implies that the number of equations in Δ_{M_v} either equals or exceeds the number of constraint functionalitities. Since $M_v \cap F_Z$ designates the constraint functionalitities in M_v, the inequation $|\Delta_{M_v}| \geq |M_v \cap F_Z|$ must hold at least for one node v. As can be seen in Table 2.2, this is not the case. ◇

v	$\Delta_{M_v} \in \Delta$	$M_v \cap F_Z$
1	$\{R \cdot i_1 - \phi_1 + \phi_2 = 0,\ i_1 - i_2 = 0\}$	$\{i_1, i_2, \phi_1, \phi_2\}$
2	$\{\dot{v}_C - C^{-1} \cdot i_2 = 0,\ v_C - \phi_2 + \phi_3 = 0,\ \phi_3 = 0,\ i_2 - i_3 = 0\}$	$\{v_C, \dot{v}_C, i_2, i_3, \phi_2, \phi_3\}$
3	$\{e - \phi_1 + \phi_3 = 0,\ \phi_3 = 0,\ i_1 - i_3 = 0\}$	$\{i_1, i_3, \phi_1, \phi_3\}$

Table 2.2. Contrasting local behavior relations and constraint variables for the submodels of the example. In a steady-state analysis the equation $\dot{v}_C = 0$ is added to Δ_{M_2} by the constraint processing method, in a dynamic analysis the equation $v_C(0) = v_0, v_0 \in \mathbf{R}$.

Observe that the equations in Δ can be sorted in such a way, that all unknown variables can be computed by local value propagation (see Figure 2.5). Within this computation at first values for the potentials, ϕ_v, are determined and afterwards values for the currents, i_v. In this sense Δ could be called causal at the level of constraint variables—however, it is not causal at the level of the submodels M_1, M_2, and M_3.

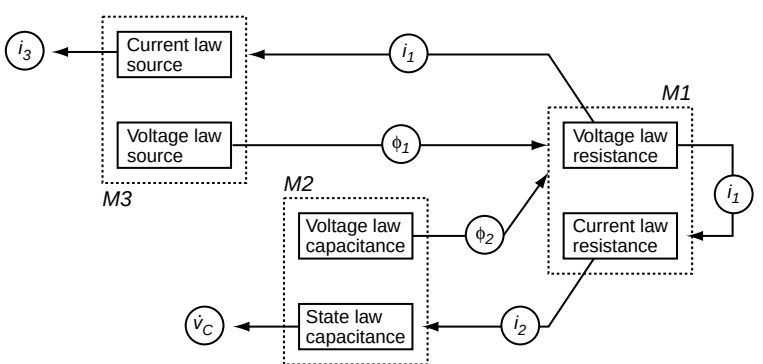


Figure 2.5. Computing sequence for the example circuit: The graph of the behavior laws and constraint variables forms a DAG; at the submodel level the graph contains cycles.

The modeling of complex systems often happens in a bottom-up manner: Small building blocks, the submodels or components, are synthesized towards a single global model. Obligatory for this purpose is a *context-free* description of the submodels—a requirement that is also referred to as *no-function-in-structure* principle. The principle claims that the behavior of a synthesized model emerges in a consistent manner from the behavior of its submodels, independently of the global model's internal structure and the number and type of the submodels used.[5]

Prerequisite for the no-function-in-structure principle is locality, which guarantees that the state prescription function Δ can be divided according to the submodels' functionalities. However, no-function-in-structure tightens the locality restriction: The behavior relation of a submodel M, Δ_M, must not depend on any other submodel, say, no state or parameter within Δ_M contains assumptions or is affected by a state or parameter of another submodel. In this sense, no-function-in-structure restricts the definition of Δ_M, while locality restricts the effects of Δ_M during simulation.

The no-function-in-structure principle is infringed in some examples of this work, and, possibly surprising, this violation forms the key for a powerful model simplification. In this place it is specified what the no-function-in-structure principle means in connection with a behavior model $B\langle F, \mathcal{M}\rangle$ that is defined over a model $\langle F, \mathcal{M}\rangle$. Note that a quantifying description of the no-function-in-structure principle does not exist in the literature on the subject.

Definition 2.6 (No-Function-in-Structure Principle) *Let $\langle F, \mathcal{M}\rangle$ be a model, and let $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ be a behavior model over $\langle F, \mathcal{M}\rangle$. Then $B\langle F, \mathcal{M}\rangle$ complies with the no-function-in-structure principle if the following holds.*

(1) $B\langle F, \mathcal{M}\rangle$ is a local behavior model.

(2) No state or parameter functionality is shared amongst two submodels:
$\forall\, v, w \in \{1, \ldots, |\mathcal{M}|\}, v \neq w : \; (M_v \cap M_w) \cap (F_X \cup F_P) = \emptyset$

Example. Again, the electrical circuit example from Figure 2.3 is considered. The capacity submodel, M_2, is extended by the functionalities R and e, and the local behavior relation, Δ_{M_2}, is reformulated respecting R and e. Note that the resulting new behavior model and the original behavior model from Page 30 specify the same behavior.

- *Modified Submodels.*
  $M_2 = \{R, C, e, i_2, i_3, \phi_2, \phi_3, v_C, \dot{v}_C\}$

[5] *"The laws of the parts of the device may not presume the functioning of the whole."* (139, pg. 16). And, as Kuipers points out in this connection: *"The behavioral repertoire of a type of component must be specified completely, and independently of the contexts in which instances of that component might appear."* (153, pg. 6).

- *Modified Behavior Relations.*
  $\Delta_{M_2} : \dot{v}_C + \frac{1}{RC} \cdot v_C - \frac{e}{Rc} = 0, \quad X_{v_C} \rightarrow X_{v_C}$ (defined implicitly)
- The other elements of $B\langle F, \mathcal{M} \rangle$ remain unchanged.

In the modified behavior model the local behavior relations Δ_{M_1} and Δ_{M_2} share the parameter variable R. Hence $B\langle F, \mathcal{M} \rangle$ does not fulfill the no-function-in-structure principle; nevertheless, $B\langle F, \mathcal{M} \rangle$ establishes a local behavior model.

2.2 Behavior Model Types

This section defines several specializations of the generic behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$. These specialization affect the global state prescription function, the underlying time base, and the domains of the variables in F. All specialization mentioned here play a role in one or more of the projects described in Part II of this thesis; anyway, the listing is not complete in every respect.

Note that the presented behavior models are time-invariant. A behavior model is called time-invariant, if its state prescription function Δ responds in an identical way when applied to the same combination of a state vector $\mathbf{x}$ and an input vector $\mathbf{u}(t)$.

The following three subsections organize the properties of the specialized behavior models within three orthogonal classes. For instance, whether a model is input-free does neither depend on its underlying time base nor on the ranges of the involved variables. Figure 2.6 shows these classes in the form of three separate trees.

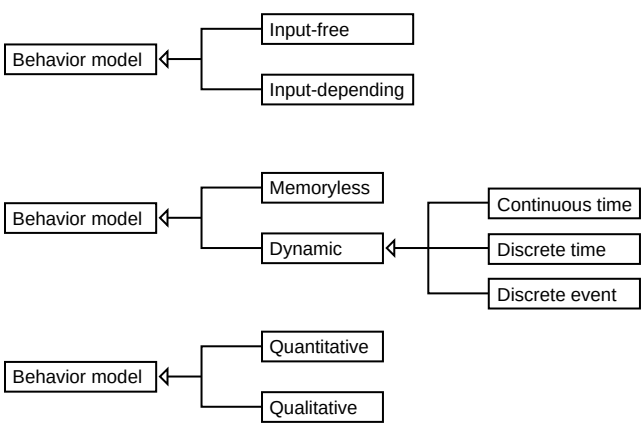


Figure 2.6. The properties of the specialized behavior models can be represented within three orthogonal classes.

Special State Prescription Forms

Definition 2.7 (Input-Free Behavior Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. An input-free behavior model over $\langle F, \mathcal{M} \rangle$ is a tuple $\langle F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ whose elements, F_Z, F_Y, and $\mathcal{V}$ are defined just as for the behavior model in Definition 2.3, Page 29.*

- *Δ is the global state prescription function with time base T. As before, Δ declares a set of state variables, $F_X \subseteq F_Z$, and a state space, $\mathcal{X}$, which is the projection of $\mathcal{Z}$ with respect to F_X. Given a state vector $\mathbf{x} \in \mathcal{X}$ and some point in time $t \in T$, Δ determines a constraint vector $\mathbf{z} \in \mathcal{Z}$ including a state, say, $\Delta : \mathcal{X} \times T \to \mathcal{Z}$.*

- *Λ is the output function. Given a constraint vector $\mathbf{z} \in \mathcal{Z}$, Λ determines an output vector $\mathbf{y} \in \mathcal{Y}$, say, $\Lambda : \mathcal{Z} \to \mathcal{Y}$.*

Remarks. Input-free behavior models cannot respond to inputs. Systems that are represented by input-free behavior models are also called "autonomous systems" (310).

Definition 2.8 (Memoryless Behavior Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. A memoryless behavior model over $\langle F, \mathcal{M} \rangle$ is a tuple $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ whose elements, F_U, F_Z, F_Y, $\mathcal{V}$, and Λ are defined just as for the behavior model in Definition 2.3, Page 29.*

- *Δ is the global state prescription function. Δ declares a set of state variables, $F_X \subseteq F_Z$. If F_X is nonempty, the state space, $\mathcal{X}$, contains a single element only. Given a vector of functions $\mathbf{u}(t) \in \mathcal{U}^T$ and some point in time $t \in T$, Δ determines a constraint vector $\mathbf{z} \in \mathcal{Z}$ say, $\Delta : \mathcal{U}^T \times T \to \mathcal{Z}$.*

Remarks. (1) Models with a singleton or an empty state set are called stationary, as opposed to dynamic models, which can undergo a state change. The output of stationary models depends in a definite way from its input. Zeigler et al. denotes systems that can be represented by memoryless behavior models as "function specified systems" (310). (2) An input-free behavior model with a singleton or an empty state set is called *constant*.

Constant behavior models are used in Section B.1 in the form of resource-based configuration descriptions, and within all model envisioning applications of Chapter D.

There is the important case of stationary models whose state space is known to be a singleton, $\mathbf{x}$, but the vector $\mathbf{x}$ is unknown a-priori. Such cases may arise when simulating a stationary model that has been derived from a dynamic model: It is known that the model has only a single state—the stationary state—but it is unknown which. A stationary model whose state is unknown is called a *selective-state* model here.

Different Time Bases

Depending on the underlying time base a generic dynamic model can assume different shapes, which are specified in the following. Note that this specialization does not apply to memoryless behavior models.

Definition 2.9 (Continuous Time Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. A behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ over $\langle F, \mathcal{M} \rangle$ establishes a continuous time model, if its global state prescription function, Δ, is a continuous function in the parameter time.*

Remarks. Typically, the state prescription function of continuous time models does not provide an explicit mapping onto the state space, $\mathcal{X}$, but defines the state vector's rate of change, $\mathbf{x}'$. A general form for such a description is a differential-algebraic system (DAE):

$$\delta(\mathbf{x}(t), \mathbf{x}'(t), \mathbf{u}(t)) = 0, \text{ where } \mathbf{x}'(t) = \frac{d\mathbf{x}}{dt}$$

Of course the representation as a differential-algebraic system includes other forms, such as the explicit state space form:

$$\mathbf{x}'(t) = \delta(\mathbf{x}(t), \mathbf{u}(t))$$

Definition 2.10 (Discrete Time Model) *Let $\langle F, \mathcal{M} \rangle$ be a model, and let $B\langle F, \mathcal{M} \rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a behavior model over $\langle F, \mathcal{M} \rangle$ with time base T. $B\langle F, \mathcal{M} \rangle$ establishes a discrete time model if its global state prescription function, Δ, is defined only for a finite number of elements in each finite subset of T.*

Remarks. Typically, discrete time models result from a discretization of continuous time models.

Definition 2.11 (Discrete Event Model) *A behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ over $\langle F, \mathcal{M} \rangle$ establishes a discrete event model, if the state space, $\mathcal{X}$, defined by the global state prescription function, Δ, contains only a fixed number of elements.*

Remarks. Note that for a finite period the state space of a discrete time model also contains a finite number of elements only. However, this number depends on the time unit chosen, a fact which distinguishes discrete time models from discrete event models. Fishwick designates discrete event models as declarative simulation models (79, pg. 19).

Models of real systems must not be purely continuous or discrete but can comprise properties from each of the mentioned time bases. For instance, models of fluidic systems usually combine continuous and event-based state prescriptions.

Different Domains and Ranges

It is difficult to draw a line between quantitative models and qualitative models. Is a simplistic model that is based on numerical equations a quantitative or a qualitative model? Or, if a model M' represents an abstraction of some other model M, should M' be called a qualitative model?

In this place we will not engage into these and related questions. The definitions presented below are oriented at the domain sets of a model's variables. This makes sense within two respects. Firstly, a restriction of a variable's value set to a finite number of elements clearly indicates a qualitative character of the associated functionality. Secondly, a state prescription function that defines constraints on qualitative variables inevitably encodes qualitative laws of behavior.

Definition 2.12 (Quantitative Behavior Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. A behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ over $\langle F, \mathcal{M} \rangle$ is called quantitative behavior model, if the domains of the functionalities in F are number fields.*

Definition 2.13 (Qualitative Behavior Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. A behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ over $\langle F, \mathcal{M} \rangle$ is called qualitative behavior model, if the domains of the functionalities in F are finite symbol sets.*

Border cases that cannot be definitely classified may be called semi-quantitative models. Examples: (1) A model that contains both quantitative and qualitative variables. (2) A model whose variables are defined over a number field, but whose state prescription, however, does not obey to physical laws. The resource-based model, which plays a role within design problems, could be called semi-quantitative (see Section B.1).

Table 2.3 lists the projects presented in this thesis and characterizes the underlying behavior models. Within some projects two models are involved.

	Task	Locality	States	Time base	Variable domain
A.1	Case-based reasoning in fluidic design	global	dynamic	continuous	quantitative
A.2	Conceptual design in chemical engineering	global	memoryless	–	semi-quantitative
B.1	Generate heuristics for configuration	local	constant	–	semi-quantitative
B.2	Flatten deep models for diagnosis	local, causal	dynamic	continuous	quantitative
C.1	Generate a WDS from a reference circuit	local, causal	dynamic	continuous, discrete-time	quantitative
C.2	Learn similarity measure from classes	global	dynamic	discrete-event	quantitative, qualitative
D.1	Model formulation for local area networks	local	memoryless	discrete-event	qualitative
D.2	Maintaining knowledge bases	structure model			
D.3	Analyzing fluidic system structures	structure model			

Table 2.3. Tasks presented in this thesis and their underlying behavior models.

2.3 Behavior Model Processing

Each behavior model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ along with an input $\mathbf{u}$ or $\mathbf{u}(t)$ is a set of constraints that prescribe particular values for the variables in F_Z. Processing a behavior model means constraint processing in the broader sense.

Given a memoryless behavior model, an instantiation of the variables in F_Z that fulfills all constraints is called consistent behavior model instance, or simply: behavior. Given a dynamic behavior model, a *sequence* of consistent instantiations of the variables in F_Z is called behavior. This sequence must comply with the underlying time base, which implies among others that the instantiations are linearly ordered according to the parameter "time".

When working with behavior models, frequently occurring questions are concerned with the *admissibility* of variable assignments: Can a partial variable assignment be completed towards a behavior? By which values can a partial variable assignment be completed to become a behavior? The next definition introduces necessary concepts to reason about such questions.

Definition 2.14 (Behavior Model Instance) *Let $\langle F, \mathcal{M}\rangle$ be a model, let $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ be a behavior model over $\langle F, \mathcal{M}\rangle$, and let F_β be a subset of F_Z. A behavior model instance with respect to $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ and an input vector $\mathbf{u}$ is a mapping β, $\beta = \{(f, v_f) \mid f \in F_\beta, v_f \in Z_f\}$. It may fulfill one or more of the following properties.*

- *A behavior model instance β is called partial, if F_β is a proper subset of F_Z.*
- *A behavior model instance β is called consistent, if a mapping $\bar{\beta} = \{(f, v_f) \mid f \in F_Z \setminus F_\beta, v_f \in Z_f\}$ exists such that all constraints defined by Δ are fulfilled. $\bar{\beta}$ is called a completion of β with respect to $\mathbf{u}$.*
- *A consistent behavior model instance β is called determined, if for any two completions $\bar{\beta}_1$ and $\bar{\beta}_2$ applies that $\bar{\beta}_1 = \bar{\beta}_2$. Otherwise β is called under-determined.*
- *A determined behavior model instance β is called over-determined, if a subset of β is also determined. Otherwise β is called definite.*
- *A behavior model instance β is called contradictory if no completion $\bar{\beta}$ exists such that all constraints defined by Δ are fulfilled.*

Remarks. If β represents a definite behavior model instance, its associated set of functionalities, F_β, forms a set of state variables.

Example. We pick up the electric circuit example of Figure 2.3, Page 27. For the vector of component properties, (R, C), the values $(2, 20)$ with $[R] = k\Omega$ and $[C] = \mu F$ are put in; the input function is chosen constant, $e(t) = 10$ with $[e] = V$. Based on these values, Table 2.4 gives examples for the different types of behavior model instances; $[i_v] = mA$, $[\phi] = V$, $[v_C] = V$.

Instance type	i_1	i_2	i_3	ϕ_1	ϕ_2	ϕ_3	v_C	$\dot{v}_C$
Partial β (*)	$\perp$	$\perp$	$\perp$	10.0	$\perp$	$\perp$	$\perp$	$\perp$
Determined (definite) β	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0.0	$\perp$
Over-determined β	5.0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0.0	$\perp$
Contradictory β	5.0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1.0	$\perp$
Completions of (*)	5.0	5.0	5.0		0.0	0.0	0.0	250
	4.5	4.5	4.5		1.0	0.0	1.0	225

Table 2.4. Examples of the different behavior model types for the electrical circuit. The $\perp$-sign indicates that β is not defined for the respective functionality.

The computation of a completion $\bar{\beta}$ of some behavior model instance β is called *simulation*. Depending on the constraints defined by Δ, say, the behavior model type, several simulation methods can be distinguished. Given a state vector and some input, simulation encompasses both the computation of the next state for the variables

in F_X, and the computation of the remaining constraint variables in $F_Z \setminus F_X$. For instance, when processing a continuous-time behavior model specified in the form of a differential-algebraic system, these computations are intertwined and are realized by a single method such as DASSL (210). When processing an event-based behavior model, the state computation and the computation of the remaining constraints may happen within two steps.

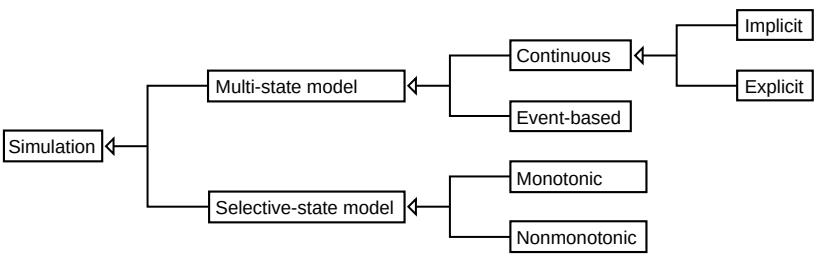


Figure 2.7. Simulation approaches of multi-state models and selective-state models used in this thesis.

The figures organize this view. Figure 2.7 shows different approaches to state computation given a multi-state model and a selective-state model respectively. Figure 2.8 shows various methods for symbolic and numerical constraint processing utilized during the simulation of the different types of behavior models.

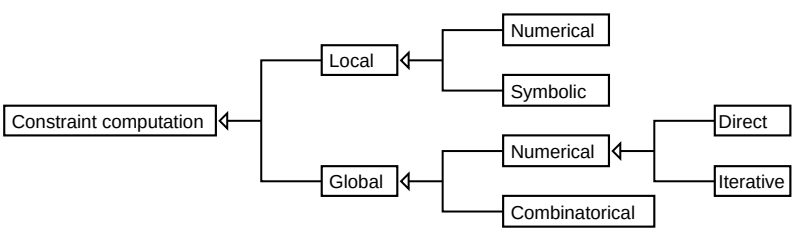


Figure 2.8. Constraint processing methods used in this thesis.

Both the simulation approaches and constraint processing methods can be classified with respect to causality. Altogether three appearances of causality and non-causality can be distinguished:

(1) Continuous multi-state models are processed by numerical integration methods. An integration method is called causal if it employs only values at prior computation time samples; a method is called non-causal if in addition to prior values also values of time samples at and after the present time are employed (310). Causal and non-causal methods are also referred to as explicit and implicit methods respectively.

(2) When processing selective-state models, the unknown single or final state has to be determined (refer back to Page 36). Of course the final state can be computed by starting with the initial state and going through all intermediate states until the final state is reached. We call such a strategy causal or monotonic since it follows the linear order of states. A non-causal or non-monotonic method leaves this linear order and applies some kind of generate-and-test strategy.

(3) The third type of causality corresponds in a one-to-one manner to local and global constraint processing methods. A constraint processing method is called causal or local, if it only considers one constraint at a time to solve a constraint satisfaction problem. Otherwise the constraint processing method is called non-causal or global.

2.4 Model Construction (II)

Main concern of this thesis is the identification and utilization of model construction approaches to solve complex analysis and synthesis problems. Based on the definitions just given this section introduces five generic principles.

Starting point for model construction is always a source model, which is modified in the model construction phase. The idea is that the new model—possibly along with readjusted inference methods for model processing—is better suited to solve the problem at hand, say, to answer the interesting question (see Figure 2.9).

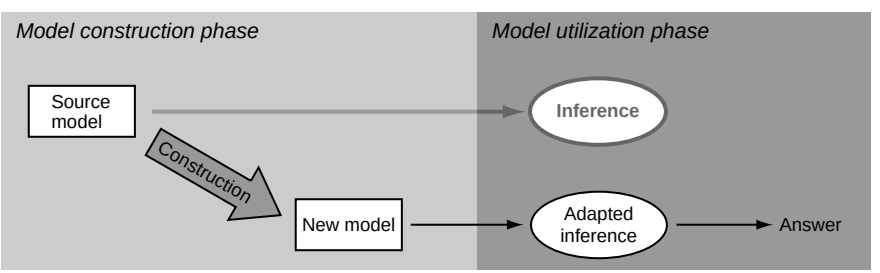


Figure 2.9. Instead of using the source model to answer an interesting question, a new model, which is constructed from the source model, and an adapted inference method are employed.

Let $\langle F, \mathcal{M} \rangle$ be a model, and let $S\langle F, \mathcal{M} \rangle \in \mathcal{S}\langle F, \mathcal{M} \rangle$ and $B\langle F, \mathcal{M} \rangle \in \mathcal{B}\langle F, \mathcal{M} \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$. Together, $S\langle F, \mathcal{M} \rangle$ and $B\langle F, \mathcal{M} \rangle$ form the source model. Model construction can be considered as two mappings, γ_S, γ_B, which map from the source model onto a new model:

$$S\langle F, \mathcal{M} \rangle \xrightarrow{\gamma_S} S'\langle F', \mathcal{M}' \rangle, \quad B\langle F, \mathcal{M} \rangle \xrightarrow{\gamma_B} B'\langle F', \mathcal{M}' \rangle$$

where $S'\langle F', \mathcal{M}'\rangle$ and $B'\langle F', \mathcal{M}'\rangle$ designate a structure model and a behavior model over $\langle F', \mathcal{M}'\rangle$. The effects of a structure model mapping, γ_S, and a behavior model mapping, γ_B, may range from a superficial variation up to a radical reformulation of the source model. At any rate, the application of γ_S and γ_B has teleological character: Both functions address deficits respecting the source model or its utilization—deficits that are quantifiable by means of one or more of the following properties: time, place, handling, maintenance, intricateness, comprehensibility, algorithms, representation.

Here is a key point where this thesis sets in. The functions γ_S and γ_B can be classified—with respect to the mentioned properties and with respect to the problem class for which the new model is constructed. In this connection we introduce the following model construction approaches (principles): model refinement, model simplification, model compilation, and model reformulation. Moreover, the concept of "model envisioning" is introduced; under this term we summarize methods that prepare structure models in a graphical way. In order to make the model construction approaches comparable to each other they are characterized according to the properties specified in Table 2.5.

Property	Semantics
Characteristics	The intended modification of the structure model and the behavior model.
Modeling effects	The effects of a modification from the modeling perspective, say, a user's point of view.
Processing effects	The effects of a modification with respect to model processing. It includes (1) processing efficiency, typically the runtime complexity, (2) processing difficulty, which describes how intricate the employed algorithms are, and (3) model handling, which relates to the maintenance effort of the modified model.
Area of application	The problem classes, where the model construction approach typically plays a role.
Techniques	Techniques, algorithms, and strategies to implement the model construction approach; say, the functions γ_S and γ_B.
Case studies	Problem instances, where the model construction approach has been applied.

Table 2.5. The basic properties that have been used to characterize the model construction approaches.

Note that our list of model construction approaches cannot be complete, and, of course, additional approaches will be developed in the future. However, in this place we do not only aim at an overview of model construction but show also how it is put into practice: The mentioned approaches have been employed successfully within

several projects—a part of which is presented in Part II of this thesis. Especially against the background that recipes for the operationalization of a model construction approach can hardly be stated, the realized implementations can serve as a pool of ideas when tackling new analysis and synthesis tasks.

Remarks. (1) The functions γ_S and γ_B can be compared to the "system morphism" idea of Wymore and Zeigler et al. System (homo-, iso-) morphisms are a concept to transform one system description into another. The main contribution of Wymore and Zeigler et al. is the development of a generic framework for system description, system design, and system analysis. Nevertheless, their work is less concerned with the identification and characterization of special instances of morphisms. In particular they do not investigate different morphisms respecting different problem classes, and they engage only to a small extent in the realization and application of morphisms within analysis and synthesis tasks. (2) The function γ_S operationalizes design knowledge on structure. To provide a means for the specification of this kind of knowledge, Chapter 3 introduces design graph grammars (268, 241), which come along with a clear and widely-accepted semantics.

Model Refinement

Model refinement does not relate to complexity issues. Instead, we comprise methods under this term that are used to round out an incomplete model, to adapt an almost adquate model, or to repair a model that has minor defects. Confer Stein and Curatolo:

> *"By model refinement we designate a completion process to such an extent that the instance of the problem class can be mastered with the refined model."*
>
> Stein and Curatolo, 1996, pg. 58

Formal Description Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E, \sigma \rangle$ and $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$.

(1) *Characteristics.*
The behavior model is adjusted; i. e., the complexity of Δ is not influenced. Component parameters $f \in F_P$ are modified or get assigned a value at all.

The graph of the model, $\langle V, E \rangle$, remains unchanged; possibly, the labeling function σ is redefined (see Figure 2.10).

(2) *Modeling Effects.*
Gaps in the structure model or in the behavior model are filled, thus becoming complete. The behavior model becomes more precise or processible at all.

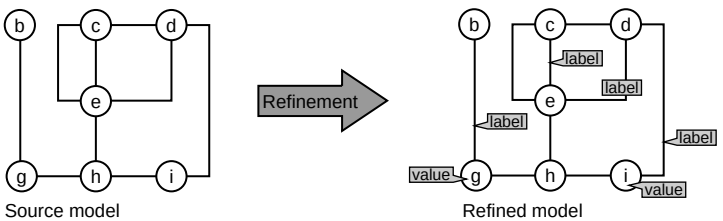


Figure 2.10. The structure model of the refined model is isomorphic to the graph of the source model.

(3) *Processing Efficiency.*
The processing efficiency, the processing difficulty, and the model handling remain unchanged.

(4) *Area of Application.*
There are two typical areas where model refinement is applied: (a) The determination of single of model parameters by means of identification or simulation. (b) The estimation of confidences or importance values within cause-effect behavior models.

(5) *Techniques.*
γ_S: Identification of missing connections in resource-based models (159), (257, pg. 81); path differentiation and path division to introduce additional differentiating rules within a rule base for diagnosis (127, pg. 25); symptom conditionalization and symptom distinction to refine existing diagnostic rules with respect to their confidence factors (127, pg. 26).

γ_B: Determination of unknown values for functionalities in F_P by means of simulation or experimentation; estimation of membership functions by domain experts; quantification of dependencies and adaptation of confidence values with statistical methods, prevalently regression; identification of inconsistencies in weighted rules by a tentative inference (127, pg. 23). Prerequisite for statistical refinement techniques are knowledge sources in the form of example bases or case bases.

(6) *Case Studies.*
Instantiating of heuristic diagnosis rules (107); selection and adaptation of design rules in impeller design (260); evaluating repair rules in hydraulic circuit design (288, pg. 65). The case studies are not included in this thesis.

Model Simplification

Model simplification aims at a reduction of either a model's analytic complexity (cf. Page 26), a model's constraint complexity, or both. A reduction of the *search space complexity* may be an additional consequence but is not top priority. While analytic complexity and constraint complexity play a dominant role in analytical problem solving tasks, the search space complexity is of paramount importance within synthesis tasks: It is a measure for the number of models that have to be synthesized and analyzed in order to solve a synthetical task, say, a configuration or design problem.

The concept of model simplification has been captured by several researches before. The authors quoted below are two representatives; they anticipate quite well the items "characteristics" and "modeling effects" of our formal description. Note, however, that Fishwick uses the term "abstraction" instead of "simplification", and his term "process" corresponds to "behavior model" in our terminology.

> *"Model simplification means to reduce the number of interactions or to substitute simpler interactions for more complex ones between the model constituents."*
>
> Karplus, 1977, pg. 3

> *"Abstraction of a process will inevitably involve a reduction in model components and interactions, along with the reduction in behavioral complexity of the model when simulated."*
>
> Fishwick, 1988, pg. 18

Formal Description Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E, \sigma \rangle$ and $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$.

(1) *Characteristics.*
Simplification of behavior models usually happens within two respects. Firstly, the set of functionalities, F, may be restricted to a subset $F' \subseteq F$, which entails a reduction of Δ's domain and range. Secondly, the complexity of the functions in Δ may be reduced, even up to the total omission of Δ. The former establishes an aggregation of function (84) and is directly connected to structure simplification; the latter falls into the class of behavior aggregation.

If the set of functionalities, F, is restricted to a subset $F' \subseteq F$, edges in $\langle V, E \rangle$ are deleted and elements in $\mathcal{M}$ may become empty sets, resulting in a simpler structure model $\langle V', E', \sigma' \rangle$. The graph $\langle V', E' \rangle$ is a contraction of $\langle V, E \rangle$, as shown in Figure 2.11.

(2) *Modeling Effects.*
The lessened interaction between the submodels in $\mathcal{M}$ results in the neglection

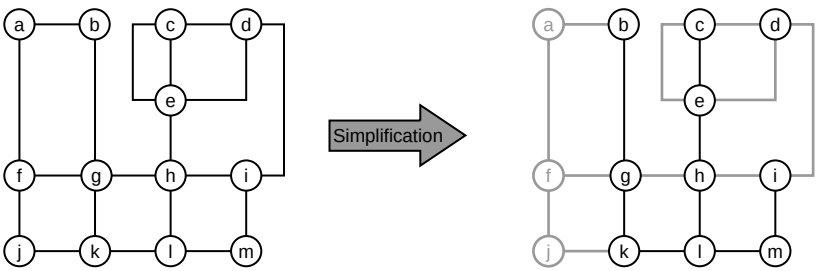


Figure 2.11. The simplified structure model emerges from a contraction of $\langle V, E \rangle$.

of physical effects. The simplification of Δ results in a coarsening of physical phenomena or in physically wrong connections. The behavior is rendered inaccurately up to certain degree.

A reduction of $\mathcal{M}$ and Δ makes the simplified model easier to understand (78).

(3) *Processing Effects.*
Both structure model and behavior model can be processed more efficiently; the processing difficulty is reduced; the model handling is simplified.

(4) *Area of Application.*
Analysis of large or numerically demanding behavior models; synthesis of behavior models without knowledge about the model structure (= design).

(5) *Techniques.*
γ_S: Elimination of feedback loops; equalization of the node degree in $\langle V, E \rangle$; elimination of edges to create a causal ordering, say, a unique computation sequence when determining unknown functionalities (194, 90).

γ_B: Fuzzyfication of equations in Δ; piecewise linear reasoning (232); linearization of higher order polynomials; balancing of dominant terms in complex equations (306); state combination by aggregating similar states; numerical coarsening by means of scaling down numerical precision of the functionalities in F; order of magnitude reasoning (217); reduction of the source model onto a structure model by omitting Δ totally, which is called "representational abstraction" by Fishwick (78).

(6) *Case Studies.*
Case-based design of fluidic circuits (Section A.1); conceptual design in chemical engineering (Section A.2); graph-based simulation of fluidic circuits (not included in this thesis).

A model simplification represents a more or less serious intervention in the physical underpinning of the source model. Hence, model simplification is always bound

up with model evaluation. It has to be ensured that the simplified model is able to answer the interesting question in connection with the intended experiment.

Model Compilation

Model compilation is the anticipation of model processing effort; say, processing effort is shifted from the model utilization phase to the model construction phase. Model compilation is a powerful construction approach to address a model's analytic complexity, its constraint complexity, or the search space complexity. Figuratively speaking, model compilation means to create a compiled model by introducing either (1) computational short cuts within long-winded calculations that are caused by a complex behavior model, or (2) exploratory short cuts within a large search space that results from problem-inherent combinatorics.

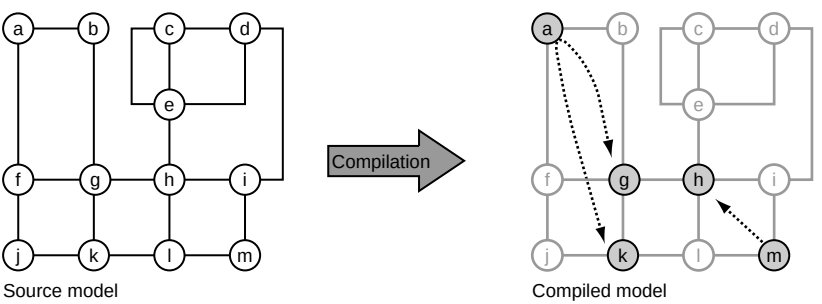


Figure 2.12. Computational short cuts: Hints in the compiled behavior model short-circuit the computation of constraints between functionalities of different submodels. For example, the input set, F_U, could directly be mapped onto the output set, F_Y.

The idea behind several model compilation approaches is to break global connections within the source model down to local connections, which are encoded within the compiled model in the form of special hints. These hints can take one or more of the following three forms.

- Hints that use memorization to short-circuit involved state prescription constraints between functionalities of different submodels (see Figure 2.12). Cause effect chains are shortened, possibly to simple associations. Involved state prescription constraints, e. g. in the form of equation systems, are typical for technical systems whose behavior is described by compatibility constraints and continuity constraints (297).

- Hints that suppose an order within a sequence of tentative search space decisions (see Figure 2.13). These hints are introduced as tags at the respective choice points in the search space and control the search.

- Hints that restrict the search space by introducing additional state prescription constraints (see Figure 2.14).

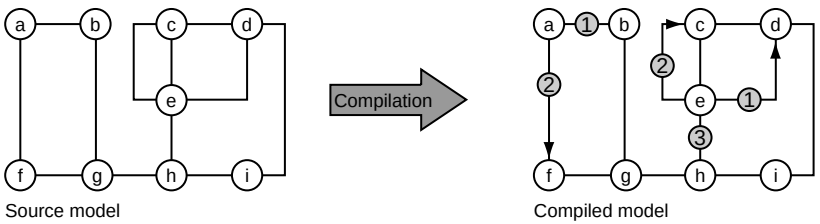


Figure 2.13. Exploratory short cuts: Hints in the compiled structure model define an order on alternative submodels when exploring the search space while solving a synthesis task.

Model compilation methods can also be characterized by their *scalability*. Using a scalable method, there is a trade off between the preprocessing effort and the knowledge gained for the model utilization phase. The scalable character of a compilation method may be bound up with the depth of the analyzed search space, or the precision at which simulations are performed. However, a compilation method that analyzes a model with respect to special structures is normally not scalable. In Chapter B we will introduce representatives for both kinds of methods.

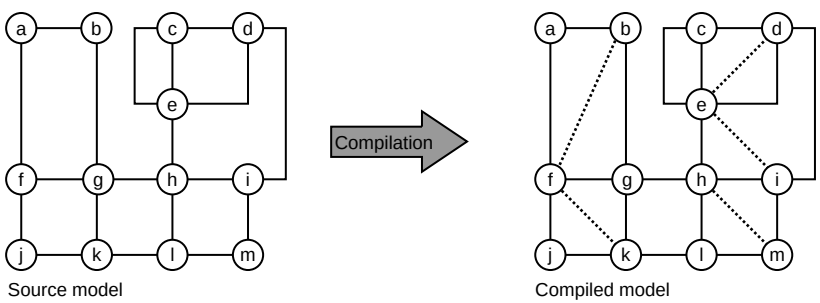


Figure 2.14. Exploratory short cuts: Hints in the form of additional state prescription constraints (dashed lines) restrict the search space.

Formal Description Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E, \sigma \rangle$ and $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$.

(1) *Characteristics.*
The set of functionalities, F, remains unchanged. The behavior model may be equipped with additional constraints that encode numerical or search-specific hints.

The labeling function σ is redefined if a preference order is encoded on the nodes in $\langle V, E \rangle$. New edges are introduced if the state prescription function, Δ, is extended by additional constraints (see Figure 2.14).

(2) *Modeling Effects.*
The modeling accuracy and the level of detail is not reduced, although the ways of computing model behavior may be completely altered.

(3) *Processing Efficiency.*
The model can be processed much more efficiently referring to the interesting problem solving task. However, no respect for the inference intricatenesses is shown; i. e., the implementation of the necessary inference (simulation) algorithms may be more challenging than before compilation. Moreover, model handling gets more complicated since modifications of the model may entail a renewed preprocessing.

(4) *Area of Application.*
Whenever processing efficiency is highest bid and processing hints can be computed at all, model compilation is expedient. Given this situation, following further prerequisites must be fulfilled. (a) Firstly, the problem solving task is apportionable into two phases: A model construction or preprocessing phase, where computing power and/or computing time are given on a large scale, and a model utilization or problem solving phase, where computing resources are short. (b) Secondly, for the portion, p, of problem instances that make a renewal of the compilation process necessary holds $p \ll 1$.

(5) *Techniques.*
γ_S: Topological analyses from which computational constraints or search constraints are derived; determination of candidates for a dependency-directed or knowledge-based backtracking; identification of spheres of influence of both revision and trashing (170, pg. 111); decomposition of equation models into (a) involved subproblems, which must be processed by a global method, and (b) feedback-free subproblems, which can be tackled by a local inference approach (262); model decomposition by causal analysis, which means the application of equation model decomposition onto qualitative simulation (42).

γ_B (demanding computation): Pre-computation of typical simulation situations and encoding of input/output associations in the form of look-up tables; compilation of rules into a rete-network (83); utilization of an assumption-based truth maintenance system (ATMS) in order to organize results of computational expensive inference problems (138); sharing of computation results by identifying and replacing instances of similar submodels, a concept that can be realized algebraically at the level of Δ (262) or at the more abstract level of model fragments; case-based learning of characteristic features to select suited inference methods for the computation of Δ (263).

γ_B (large search space): Behavior aggregation by a precedent detection of repeating cycles (295); coalescing a system by methods from the field of inductive inference (12), which is called "Abstraction by Induction" in (78); extensive or even exhaustive search in order to analyze the search space or to develop a decision strategy at choice points; ordering of value assignments in constraint-satisfaction problems with finite domains (85, 52, 53).

(6) *Case Studies.*
Compilation of dynamic behavior models for diagnoses purposes (Section B.2); generation of search heuristics in configuration problems (Section B.1); identification of nogoods to speed up model fragment synthesis (259) (not included in this thesis).

Remarks. The techniques presented here address the compilation of models of *technical systems*. They take advantage of this fact: Observe, for instance, that within a diagnosis task not at all points of time all model functionalitities must be known, or that behavior and structure are coupled, which submits to infer computational hints for the behavior model from the underlying structure model.

We call a compilation method that is not specially keyed to models of technical systems a *knowledge* compilation method. Knowledge compilation methods presuppose a determined knowledge representation form, but no problem solving task, no domain, and no model. The rete-algorithm mentioned above is such a knowledge compilation method; its prescribed knowledge representation form is the rule form. Another rule-based knowledge compilation method has been developed by Zupan. While the rete-algorithm has been developed for rule languages whose interpretation is defined on the recognice-and-act cycle, Zupan's compilation method exploits the fixed-point convergence of the rule language to be compiled (312).

The following examples give an idea of the spectrum of knowledge forms where compilation methods can be applied. (1) The syntatic analysis and substitution of algebraic terms, which ensures a unique term occurence with respect to a given normal form, establishes knowledge compilation method. (2) A still more basic knowledge compilation method is based on Horn approximations (249); its prescribed knowledge representation form are formulas in propositional form. (3) If graphs are the interesting knowledge form, knowledge inference may bound up with graph matching or subgraph isomorphy. For the latter problem Messmer and Bunke have developed a compilation method with scalable preprocessing effort (181).

Model Reformulation

In a literal sense, every construction of a new model from a source model could be entitled a reformulation. This not the intention here, but the term model reformulation is used as a collective term for model constructions that are indifferent with respect to both modeling effects and processing efficiency.

Model reformulation aims at issues from one or more of the following fields: knowledge representation, knowledge acquisition, model maintenance, available inference methods, user acceptance. After a model reformulation, the resulting model is in a form ready to become used for the problem solving task in question. Model reformulation does not target on complexity issues in first place.

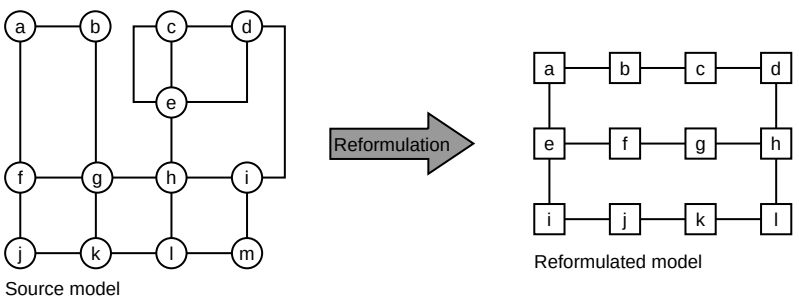


Figure 2.15. Model reformulation is usually bound up with a paradigm shift in model processing, entailing both a new structure model and behavior model.

Formal Description Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E, \sigma \rangle$ and $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$.

(1) *Characteristics.*
The set of functionalities, F, may or may not be altered. Typically, the state prescription function, Δ, is reformulated. Put another way, there is a paradigm shift in model processing.

(2) *Modeling Effects.*
Ideally, there are no effects on the model's accuracy or its level of granularity.

(3) *Processing Effects.*
Ideally, the processing efficiency is not affected. Nothing can be said regarding processing difficulty. The model handling may be simplified.

(4) *Area of Application.*
There is no specific area of application. Model reformulation comes into play if a model that has been developed with respect to a special processing approach shall be transformed for another processing approach.

(5) *Techniques.*
There is no specific reformulation technique.

(6) *Case Studies.*
Constructing wave-digital filters from analogous circuits (Section C.1); learning similarity measures from user specifications (Section C.2); coding qualitative diagnosis models as propositional formulae (not included in this thesis).

As opposed to model refinement, model simplification, or model compilation, there is no pool of techniques by which a model reformulation is to be realized. This is in the nature of things; model reformulation takes a model that has been used successfully with processing paradigm A and tries to transform this model such that it can be used within processing paradigm B. I. e., there is no model-inherent objective; the reformulation constraints are exogenous variables.

At first sight model reformulation appears to be a close relative of model compilation. This, however, is not the case. The maxim of model compilation is processing efficiency, and the problem solving task could be done without a compilation—at a lower processing effiency, of course. We speak about model reformulation, if a model must be transformed at first into another representation in order to be processed within the problem solving task.

Model Envisioning

Model envisioning implies no constructional mission but is a collective term for analysis methods that base on the rendering of structure models. The objectives of model envisioning comprise the provision of insights and the simplification of the access when dealing with complex models. The graphical preparation of a structure model can be used for analysis, modification, maintenance, or acquisition purposes—with respect to both structural and behavior model properties.

In this vein model envisioning defines a new kind of problem solving method that has been developed and applied by Niggemann and Stein within several projects (265). Model envisioning is the attempt to identify and to visualize a model's *natural* structure. It is related to different areas of visualization and makes heavy use of methods from the field of cluster detection and graph drawing (282, 69, 134).

Formal Description Let $\langle F, \mathcal{M} \rangle$ be a model, and let $\langle V, E, \sigma \rangle$ and $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ be a structure model and a behavior model over $\langle F, \mathcal{M} \rangle$.

(1) *Characteristics.*
The graph of the model, $\langle V, E \rangle$, is modified with respect to the envisioning goal. Moreover, the resulting graph is enriched by graphical information, which specify clustering and geometry information that are used for layout purposes.

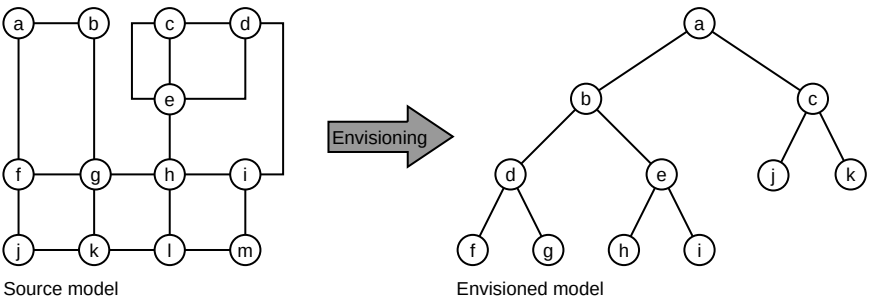


Figure 2.16. Model envisioning deals with the graphical preparation of structure models.

(2) *Modeling Effects.*
Of no account, the model is solely used for envisioning purposes.

(3) *Processing Effects.*
Of no account, the model is solely used for envisioning purposes.

(4) *Area of Application.*
Representation and preparation of complex structure models whereas the following goals can be pursued: (a) Extraction of insights for special analysis tasks (diagnosis) or synthesis tasks (model formation, design), (b) providing a means for understanding and maintaining large models, (c) isolation of particular parts of a model.

(5) *Techniques.*
γ_S: Identification of sub-structures in $\langle V, E \rangle$ by means of graph matching; topological modifications of $\langle V, E \rangle$ with graph grammars; identification of equivalence classes in V by means of clustering methods; layout of $\langle V, E \rangle$ by developing a mapping α that assigns each node in V a point in the Euklidian plane.

A variety of graph drawing approaches exist to define α, which can be divided into two classes: (a) Categorical approaches, which try to arrange a graph according to a particular scheme, paradigm, or philosophy, such as hierarchical leveling, attracting forces, or recurring resemblances (280, 60, 220). (b) Without implying any graph structure, where the layout of a graph is defined by a quality measure, q, that captures a variety of esthetics criteria (49, 18).

(6) *Case Studies.*
Model formulation for LANs (Section D.1); envisioning of large configuration knowledge bases (Section D.2); envisioning of fluidic axes and their couplings (Section D.3).

Discussion

Table 2.6 contrasts the presented model construction approaches. The used symbols are interpreted as follows: ↑↑ (↑) means strong (low) positive impact, while ↓↓ (↓) means strong (low) negative impact, the dash stands for no impact. Note that this table represents the dependencies in an oversimplified way and should only be used as a road map.

Approach	Modeling quality	Processing efficiency	Processing difficulty	Handling difficulty
Refinement	↑	–	–	–
Simplification	↓↓	↑	↓	↓
Compilation	–	↑↑	↑	↑
Reformulation	–	–	↓	↓↓
Envisioning	–	–	↑	↑

Table 2.6. Short characterization of the model construction approaches.

The model construction approaches "simplification", "compilation", and "reformulation" could be called *process-centered*, while "refinement" and "envisioning" should be called *model-centered*. This classification reflects the pursued intentions of the knowledge engineer and is explicated now.

- *Process-Centered Model Construction.* Model simplification as well as model compilation relate to the difference between the provided and the required computing power, when going to solve a problem with the source model. A model is simplified if the interesting problem solving task cannot be handled with this model—a situation that occurs if, for instance, a design task is addressed with a model conceived for an analysis task. In fact, model compilation can provide a way out in such a situation as well. The essentials for a compilation strategy are twofold: The task in question can be tackled with acceptable computing power, and, the employment of the necessary computing power can be alloted a model construction and a model utilization phase.

  Model reformulation relates to the conceptual difference between the source model and the model required for the processing method at hand. Note that, just as in the simplification or the compilation case, a shortcoming of the processing situation is addressed.

- *Model-Centered Model Construction.* By contrast, model refinement and, in some degree, model envisioning address a shortcoming in the model itself. The former approach fills gaps or fixes inaccuracies, while the latter approach modifies a model in order to render structural properties for subsequent knowledge processing tasks.

> Note that a model simplification strategy is also advisable if the simplified model is easier to understand or if its behavior is easier to interpret than were the respective considerations at the source model (78). Given this case, the related simplification establishes a model-centered construction as well.

Model construction approaches must not be applied as a pure concept but may complement themselves when applied multiply. For example model simplification and model compilation go well together for several reasons, as demonstrated within the case study in Section B.2.

Related Work. Literature on model construction concentrates primarily onto model simplification (see 310, Chap. 14), for which different nomenclatures are used. Fishwick (78), for instance, uses the term "model abstraction" and "abstract model" in a similar sense as the term "simplification" is used here. An exception forms his simplification method "abstraction by induction", which should rather be counted as a model compilation method. Frantz comprises under the term "model abstraction" also compilation and refinement operations. Zeigler et al. use the term "approximate morphism".

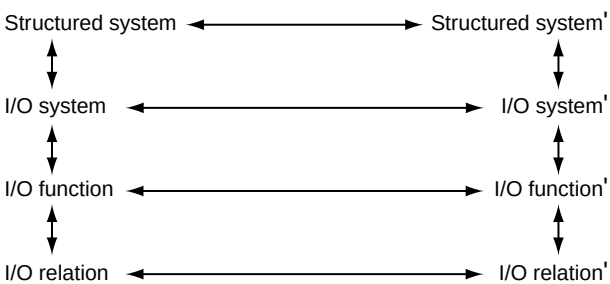


Figure 2.17. Hierarchy of system specifications according to Zeigler et al. (310). The horizontal lines represent mappings for model construction, called morphisms here.

The (deficient) differentiation between rather different model construction approaches is the consequence of a generic principle: Existing classifications of both models and model construction strategies are, in first place, developed from a modeler's point of view and only to a small extent from the standpoint of problem solving tasks.

Figure 2.17 shows the modeler's point of view as it can be found in (310, pg. 296 or 377). The vertical arrows in this diagram connect a behavior model at different levels of explicitness. At the lowermost level, behavior is specified by input/output relations; when going up in the hierarchy, the models get supplemented bit by bit: by a global state prescription function, by initial states, by local behavior relations, and, finally, on the topmost level, by a component influence structure. The horizontal arrows represent mappings between two models; they are called morphisms here and correspond to our construction functions γ_S and γ_B.

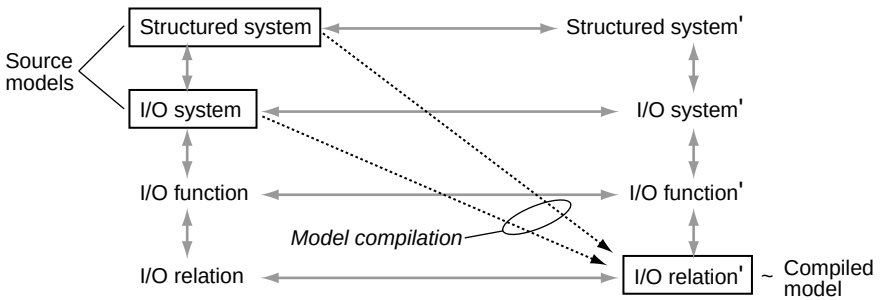


Figure 2.18. Integrating the idea of model compilation into Zeigler et al.'s hierarchy of system specifications: A high-level system description, e. g. an equation model, is broken down to plain I/O relations by means of model compilation.

We can use this diagram to point up the effects of a model compilation that introduces computational short cuts: It is a mapping from a model on the left-hand side onto the I/O-relation model at the lowermost level on the right-hand side (see Figure 2.18).

3

Design Graph Grammars: Specifying Knowledge on Structure

The previous chapter introduced a modeling perspective for technical systems and, based on this perspective, formulated several model construction principles. The following questions may suggest themselves.

- Do universal, i. e., model-independent approaches to the described model construction principles exist?
- Given different instances of the same model construction principle, which aspects do they have in common?
- Can a theory of model construction principles be developed?

Since each domain, each task, and each granularity level has its own, particular modeling constraints, the existence of a generic recipe for the simplification, compilation, or transformation of a model is unlikely. This—expected—answer is also reflected by the spectrum of problem solving methods that has been employed within the projects outlined in Part II of this work.

However, many model modification strategies affect structural properties of a model $\langle F, \mathcal{M} \rangle$. These modifications can be specified in a uniform way since the underlying structure models, $S\langle F, \mathcal{M} \rangle$, are of the same form, namely labeled multigraphs $\langle V, E, \sigma \rangle$. A uniform specification reveals similarities when tackling different model construction tasks and can thus prepare the ground for the development of a model construction theory.

History. The idea to employ graph grammars as a means to describe, or, as the case may be, to operationalize knowledge on the analysis and synthesis of structure models suggests itself. With a close look at expressiveness and applicability in engineering domains we have developed an advancement of classical graph grammar approaches, which we call "design graph grammars" (268, 241). Design graph grammars (DGG) provide a precise semantics for the structural modification a model

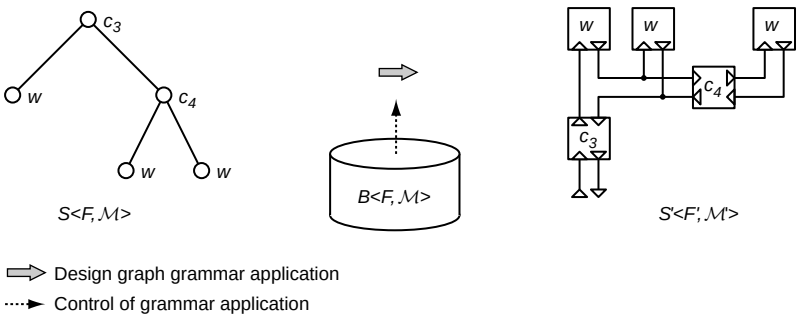


Figure 3.1. Design graph grammars provide a universal means for structure model manipulation. The problem-specific knowledge, which controls the manipulation process, may rely on a behavior model analysis.

undergoes during its simplification or transformation. They are used throughout the projects of Part II as a formal description methodology.

Note that the control, say, the application of a design graph grammar, is grounded on deep domain knowledge and is typically tied to the underlying behavior model $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ (see Figure 3.1). For instance, within an optimization or synthesis task, structure modifications will often realize repair rules, which improve unsatisfying model behavior detected during the analysis of $B\langle F, \mathcal{M}\rangle$ (238, 270). The control knowledge can be acquired from domain experts; it makes up a predominant portion of the solution, and, as said above, it must be developed from scratch as the case arises.

The remainder of the chapter is devoted to design graph grammars and is organized as follows. Section 3.1 is intended as an introduction and motivation around structure model manipulation. Section 3.2 then presents design graph grammars as an appropriate means for this objective. The development and use of design graph grammars is connected to various theoretical issues, which are addressed in the last two sections. Section 3.3 examines the relationship of design graph grammars to classical graph grammars. Section 3.4 presents issues concerning the application of graph grammars in structure analysis and system design.

3.1 Design Tasks and Transformation Rules

The design of a system encompasses a variety of different aspects or tasks—not only the traditional construction process with which it is usually associated. For each of these tasks different operations of varying complexity are required:

- Insertion and deletion of single items in a system,
- change of specific items and connection types,
- manipulation of sets of items, e. g., for repair or optimization purposes.

The operations delineated above can be viewed as transformations on graphs; they are of the form *target* → *replacement*. A precise specification of such graph transformation rules can be given with graph grammars. A central concept in this connection is bound up with the notions of matching and context, which, in turn, ground on the concept of isomorphism (126).

Definition 3.1 (Isomorphism, Isomorphism with Labels) *Let $G = \langle V, E\rangle$ and $H = \langle V_H, E_H\rangle$ be two graphs. An isomorphism is a bijective mapping $\varphi : V \to V_H$ for which holds: $\{a, b\} \in E \Leftrightarrow \{\varphi(a), \varphi(b)\} \in E_H$, for any $a, b \in V$. If such a mapping exists, G and H are called isomorphic.*

G and H are called isomorphic with labels, if G and H are labeled graphs with labeling functions σ and σ_H, and the following additional condition holds: $\sigma(a) = \sigma_H(\varphi(a))$ for each $a \in V_G$, and $\sigma(e) = \sigma_H(\varphi(e))$ for each $e \in E$, where $\varphi(e) = \{\varphi(a), \varphi(b)\}$ if $e = \{a, b\}$.

Figure 3.2 shows an example of isomorphic and non-isomorphic graphs.

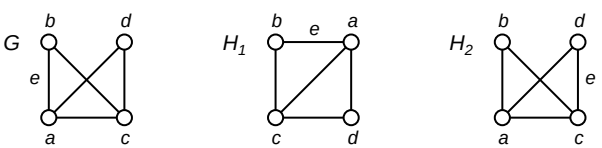


Figure 3.2. A graph G, a graph H_1 that is isomorphic to G, and a graph H_2 that is not isomorphic to G.

Definition 3.2 (Matching, Context) *Given are a labeled graph $G = \langle V, E, \sigma\rangle$ and another labeled graph, C. Each subgraph $\langle V_C, E_C, \sigma_C\rangle$ in G that is isomorphic to C, is called a matching of C in G. If C consists of a single node only, a matching of C in G is called node-based, otherwise it is called graph-based.*

Moreover, let T be a subgraph of C, and let $\langle V_T, E_T, \sigma_T\rangle$ denote a matching of T in G. A matching of C in G can stand in one or both of the following relations to a matching of T in G.

(1) $V_T \subset V_C$. *Then the graph* $\langle V_C, E_C, \sigma_C \rangle$ *is called a context of* T *in* G.

(2) $\langle V_T, E_T, \sigma_T \rangle = \langle V_C, E_C, \sigma_C \rangle$. *Then* T *is called context-free.*

A matching of a graph H *in* G *is denoted by* $\widetilde{H}$. *In general, we will not differentiate between a graph* H *and its isomorphic copy.*

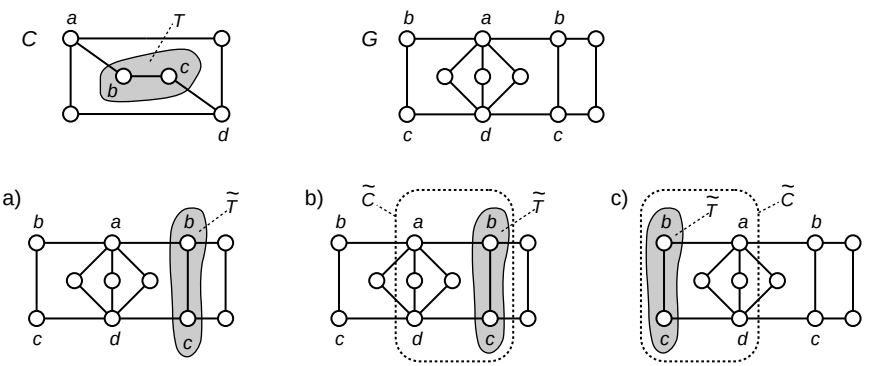


Figure 3.3. Above: A graph C with a subgraph T, and a graph G. Below one can see three matchings of T and C in G: a) matching of T without context; b) matching of T with context C; c) matching of T with context C whereas the matching of T is a strict degree matching.

Remarks. Each matching $\widetilde{H}$ of a graph H within a graph G represents a subgraph of G, which means that every node of $\widetilde{H}$ may be connected to the remainder of G by some edges. This understanding of matching may be sufficient for many purposes, but the domain of technical systems requires more flexibility. Thus, the matching concept is extended to allow a finer control: Let $V^*_H \subset V_H$ be a set of nodes in H. Then $\widetilde{H}$ is a *strict degree matching* of H in G if no node in $V^*_{\widetilde{H}} \subset V_{\widetilde{H}}$ is connected to the remainder of G by some edge. The use of this type of matching will be indicated by specifying a set V^*_H. In practice, the nodes in V^*_H are marked with an asterisk. Figure 3.3 illustrates the notions of matching and context.

Existing graph grammar approaches are powerful, but lack within two respects. Firstly, the notion of context is not used in a clear and consistent manner, which is also observed in Drewes et al. (58, pg. 97). Secondly, graph grammars have rarely been applied to solve synthesis and analysis problems in the area of technical systems. Instead, graph grammar solutions focus on software engineering problems for the most part (62, 63, 64, 132, 133, 149, 162, 222, 228, 243, 246).

The systematics of design graph grammars introduced here addresses these shortcomings. Figure 3.4 relates classical graph grammar terminology to typical design tasks; the remainder of the subsection presents examples for differently powerful rules of type *target → replacement*. A precise analysis of the relationship between classical graph grammar families and design graph grammars can be found in Section 3.3.

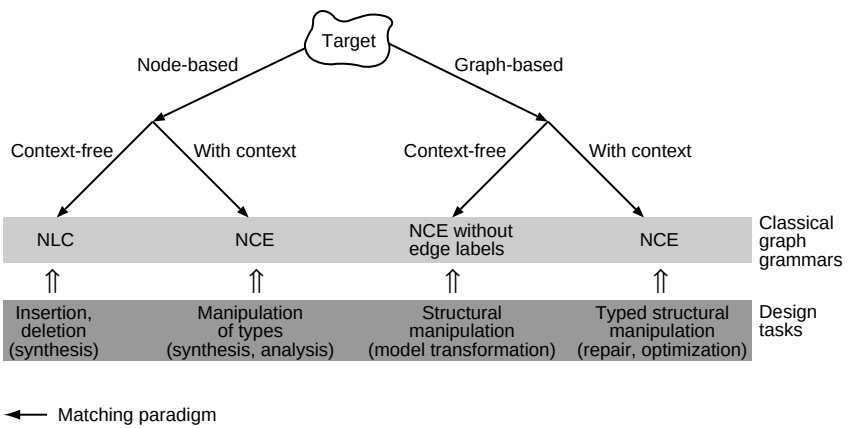
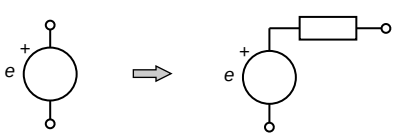


Figure 3.4. A graph grammar hierarchy for the various design tasks. The abbreviations NLC and NCE denote classical graph grammar families.

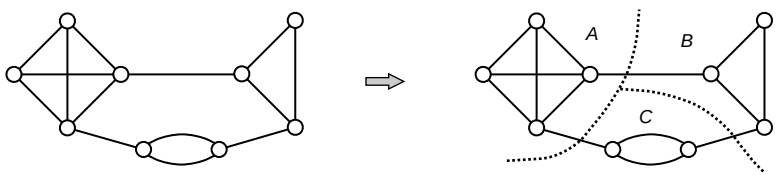
- *Node → Node.* Context-free transformation based on node labels. Graph grammars with rules of this type are called *node label controlled* graph grammars (NLC grammars). The figure shows a type modification of a mixing unit, which can be realized by this class of rules.

- *Node → Graph.* Context-free transformation based on node labels (NLC grammars). The figure shows the replacement of an ideal voltage source with a resistive voltage source (synthesis without context).



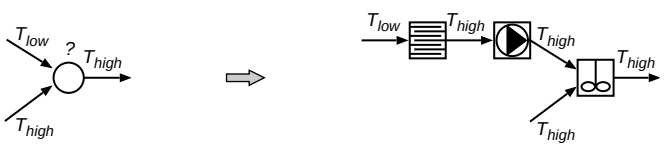
- *Node with Context → Node.* Node-based transformation based on node labels and edge labels. Graph grammars with rules of this type are called *neighborhood*

controlled embedding graph grammars (NCE grammars). The figure shows the clustering of graphs (analysis and synthesis with context), an example for this type of transformation.

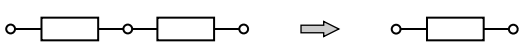


- *Node with Context → Graph.* Node-based transformation based on node labels and edge labels (NCE grammars). The figure shows the replacement of an unknown unit by inserting a heat transfer and a pump unit to fulfill the temperature constraints (synthesis with context).

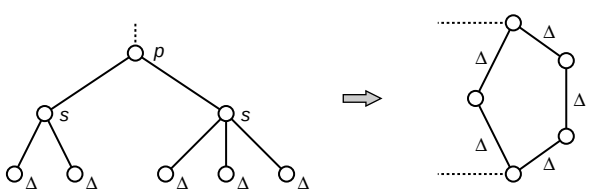
Another example for a transformation of this type is the replacement of an unknown unit by inserting a heat transfer unit, a pump unit, and a mixing unit (synthesis with context).



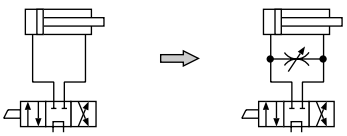
- *Graph → Graph.* Context-free transformation based on graphs without edge labels (NCE grammars without edge labels). The figure shows the replacement of two resistors in series with one resistor.



Another example for a transformation of this type is the conversion of a structure description tree into a parallel-series graph (model transformation).



- *Graph with Context → Graph.* Context-sensitive transformation based on graphs with edge labels (NCE grammars). The figure shows the insertion of a bypass throttle (repair, optimization).



3.2 Design Graph Grammars

What happens during a graph transformation is that a node t or a subgraph T in the original graph G is replaced by a graph R. Put another way, R is embedded into G. In the sequel we will provide a formal basis for the illustrated graph transformations.

Definition 3.3 (Host, Context, Target, Replacement Graph, Cut Node) *Within the graph transformation context, a graph can play one of the following roles.*

- *Host Graph G. A host graph represents the structure on which the graph transformations are to be performed.*
- *Context Graph C. A context graph represents a matching to be found in a host graph G. The graph C is part of the left-hand side of graph transformation rules.*
- *Target Graph T. A target graph represents a graph whose matching in a host graph G is to be replaced. If T is a subgraph of a context graph C, then the occurrence of T within the matching of C in G is to be replaced. The graph T is part of the left-hand side of graph transformation rules. In case T consists of a single node, it is called* target node *and denoted by t.*
- *Replacement Graph R. A replacement graph represents a graph of which an isomorphic copy is used to replace a matching of the target graph T in the host graph. The graph R is part of the right-hand side of graph transformation rules.*
- *The nodes of the host graph that are connected to the matching of T are called* cut nodes.

Informally, a graph grammar is a collection of graph transformation rules each of which is equipped with a set of embedding instructions. The subsequent definition provides the necessary syntax and semantics.

Definition 3.4 (Context-Free Design Graph Grammar) *A context-free design graph grammar is a tuple $\mathcal{G} = \langle \Sigma, P, s \rangle$ with*

- *Σ is the set of node labels and edge labels*[1]*,*
- *P is the finite set of graph transformation rules, and*
- *s is the initial symbol.*

The graph transformation rules in P are of the form $T \rightarrow \langle R, I \rangle$ with

- *$T = \langle V_T, E_T, \sigma_T \rangle$ is the target graph to be replaced,*
- *$R = \langle V_R, E_R, \sigma_R \rangle$ is the possibly empty replacement graph,*
- *I is the set of embedding instructions.*

  Semantics: Firstly, a matching of the target graph T is searched within the host graph G. Secondly, this occurrence of T along with all incident edges is deleted. Thirdly, an isomorphic copy of R is connected to G according to the semantics of the embedding instructions.

  An embedding instruction in I is a tuple $((h, t, e), (h, r, f))$ with

  - *$h \in \Sigma$ is a label of a node v in $G \setminus T$,*
  - *$t \in \Sigma$ is a label of a node w in T,*
  - *$e \in \Sigma$ is the edge label of $\{v, w\}$,*
  - *$f \in \Sigma$ is another edge label not necessarily different from e, and*
  - *$r \in V_R$ is a node in R.*

  Semantics: If there is an edge labeled e connecting a node labeled h in $G \setminus T$ with a node labeled t in T, then a new edge with label f is created, connecting the node labeled h with the node r. If no edge labels are used, an embedding instruction may be abbreviated as $((h, t), (h, r))$.[2]

The execution of a graph transformation rule r on a host graph G yielding a new graph G' is called "derivation step" and is denoted by $G \Rightarrow_r G'$. A sequence of such derivation steps is called derivation*. The set of all graphs that can be generated with $\mathcal{G}$ is designated by $L(\mathcal{G})$.*

[1] Within rules, labels can also be used as variables for other labels (cf. Page 70). To avoid confusion, variable labels will be denoted with capital letters, while all other (type-specifying) labels get small letters.

[2] Grammars that do not change edge labels in the embedding process are called neighborhood uniform grammars.

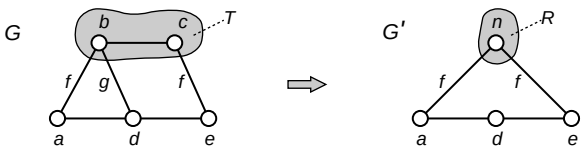
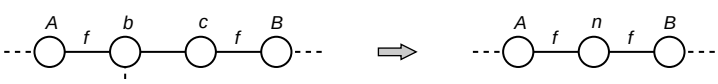


Figure 3.5. Application of a context-free graph transformation rule, $T \rightarrow \langle R, I \rangle$, to a host graph G.

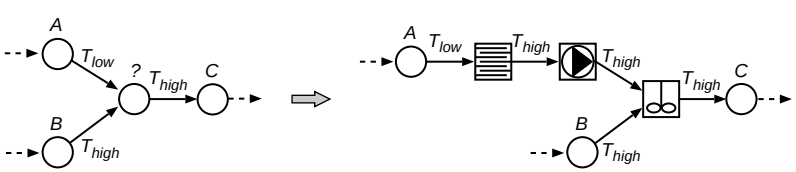
Example 1. The transformation of a graph G into a graph G', depicted in Figure 3.5, illustrates how a graph transformation rule works. The rule $T \rightarrow \langle R, I \rangle$ has the following components.

- Target graph $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2\}, \{\{1,2\}\}, \{(1,b),(2,c)\} \rangle$
- Replacement graph $R = \langle V_R, E_R, \sigma_R \rangle = \langle \{3\}, \{\}, \{(3,n)\} \rangle$
- Embedding instructions $I = \{((a,b,f),(a,n,f)),((e,c,f),(e,n,f))\}$. Alternatively, one can employ variable labels, which yields $I = \{((A,B,f),(A,n,f))\}$.
- A graphical specification of the rule may look as follows. The dashed lines represent an arbitrary number (inclusive zero) of incident edges.



Example 2. Let $\mathcal{G} = \langle \Sigma, P, s \rangle$ be a design graph grammar from the domain of chemical engineering that specifies some transformation from Section 3.1. $\Sigma = \{?, A, B, C, D, T_{low}, T_{high}, heater, pump, mixer\}$, P contains rules similar to the following.

$$T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3,4\}, \{(1,3),(2,3),(3,4)\}, \{(1,A),(2,B),(3,?),$$
$$(4,C),((1,3),T_{low}),((2,3),T_{high}),((3,4),T_{high})\} \rangle$$
$$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{5,6,7,8,9,10\}, \{(5,7),(7,8),(8,9),(6,9),(9,10)\}, \{(5,A),$$
$$(6,B),(7,heater),(8,pump),(9,mixer),(10,C),((5,7),T_{low}),$$
$$((7,8),T_{high}),((8,9),T_{high}),((6,9),T_{high}),((9,10),T_{high})\} \rangle$$
$$I = \{((D,A),(D,A)),((D,B),(D,B)),((D,C),(D,C))\}$$



Context-Sensitive Design Graph Grammars

This subsection introduces the notion of context into design graph grammars, leading to the context-sensitive variant of Definition 3.4.

Definition 3.5 (Context-Sensitive Design Graph Grammar) *A context-sensitive design graph grammar is a tuple $\mathcal{G} = \langle \Sigma, P, s \rangle$ as described in Definition 3.4 whose graph transformation rules P are of the form $\langle T, C \rangle \rightarrow \langle R, I \rangle$ with*

- *$T = \langle V_T, E_T, \sigma_T \rangle$ is the target graph to be replaced,*
- *C is a supergraph of T, called the context,*
- *$R = \langle V_R, E_R, \sigma_R \rangle$ is the possibly empty replacement graph,*
- *I is the set of embedding instructions for the replacement graph R.*

  Semantics: Firstly, a matching of the context C is searched within the host graph G. Secondly, an occurrence of T within the matching of C along with all incident edges is deleted. Thirdly, an isomorphic copy of R is connected to G according to the semantics of the embedding instructions.

  The set I of embedding instructions consists of tuples of the same form and semantics as in the context-free case in Definition 3.4.

In the following we will not explicitly distinguish between both graph grammar types, since the used variant is obvious from the context and rule form.

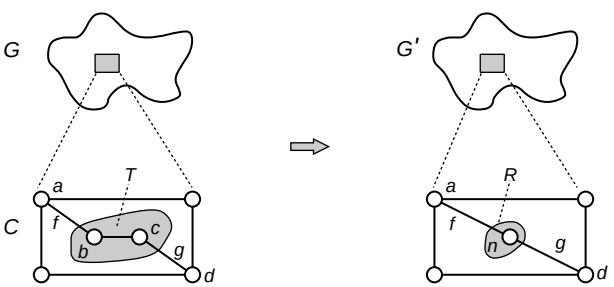


Figure 3.6. Application of a context-sensitive graph transformation rule, $\langle T, C \rangle \rightarrow \langle R, I \rangle$, to a host graph G.

Example. The transformation of a graph G into a graph G', depicted in Figure 3.6, illustrates how a context-sensitive graph transformation rule works. The rule $\langle T, C \rangle \rightarrow \langle R, I \rangle$ has the following components.

$$
\begin{aligned}
T &= \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2\}, \{\{1,2\}\}, \{(1,b),(2,c)\} \rangle \\
C &= \langle V_C, E_C, \sigma_C \rangle = \langle \{3,4,5,6,7,8\}, \{\{3,4\},\{3,7\},\{7,8\},\{4,8\},\{3,5\}, \\
&\qquad \{5,6\},\{6,8\}\}, \{(3,a),(5,b),(6,c),(8,d), \\
&\qquad (\{3,5\},f),(\{6,8\},g)\} \rangle \\
R &= \langle V_R, E_R, \sigma_R \rangle = \langle \{9\}, \{\}, \{(9,n)\} \rangle \\
I &= \{((a,b,f),(a,n,f)),((d,c,g),(d,n,g))\}
\end{aligned}
$$

With variable labels, the set I may also be formulated as $\{((A,B,C),(A,n,C))\}$.

Remarks. Design graph grammars differ not only in matters of context but also in the size of the target graph. If all target graphs in the graph transformation rules consist of single nodes, the graph grammar is called *node-based*, otherwise it is called *graph-based*. The Figures 3.7 and 3.8 illustrate some cases where node-based graph transformation rules are insufficient. Note that such rules are required for optimization and repair tasks.

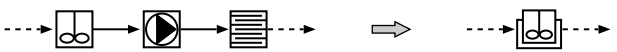


Figure 3.7. Replacement of a partial chain consisting of a mixer, a pump, and a heat transfer unit by a mixer device with built-in heat transfer. Again, the dashed lines stand for a variable number of edges. For the sake of simplicity edge labels have been omitted.

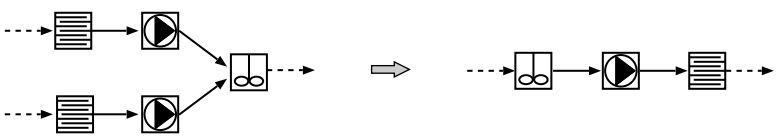


Figure 3.8. Combination of two identical partial chains through relocation. Depending on the properties of the substances involved, a different mixer device has to be used.

The distinction between node-based and graph-based graph grammars is of relevance, since they fall into different complexity classes due to the subgraph matching problem connected to the latter (see Section 3.4, Page 78).

On the Semantics of Labels

Labels are of paramount importance for the graph transformation process; all tasks belonging to a transformation step such as matching of target and context graphs or the embedding of replacement graphs rely on them. This section addresses some issues related to labels: terminal and nonterminal labels, variable labels, and conflicting embedding instructions.

Terminal and Nonterminal Labels Several graph grammar approaches distinguish between terminal and nonterminal labels: Terminal labels may appear only within the right-hand sides of graph transformation rules; nonterminal labels are used within both sides. In this regard, a graph is called terminal or final if it contains only terminal labels.

Design graph grammars use the classic idea of graph matching, and there is no syntactical distinction with respect to terminals and nonterminals in the set Σ. Note that this philosophy does also reflect the human understanding of modeling in many technical domains. For the same reason, the concept of final graphs has been abandoned.

Variable Labels Variable labels are introduced for convenience purposes: They allow for the formulation of generic rules, which match situations belonging to identical topologies that differ with respect to their labels. Without variable labels one rule for each such situation would have to be instantiated, leading to a large rule set due to the combinatorial explosion.

Furthermore, the use of variable labels within rules and embedding instructions leads to the question of binding: Variable labels that occur exclusively within the embedding instructions of some rule are called *unbound*, and their matching is not restricted. Variable labels that occur within the context graph or target graph of a rule are bound to the label of the actually matched node; they retain their value during the replacement and embedding processes.

Observe that variable labels prevent a simple distinction between terminals and nonterminals: The labels in Σ cannot be assigned to either class by a superficial analysis of the graph transformation rules in P.

If the use of variable labels leads to conflicting embedding instructions, the *principle of the least commitment* shall apply (230): The most specialized embedding instruction is to be chosen.

3.3 Relation to Classical Graph Grammars

An important question pertains to the justification of design graph grammars, which represent a further graph transformation formalism amidst existing graph grammar concepts. In the following we describe the two general approaches to graph transformation along with their most prominent graph grammar representatives and point out their advantages and disadvantages. We then compare design graph grammars with the classical graph grammars and establish their relationships.

The Connecting Approach

The connecting approach is a node-centered concept that aims at the replacement of nodes or subgraphs by graphs. The item to be replaced is deleted along with all incident edges, and the replacement graph is embedded into the host graph by connecting both with new edges. These new edges are constructed by means of some mechanism that specifies the embedding.

In the literature, graph grammars are often distinguished by the size of the left-hand sides of rules, leading to two approaches of inherently different complexity: node replacement graph grammars and graph replacement graph grammars. Additionally, each of these two approaches is divided into context-free and context-sensitive subclasses.

Several graph grammars follow the connecting approach. According to (66), the most well-known node replacement graph grammar families are the *node label controlled* (NLC) and the *neighborhood controlled embedding* (NCE) graph grammars, whose node-based versions we describe in the following.

NLC Graph Grammars Node label controlled graph grammars perform graph transformations on undirected node-labeled graphs. A graph transformation step is based merely on node labels, i. e., there are no application conditions or contexts to be matched. The embedding is determined by a set of embedding instructions shared by all graph transformation rules. The following definition resembling the one of (66) introduces NLC grammars formally.

Definition 3.6 (NLC Graph Grammar) *An NLC graph grammar is a tuple* $\mathcal{G} = \langle \Sigma, P, I, s \rangle$ *with*

- Σ *is the set of terminal and nonterminal node labels,*[3]
- P *is the finite set of graph transformation rules of the form* $t \to R$*, where* $t \in \Sigma$ *and* R *is a labeled graph,*

[3] In the graph grammar literature it is usually distinguished between different label sets. For the sake of clarity, we use a single set Σ containing all labels.

- *I is a set of embedding instructions, and*
- *s is the initial symbol.*

An embedding instruction $(h, r) \in I$ states that the embedding process creates an edge connecting each node of the replacement graph labeled r with each node of the host graph labeled h that is adjacent to the target node.

Remarks. The domain of technical systems imposes requirements some of which cannot be met by NLC grammars: (1) There is no way to specify a context. Therefore, it is not possible to distinguish between different situations related to a single item. (2) There is no way to distinguish between individual nodes in the replacement graph, since the embedding mechanism relies solely on labels.

NCE Graph Grammars Neighborhood controlled embedding graph grammars perform graph transformations on directed or undirected labeled graphs.[4] A graph transformation step is also based on edge labels, which provide further discerning power. The embedding is determined by a set of embedding instructions belonging to each graph transformation rule.

Definition 3.7 (NCE Graph Grammar) *An NCE graph grammar is a tuple $\mathcal{G} = \langle \Sigma, P, s \rangle$ with*

- *Σ is the set of terminal and nonterminal node labels and edge labels,*
- *P is the finite set of graph transformation rules, and*
- *s is the initial symbol.*

The graph transformation rules in P are of the form $t \to \langle R, I \rangle$ with

- *$t \in \Sigma$ is the label belonging to a node v in the host graph,*
- *$R = \langle V_R, E_R, \sigma_R \rangle$ is the non-empty replacement graph,*
- *I is the set of embedding instructions for the replacement graph R and consists of tuples $(h, e/f, r)$ where*
  - *$h \in \Sigma$ is a node label and $e \in \Sigma$ is an edge label in the host graph,*
  - *$f \in \Sigma$ is another edge label, and*
  - *$r \in V_R$ is a node of the replacement graph.*

[4] In the literature on the subject, NCE grammars with and without edge labels or edge directions are distinguished by the prefixes "e" (for edge labels) and "d" (for directed edges) added to the NCE acronym. Thus, there are NCE, eNCE, dNCE and edNCE graph grammars. We will omit these prefixes and use always NCE.

An embedding rule $(h, e/f, r)$ *has the same meaning as in Definition 3.4, where it is written as* $((h, t, e), (h, r, f))$.

Remarks. Despite their superiority over NLC graph grammars, the weak context mechanisms of NCE graph grammars render them unusable for our purposes: (1) Context descriptions are restricted to the incident edges of the target node. (2) Contexts are treated not until embedding time; i. e., they cannot serve as application condition at matching time.

The Gluing Approach

The gluing approach is an edge-centered concept that aims at the replacement of hyperedges or hypergraphs by hypergraphs. Each hyperedge or hypergraph possesses a series of attachment nodes. Within a replacement step, the item to be replaced is deleted from the host hypergraph with exception of the attachment nodes, which are at the same time external nodes of the host hypergraph; the new hypergraph is embedded in its place by unifying (gluing) its attachment nodes with the external nodes.

There exist several hypergraph grammar types following the gluing paradigm. The most well-known family is called *hyperedge replacement* (HR).

HR Grammars Similarly to the connecting approach case, where node-based and graph-based grammars are distinguished, we distinguish between hyperedge-based and hypergraph-based HR grammars. Hyperedge-based HR grammars are defined as in (58).

Definition 3.8 (Hyperedge Replacement Grammar) *A hyperedge replacement grammar is a tuple* $\mathcal{G} = \langle \Sigma, P, s \rangle$ *where*

- Σ *is the set of terminal and nonterminal hyperedge labels,*
- P *is the finite set of hypergraph transformation rules over* Σ *each of which has the form* $T \to R$*, where* T *is a hyperedge label and* R *is the replacement hypergraph, and*
- s *is the initial symbol.*

Remarks. Like the NLC and NCE graph grammars, HR grammars are also not powerful enough for the design tasks envisioned: (1) HR grammars are intrinsically context-free, since the item of the left-hand side of a rule is completely deleted and replaced by the hypergraph of the right-hand side. Context that shall function as application condition must be integrated into the target. (2) With respect to their expressiveness HR grammars are weaker than the NCE grammars (66, pg. 4).

Hybrid Approaches Apart from design graph grammars there exist other hybrid approaches in the literature. In (44) Courcelle et al. present the so-called handle hypergraph grammar. This hybrid graph grammar is based on the hyperedge replacement approach and has some additional node replacement features. A similar approach that has a simpler rewriting mechanism is the HR grammar with eNCE rewriting, presented in (135). Another hybrid approach, the hypergraph NCE graph grammar, is introduced in (142); this concept is based on the node replacement approach.

Design Graph Grammars

As seen in the previous section, neither classical node replacement nor hyperedge replacement grammars provide sufficient means for solving technical design tasks; even the powerful hybrid approaches proved to be inadequate for our needs (241). Design graph grammars, on the other hand, represent an approach combining the strengths of node replacement and hyperedge replacement grammars while overcoming their weaknesses (see Figure 3.9).

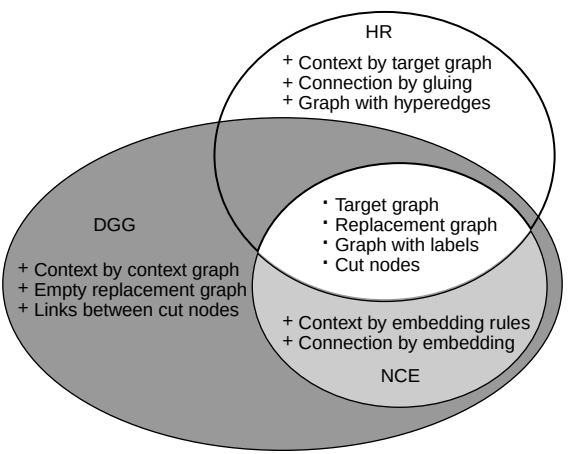


Figure 3.9. Relationship of design graph grammars to the classical grammars with respect to their features.

The core of the design graph grammar approach is node replacement based, though:

- *Replacement Paradigm.* Concise formulation of node-based graph transformation rules (NLC/NCE).
- *Embedding.* Access to individual nodes of the replacement graph (NCE); unique embedding through attachment nodes (HR).

Furthermore, design graph grammars add some features on their own:

- *Context.* Context that serves as application condition.
- *Matching.* Fine grained control of matching.
- *Embedding.* Extended replacement graph formulation; enhanced embedding instructions; flexible rule formulation by means of variable labels.

In the following we present some formal results that establish the relationship between design graph grammars and the classical graph and hypergraph grammars. The set $\mathcal{L}_{Class}$ comprises for each grammar $\mathcal{G} \in Class$ the set of graphs, $L(\mathcal{G})$, that can be generated by $\mathcal{G}$.

Theorem 3.1 ($\mathcal{L}_{NLC} \subseteq \mathcal{L}_{DGG}$) *Every NLC graph language generated by a node-based NLC graph grammar can be generated by a node-based, context-free design graph grammar.*

Proof. *Let an arbitrary NLC grammar $\mathcal{G} = \langle \Sigma, P, I, s \rangle$ for an NLC graph language L be given. We construct a node-based, context-free DGG $\mathcal{G}' = \langle \Sigma', P', s' \rangle$ based on $\mathcal{G}$ such that $L(\mathcal{G}') = L$.*

Obviously, $s' = s$ and $\Sigma' = \Sigma$. The set of graph transformation rules P' is defined as follows. P' contains a graph transformation rule $r' : t \to \langle R, I' \rangle$ for each $r \in P$ with $r : t \to R$; $I' = I$. For each embedding instruction $i \in I$ with $i = (h, r)$ there is an embedding instruction $i' \in I'$ with $i' = ((h, t, \bot), (h, v, \bot))$, where $\sigma_R(v) = r$. It is clear that $L(\mathcal{G}') = L$. ◇

Theorem 3.2 ($\mathcal{L}_{NCE} \subseteq \mathcal{L}_{DGG}$) *Every NCE graph language generated by a node-based NCE graph grammar can be generated by a node-based, context-free design graph grammar, whereas the opposite does not hold.*

Proof. *Taking an arbitrary NCE grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ for an NCE graph language L as a starting point, we construct a node-based, context-free DGG $\mathcal{G}' = \langle \Sigma', P', s' \rangle$ whose generated language $L(\mathcal{G}') = L$.*

Due to the similarity between both concepts, the construction is straightforward. We set $\Sigma' = \Sigma$, $P' = P$, and $s' = s$. The graph transformation rules are identical in syntax and semantics for both concepts; only the syntax of the embedding instructions differ: For each NCE embedding instruction $i \in I$ with $i = (h, e/f, r)$ there is a DGG embedding instruction $i' \in I'$ with $i' = ((h, t, e), (h, r, f))$. Obviously, $L(\mathcal{G}') = L$. ◇

Theorem 3.3 ($\mathcal{L}_{HR} \subseteq \mathcal{L}_{DGG}$) *Every HR language generated by a hyperedge-based HR grammar can be generated by a node-based, context-free design graph grammar, if hypergraphs are interpreted as bipartite graphs.*

Proof. According to Engelfriet and Rozenberg (66, pg. 57), $\mathcal{L}_{B_{nd}-edNCE} = \mathcal{L}_{HR}$. Hence, HR languages generated by HR grammars can be generated by nonterminal neighbor deterministic boundary edNCE grammars, which in turn can be simulated by DGGs, since $\mathcal{L}_{B_{nd}-edNCE} \subseteq \mathcal{L}_{B-edNCE} \subseteq \mathcal{L}_{edNCE}$. Thus, DGGs can generate HR languages and it follows that $\mathcal{L}_{HR} \subseteq \mathcal{L}_{DGG}$. ◇

Figure 3.10 summarizes the above statements by illustrating the expressive power of design graph grammars.

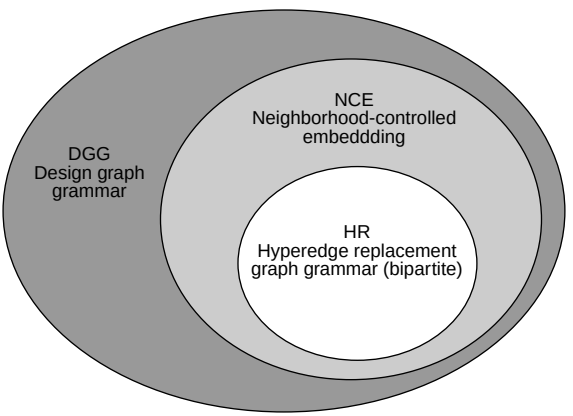


Figure 3.10. The expressive power of design graph grammars with respect to the languages that can be generated.

Relationship to Programmed Graph Replacement Design graph grammars as proposed here shall enable domain experts to formulate design expertise for various design tasks. Design graph grammars result from the combination of different features of the classical graph grammar approaches, while attention has been paid to keep the underlying formalism as simple as possible.

When comparing design graph grammars to programmed graph replacement systems (PGRS) one should keep in mind that the former is located at the conceptual level while the latter emphasizes the tool character. PGRS are centered around a complex language allowing for different programming approaches. PROGRES [5], for instance, offers declarative and procedural elements (244, 245) for data flow oriented, object oriented, rule based, and imperative programming styles. A direct comparison between PROGRES to the concept of design graph grammars is of restricted use only and must stay at the level of abstract graph transformation mechanisms.

[5] We chose PROGRES for illustration purposes only; the line of argumentation applies to other tools such as PAGG (see (247) for a description and further pointers) or FUJABA (197) as well.

However, it is useful to relate the concepts of design graph grammars to PGRS under the viewpoint of operationalization. PGRS are a means—say: one possibility—to realize a design graph grammar by reproducing its concepts. In this connection PROGRES fulfills the requirements of design graph grammars for the most part. However, PROGRES lacks the design graph grammar facilities for the formulation of context, deletion operations, and matching control, which have to be simulated by means of complex rules. Such a kind of emulation may be useful as a prototypic implementation, but basically, it misses a major concern of design graph grammars: Their intended compactness, simplicity, and adaptiveness with respect to a concrete domain or task.

3.4 Structure Analysis and Design Evaluation

The foregoing sections introduced design graph grammars as a concept to describe the transformation of a structure model $S\langle F, \mathcal{M}\rangle$ into another structure model $S'\langle F', \mathcal{M}'\rangle$ and, this way, as a means to encode model construction knowledge. Mainly in this sense design graph grammars are used within Part II of this work.

Note that this is only the descriptive side of the coin. The other is that design graph grammars possess the potential to automate demanding reasoning tasks respecting the analysis and synthesis of structure models. The basic ideas and related theorems are explained now.

The next subsection focuses on the analysis question: Does a design, say, a structure model $S\langle F, \mathcal{M}\rangle$ fulfill a given set of technical constraints? The question can be answered by solving the membership problem for the related graph G, whereas the constraints are defined implicitly by a task- and domain-specific graph grammar $\mathcal{G}$. The next but one subsection, starting at Page 81, presents a new answer to the problem of design evaluation: How good is a design, say, a structure model $S\langle F, \mathcal{M}\rangle$? Here, the question is answered by computing the "distance" between $S\langle F, \mathcal{M}\rangle$ and an optimum solution $S^*\langle F^*, \mathcal{M}^*\rangle$ that has been provided by a human designer.

Analysis Means to Solve the Membership Problem

Structure analysis means to solve the membership problem for a given graph G and a graph grammar $\mathcal{G}$. This job requires to find a derivation of G from the start symbol s; a derivation in turn is based on the application of graph transformation rules defined in $\mathcal{G}$; and, to fire a rule it is necessary that a matching of the left-hand side be found within the host graph.

Figure 3.11 hints the connection between the mentioned problems. Moreover, the figure shows two rectangles with special properties and graph grammars: Associativity, confluence, boundary, leftmost, precedence graph, and flowgraph. In the general case, the membership problem is PSPACE-complete (28); however, the listed concepts may decisively reduce the complexity of the graph membership problem.

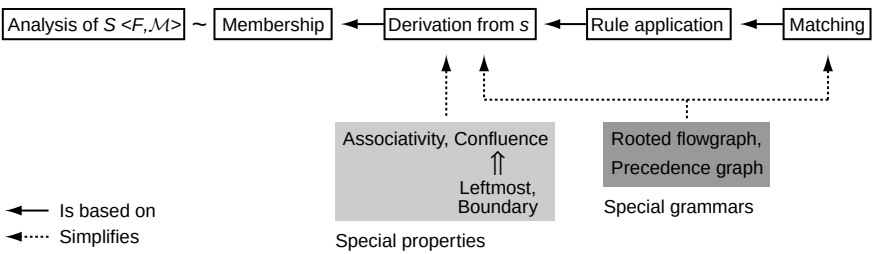


Figure 3.11. Structure analysis can be tackled by graph membership, which in turn is based on derivation, rule application, and matching. The rectangles below contain properties and graph grammars that reduce the complexity of the membership problem.

They are introduced in the remainder of this subsection and—if possible—related to design graph grammars.

Remarks. Matching is already a nontrivial issue in the context-free, graph-based case, as implied by the subgraph matching problem[6], which is NP-complete (89, 145). The inclusion of context adds to the complexity of matching, because node-based matchings with context are comparable to graph-based matchings.

Associativity and Confluence A graph grammar can be regarded as the equivalent of a Chomsky grammar for the area of labeled (multi)graphs, and as such it has similar properties. Because of its far-reaching consequences, the probably most important property is *confluence*: Particular NP- or PSPACE-complete problems related to graph grammars that have, among others, this property can be solved in polynomial time, such as the membership problem. However, before we proceed with the definition of confluence, we provide some other basic notions.

Lemma 3.1 (Associativity of Design Graph Grammars) *Let $\mathcal{G} = \langle \Sigma, P, s \rangle$ be a design graph grammar with graph transformation rules $T_1 \to \langle R_1, I_1 \rangle$ and $T_2 \to \langle R_2, I_2 \rangle$. Moreover, let G be a host graph and R_1 contain a matching of T_2. Then the following equation holds.*

$$G[T_1|R_1][T_2|R_2] = G[T_1|R_1[T_2|R_2]]$$

Definition 3.9 (Confluence (66)) *A context-free design graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ is confluent, if for every pair of rules $T_1 \to \langle R_1, I_1 \rangle$ and $T_2 \to \langle R_2, I_2 \rangle$, R_i containing a matching of $T_{i,\, i \in \{1,2\}}$, and for any arbitrary host graph G containing matchings of T_1 and T_2, the following equality holds.*

$$G[T_1|R_1][T_2|R_2] = G[T_2|R_2][T_1|R_1]$$

[6] In the field of graph theory this problem is known as the subgraph isomorphism problem. It should not be mistaken with the graph isomorphism problem, which lies in NP, but for which it is still open whether it is NP-complete (89, 13, 145, 179).

Put in other words, a design graph grammar is confluent if the sequence of rule application is irrelevant with respect to the set of derivable graphs.

The following definition of confluence develops from the definition of confluence for edNCE grammars in (66); it is more detailed and makes an a-priori statement possible.

Definition 3.10 (Confluence 2) *A context-free design graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ is confluent, if for all graph transformation rules $T_1 \rightarrow \langle R_1, I_1 \rangle$ and $T_2 \rightarrow \langle R_2, I_2 \rangle$ in P, all nodes $v_1 \in V_{R_1}, v_2 \in V_{R_2}$, and all edges labels $\alpha, \delta \in \Sigma$, the following equivalence holds.*

$$\exists \beta \in \Sigma : ((t_2, t_1, \alpha), (t_2, v_1, \beta)) \in I_1 \text{ and } ((\sigma(t_1), t_2, \beta), (\sigma(t_1), v_2, \delta)) \in I_2$$
$$\Leftrightarrow$$
$$\exists \gamma \in \Sigma : ((t_1, t_2, \alpha), (t_1, v_2, \gamma)) \in I_2 \text{ and } ((\sigma(t_2), t_1, \gamma), (\sigma(t_2), v_1, \delta)) \in I_1$$

Remarks. The definition allows for an algorithmic confluence test of context-free design graph grammars. Observe that the presence of destructive graph transformation rules makes a confluence statement improbable.

Theorem 3.4 (Context-Free Design Graph Grammars and Confluence) *Context-free design graph grammars are not inherently confluent.*

Proof. *Let $\mathcal{G} = \langle \Sigma, P, s \rangle$ be a context-free design graph grammar and $G = \langle \{v\}, \{\}, \{(v, t_1)\} \rangle$ a host graph. Let $r_1, r_2 \in P$ be two graph transformation rules as follows.*

r_1: $t_1 \rightarrow \langle R_1, I_1 \rangle$ with $R_1 = \{\}$ and $I_1 = \{\}$.

r_2: $t_1 \rightarrow \langle R_2, I_2 \rangle$ with $R_2 = \langle \{v_1, v_2\}, \{(v_1, v_2)\}, \{(v_1, t_1), (v_2, t_2), (\{v_1, v_2\}, e)\} \rangle$ and $I_2 = \{\}$.

With these two rules the subsequent derivations are possible.

(1) $G \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} G_{12} = \{\}$

(2) $G \Rightarrow_{r_2} G_2 \Rightarrow_{r_1} G_{21} = \langle \{v_2\}, \{\}, \{(v_2, t_2)\} \rangle$

Since $G_{12} \neq G_{21}$, $\mathcal{G}$ is not confluent. ◇

Leftmost and Boundary We present now two restrictions to node-based design graph grammars each of which implies confluence or even stronger properties: Leftmost derivation and boundary transformation rules. The following definitions and results are based on (66).

Leftmost derivations of design graph grammars are achieved by imposing a linear order on the nodes of the right-hand sides of the graph rules—this is necessary since there is no natural linear order as in the case of string grammars.

Definition 3.11 (Ordered Graph, Ordered Design Graph Grammar) *A graph $G = \langle V, E, \sigma \rangle$ is an ordered graph, if there is a linear order $(v_1, \ldots, v_n)$ with $v_i \in V$ for $1 \leq i \leq n$ and $n = |V|$. A design graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ is ordered if for each rule $t \rightarrow \langle R, I \rangle$ in P the replacement graph R is ordered.*

Let $\mathcal{G}$ be an ordered design graph grammar containing a graph transformation rule $t \rightarrow \langle R, I \rangle$. When embedding the replacement graph R with order $(w_1, \ldots, w_R)$ into a host graph G with order $(v_1, \ldots, v_{i-1}, t, v_{i+1}, \ldots, v_G)$, the order of the resulting graph G' is constructed as follows: $(v_1, \ldots, v_{i-1}, w_1, \ldots, w_R, v_{i+1}, \ldots, v_G)$.

Definition 3.12 (Leftmost Derivation) *Let $\mathcal{G}$ be an ordered design graph grammar. For an ordered graph G a derivation step $G \Rightarrow_{v,p} G'$ generated by $\mathcal{G}$ is a leftmost derivation step if v is the first nonterminal node in the order of G; p represents the graph transformation rule used. A derivation is leftmost if all its steps are leftmost. The graph language leftmost generated by $\mathcal{G}$ is denoted by $L_{lm}(\mathcal{G})$.*

Lemma 3.2 (Expressiveness of Leftmost Generated Languages) *Let $\mathcal{G}$ be an ordered design graph grammar. Then $L_{lm}(\mathcal{G})$ does not depend on the sequence of rule applications.*

Proof. See (66, pg. 40) where Engelfriet and Rozenberg show that the restriction to leftmost derivations is equivalent to the restriction to confluent grammars. The same argumentation holds for design graph grammars. ◇

The boundary property defines a subclasses of the whole class of design graph grammars by restricting the form the transformation rules in P.

Definition 3.13 (Boundary Design Graph Grammar (229)) *A design graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ is called boundary if one of the following properties holds for every graph transformation rule $T \rightarrow \langle R, I \rangle$.*

(1) R does not contain adjacent nonterminal nodes.

(2) I does not contain embedding instructions $((h, t, e), (h, x, f))$ where h is nonterminal.

Lemma 3.3 (Expressiveness of Boundary Design Graph Grammars) *Every boundary design graph grammar is confluent.*

Proof. According to Engelfriet and Rozenberg, boundary graph grammars are confluent by definition; this follows from property (2) of Definition 3.13 (66, pg. 56). ◇

Remarks. Leftmost derivations and boundary design graph grammars come along with interesting properties: (1) Confluent design graph grammars are associative. (2) The membership problem for confluent design graph grammars is in NPTIME

(66, pg. 82). (3) The membership problem for boundary design graph grammars is in PTIME, if, due to labeling restrictions, the subgraph matching problem can be solved in polynomial time (251).

One way to make a node-based design graph grammar boundary is the insertion of additional terminal nodes (junctions) into rules that have adjacent nonterminals on the right-hand side; this will ensure property (1) of Definition 3.13.

Rooted Flowgraph and Precedence Graph Grammars For rooted context-free flowgraph languages and for languages generated by precedence graph grammars the membership problem can be solved in polynomial time. Because of this remarkable property they are subject matter of this paragraph.

Flowgraph languages supply a suitable mechanism to represent the control flow of source programs. They have a strong resemblance to series-parallel graphs, to which the graphs generated by our design graph grammars for chemical engineering (see Section A.2) are also similar. To test whether a given graph belongs to the language of rooted context-free flowgraphs, both the given graph and the flowgraph grammar are serialized by imposing ordered spanning trees. Based on the tree ordering, membership recognition can be realized in polynomial time in the size of the input graph. Whether a similar algorithm for the membership problem for non-rooted flowgraph grammars exists still remains an open problem. Details may be found in (162) and (161).

Precedence graph grammars are context-free graph grammars that have been enriched with precedence relations. For this grammar class the membership problem is decidable in $O(n^2)$ time if certain conditions are met, where n designates the number of nodes of the input graph. In a precedence graph grammar every pair of adjacent nodes, (v, w), gets assigned a precedence determining whether v is to be processed before, after, or in parallel to w. Moreover, the precedence relation must be unique, the graph grammar must be confluent, and its productions reversible. Details may be found in (132).

Evaluation Means to Compute the Graph Distance

The computer-based evaluation of design solutions is of upcoming importance, especially against the background that design problem solving on a computer becomes better and better (264, 110). This subsection presents a new answer to the question of—what we call—a solution's *relative design quality*. The relative design quality designates a quality assessment that employs a benchmark in the form of another design solution for the same problem.[7]

[7] The concept of relative design quality has been applied to evaluate the generated designs within Case Study A.1 (cf. Page 106). Clearly, the concept can have its pitfalls, which are connected with the structure of the solution space.

Given two design solutions, i. e. in our case, two graphs G and G^* that encode a machine-generated structure model $S\langle F, \mathcal{M}\rangle$ and a well-designed or "optimum" human-generated structure model $S^*\langle F^*, \mathcal{M}^*\rangle$, then the similarity between G and G^* tells us a lot about the quality of G.

The standard approach to similarity assessment is to weigh meaningful feature vectors of the interesting objects (7, 223, 301), an avenue which we follow in Section A.1 and C.2. Observe that such a compiled view is not necessary here, since we can fall back on knowledge from first principles: When using the graph grammar $\mathcal{G}$ to derive G from G^* (or vice versa), the complexity of the derivation can be interpreted as a measure of their similarity. Since this way follows the constructional principles of the domain, the complexity of the derivation corresponds to the true transformational effort—under certain conditions. These conditions relate to the concepts of shortest derivation, monotonicity, and shortcut-free, which are introduced now. Figure 3.12 illustrates the connections.

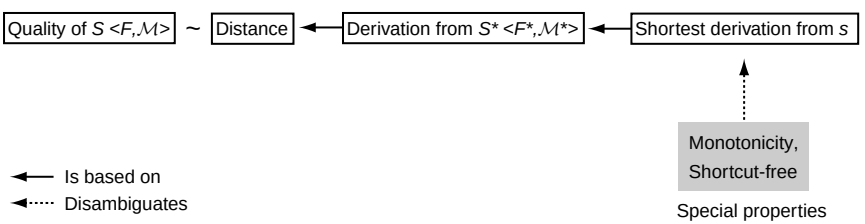


Figure 3.12. The evaluation of a structure model can be tackled by computing its distance, say by deriving it from an optimum solution $S^*\langle F^*, \mathcal{M}^*\rangle$.

Shortest Derivation The length of a derivation is a measure for the complexity of a design. Depending on the design graph grammar used and on the order of graph transformation rules applied, a derivation will take at least linear time with respect to the size of the graph. On the other hand, the worst case runtime complexity for a derivation is unbound if cyclic partial derivations exist or if destructive graph transformation rules are applied. Thus, a statement can be ventured for the shortest derivation only.

Definition 3.14 (Derivation) *A derivation is a sequence of graphs $\pi = (G_1, \ldots, G_n)$ for which the derivation steps $G_i \Rightarrow_{r_i} G_{i+1}, i \in \{1, \ldots, n-1\}$, have been achieved by applying the graph transformation rules $r_i \in P$ (cf. Definition 3.4 on Page 66). $\pi_{\mathcal{G}}$ denotes some derivation based on graph transformation rules of the design graph grammar $\mathcal{G} = \langle \Sigma, P, s\rangle$, and $\pi_{\mathcal{G}}(G)$ denotes a derivation $(s, \ldots, G)$. A shortest derivation is denoted by π^*.*

Definition 3.15 (Derivation Rule Sequence) *Let $\pi = (G_1, \ldots, G_n)$ be a derivation. We define the derivation rule sequence belonging to π as $\rho_\pi = (r_1, \ldots, r_{n-1})$, where $G_i \Rightarrow G_{i+1}$ is achieved by applying the graph transformation rule r_i, $1 \leq i \leq n-1$.*

Typically, the human understanding of the design of technical systems bears a monotonic character. This means that the design process is constructive, deletion operations are avoided where possible, leading to a system with the smallest number of construction steps possible. The following definitions shed some light on this matter.

Definition 3.16 (Deletion Operation) *A deletion operation is a graph transformation step $G \Rightarrow G'$ such that*

$$|V_T| > |V_R| \quad or \quad |E_T| > |E_R|$$

where $\langle V_T, E_T, \sigma_T \rangle$ and $\langle V_R, E_R, \sigma_R \rangle$ designate the target and replacement graph respectively.

Remarks. Definition 3.16 tolerates that constructive graph transformation steps may perform partial deletions, as long as there are more insertions. Figure 3.13 illustrates the consequence of the presence of deletion operations within a derivation.

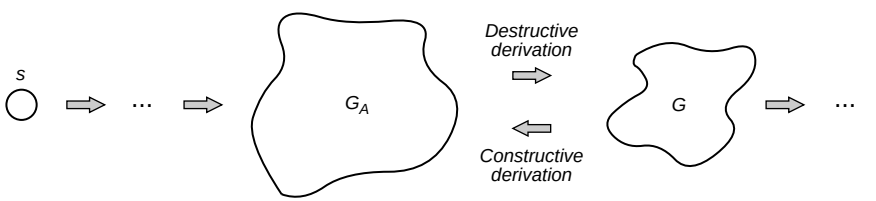


Figure 3.13. A derivation containing deletion operations. Due to possible cycles the derivation length is unbound.

With the aid of the above notions, the aforementioned restriction to rule structures can be introduced formally.

Definition 3.17 (Monotonicity, Shortcut-Free) *Let G, G' be graphs and $\mathcal{G}$ a design graph grammar. A derivation $\pi = (G, \ldots, G')$ is called monotonic, if and only if ρ_π does not involve deletion operations.*

- *$\mathcal{G}$ is monotonic, if and only if for every $G \in L(\mathcal{G})$ there exists a monotonic derivation $\pi_{\mathcal{G}}(G)$.*
- *$\mathcal{G}$ is called shortcut-free, if for every $G \in L(\mathcal{G})$ the shortest derivation is a monotonic derivation.*

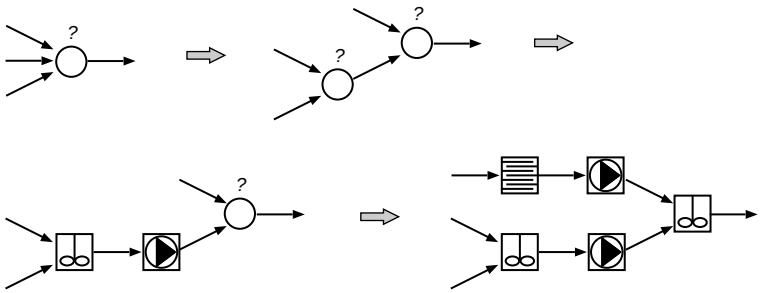


Figure 3.14. A monotonic derivation of a chemical plant's structure model.

Distance between Graphs Shortest derivations play a role when computing the distance between two graphs. As mentioned at the outset, the quality of a design G can be measured by the distance between G and the ideal design G^*, as provided by an expert. In practice, this is done by determining the necessary graph transformation steps required for the derivation $(G, \ldots, G^*)$ and calculating the involved effort.

Since our approach depends on a concrete design graph grammar within a given domain, it is assumed that the ideal design G^* is also derivable with the given grammar. In this connection we distinguish between the *direct* distance between two graphs and their *derivational* distance. Figure 3.15 illustrates the notions.

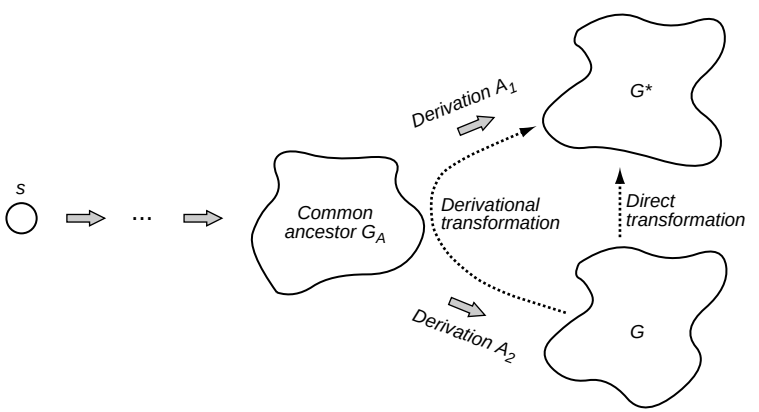


Figure 3.15. Distance between a design G and the ideal design G^* with respect to the design graph grammar derivation.

Talking about Figure 3.15, it is clear that the derivational distance between the two designs is equivalent to the effort necessary for the "derivation" $(G, \ldots, G_A, \ldots, G^*)$. Put in other words, the distance between G and G^* is bound by the effort required to transform G back into an ancestor G_A plus the effort required to derive

G^* from this common ancestor G_A.

Remarks. The concept of derivational transformation is brought into play since domain knowledge for $\pi_{\mathcal{G}}(G)$, i. e., for a construction from scratch, is available in many cases. In contrast, starting a design process with an existing structure model requires knowledge for an arbitrary design adaptation. Observe that in some favorable cases it may happen that a design G is an ancestor of the ideal design G^*.

Determining the Transformation Effort Determining the graph transformation sequence for the derivation $(G, \ldots, G^*)$ depends on two factors: The knowledge about the derivation of G and G^* and the desired granularity of the distance statement. With respect to the former it can be argued that the design G has been automatically generated—its derivation is therefore known; a derivation of G^*, if not available, must be determined by some heuristic search procedure (209).

As far as the desired granularity of the distance statement is concerned, a naive approach consisting of a simple comparison of derivations is conceivable. It implies an element-wise comparison of $\pi(G)$ and $\pi(G^*)$, i. e., to search for a graph G_A in $\pi(G) \cap \pi(G^*)$. A more elaborate approach involving finding the largest common ancestor results in a more meaningful upper bound for the derivational distance.

The search for a common ancestor is a nontrivial task involving the graph matching problem mentioned on Page 78; the search for the largest common ancestor is even more toilsome, since there may exist more than one derivation for a given graph. Note that this problem does not correspond to the NP-hard maximum common subgraph problem (146), although the algorithms described there could be used to find at least an approximation of the largest common ancestor.

Again, the monotonicity property proves to be a valuable feature of a design graph grammar: The absence of deletion operations reduces the search space considerably. A sufficient, even though rarely satisfiable condition for monotonicity is the following.

Lemma 3.4 (Monotonicity Condition) *Let a design graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ be given. $\mathcal{G}$ is monotonic if the following holds for every graph transformation rule $r = \langle T, C \rangle \rightarrow \langle R, I \rangle$, $r \in P$: R encompasses a matching of T.*

Put in other words, the target graph is a subgraph of the replacement graph.

After determining the graph transformation rules required for the derivation $(G, \ldots, G^*)$, the effort necessary for this transformation can be calculated from both the domain and the graph-theoretical point of view.

In order to take the domain into account, we introduce a function $c_{dom} : P \rightarrow \mathbf{R}_0^+$, which yields for a graph grammar $\mathcal{G} = \langle \Sigma, P, s \rangle$ the effort of the application of a rule $r \in P$ within the domain *dom*. Then, the overall domain effort when transforming a design according to a derivation π then is:

$$\mathit{effort}(\pi) = \sum_{r \in \rho_\pi} c_{dom}(r)$$

If a function c_{dom} cannot be stated, a function $c_{DGG} : P \rightarrow \mathbf{R}_0^+$ that computes a graph-theoretical effort, including aspects such as context and matching, must be used instead:

$$\mathit{effort}(\pi) = \sum_{r \in \rho_\pi} c_{DGG}(r)$$

Part II

Case Studies

A

Model Simplification

The term model simplification speaks for itself: By reducing a model's complexity a problem solving task in question shall become tractable. Within analysis tasks, model simplification aims at a reduction of a model's constraint complexity; with synthesis tasks, search space complexity is of paramount importance: It is a measure for the number of models that have to be synthesized and analyzed in order to solve a configuration or design problem. See Page 46, Section 2.4, for both an introduction to generic model simplification techniques and a comparison to other model construction approaches.

The two case studies of this chapter present approaches to solve complex design tasks. Both tasks comprise creative aspects, and for neither a design recipe is at hand: The acquisition effort for the design knowledge exceeds by far the expected payback (167), and, moreover, the synthesis search spaces are extremely large and scarcely to control—despite the use of knowledge-based techniques.

Two possibilities to counter this situations are "competence partitioning" and "expert critiquing". The idea of competence partitioning is to separate the creative parts of a design process from the routine jobs, and to provide a high level of automation regarding the latter (see (257, pg. 93) or (261)). Expert critiquing, on the other hand, employs expert system technology to assist the human expert rather than to automate a design problem in its entirety (108, 77).

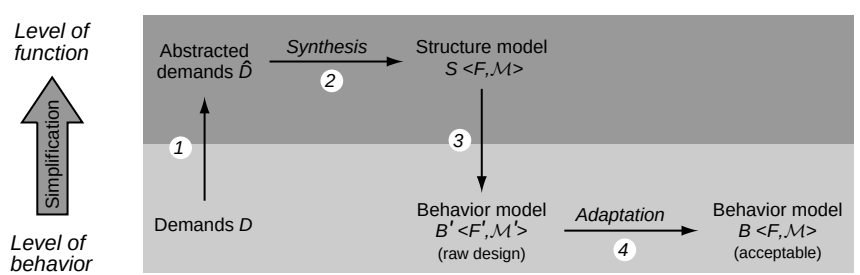


Figure A.1. The paradigm of functional abstraction in design problem solving.

The paradigm of "design by functional abstraction", illustrated in Figure A.1, can be regarded as a special expert critiquing representative. We have chosen this name for the problem solving method to reveal its similarity with the problem solving method HEURISTIC DIAGNOSIS, which became popular as the diagnosis approach underlying MYCIN (41)[1].

The key idea of design by functional abstraction is a *systematic* construction of candidate solutions within a very simplified design space, which typically is some structure model space. A candidate solution, $S\langle F, \mathcal{M}\rangle$, is transformed into a preliminary raw design, $B'\langle F', \mathcal{M}'\rangle$, by *locally* attaching behavior model parts to $S\langle F, \mathcal{M}\rangle$. The hope is that $B'\langle F', \mathcal{M}'\rangle$ can be repaired with reasonable effort, yielding an acceptable design $B\langle F, \mathcal{M}\rangle$.

Design by functional abstraction makes heuristic simplifications at least at two places: The original demand specification, D, is simplified towards a functional specification $\hat{D}$ (Step (1) in the figure), and, $S\langle F, \mathcal{M}\rangle$ is transformed locally into $B'\langle F', \mathcal{M}'\rangle$ (Step (3) in the figure). Both, the synthesis step and the adaptation step may be operationalized with complete algorithms (Step (2) and (4) in the figure).

The solutions in the following case studies were developed after this paradigm.

[1] See Section 1.4, Page 22, for a discussion and a graphical illustration of this method.

A.1 Case-Based Design in Fluidics

Fluidic drives are used to realize a variety of production and manipulation tasks. Even for an experienced engineer, the design of a fluidic system is a complex and time-consuming task, that, at the moment, cannot be automated completely. Designing a system means to transform demands, $\mathcal{D}$, towards an explicit system description, which is a behavior model $B\langle F, \mathcal{M}\rangle$ of the desired system in most cases:

$$\mathcal{D} \longrightarrow B\langle F, \mathcal{M}\rangle \tag{A.1}$$

Taken the view of configuration, the designer of a fluidic system selects, parameterizes, and connects components like pumps, valves, and cylinders such that $\mathcal{D}$ is fulfilled by the emerging circuit.[2] Solving a fluidic design problem at the component level is pretty hopeless, and we will apply model simplification to reach tractability. Note that model simplification must not be bound up with a coarsening of the behavior model or a reduction of the number of components involved: As pointed out in (257, pg. 27), the reduction of an unconstrained synthesis space towards an (even large) space of variants can transform an innovative design problem towards a tractable configuration problem. This strategy is applied successfully here; the rationale was already given in (258), under the label "functional decomposition" and "functional composition" respectively.

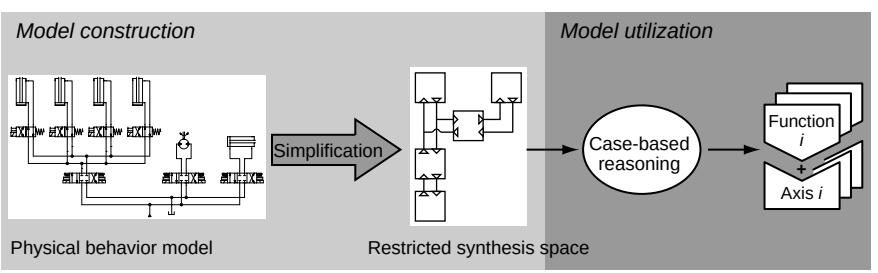


Figure A.2. Model simplification is realized by restricting the space of possible structure models towards a regular subset. Based on the simplified model, the design problem can be tackled with methods from the field of case-based reasoning.

The principle claims to perform a configuration process at the level of functions (instead of components), which in turn requires that fluidic functions possess constructional equivalents that can be treated in a building-block-manner. This requirement is fairly good fulfilled in the fluidic domain, and the respective building blocks are called "fluidic axes".

[2] The ideas presented in section have been verified in the hydraulic domain in first place; however, they can be applied in the pneumatic domain in a similar way, suggesting us to use preferably the more generic word "fluidic".

Definition A.1 (Fluidic Axis) *A fluidic axis both represents and fulfills a function D of an entire fluidic plant. The axis defines the connections and the interplay among the working, control, and supply elements that realize D.*[3]

The overall design approach developed in this section follows the paradigm depicted in Figure A.1, Page 89: (1) The original demand specification, $\mathcal{D}$, is abstracted towards a functional specification $\hat{\mathcal{D}}$. (2) At this functional level a structure model $S\langle F, \mathcal{M}\rangle$ according to the coupling of the fluidic functions in $\hat{\mathcal{D}}$ is generated. (3) $S\langle F, \mathcal{M}\rangle$ is completed towards a tentative behavior model $B'\langle F', \mathcal{M}'\rangle$ by plugging together locally optimized fluidic axes; here, this step is realized by a case-based reasoning approach (see Figure A.2). (4) The tentative behavior model $B'\langle F', \mathcal{M}'\rangle$ is repaired, adapted, and optimized globally.

Remarks. A human designer is capable of working at the component level, *implicitly* creating and combining fluidic axes towards an entire system. His ability to automatically derive function from structure—and vice versa: structure *for* function—allows him to construct a fluidic system without the idea of high-level building blocks in the form of fluidic axes.

Underlying Models

A fluidic system can be described at different levels (layers) of abstraction. From the standpoint of design problem solving, the following layers are important: The demand layer, which specifies a complex set of demands $\mathcal{D}$, the functional layer, which specifies the model of fluidic function $\hat{B}\langle\hat{F}, \hat{\mathcal{M}}\rangle$, and the component layer, which specifies the electro-fluidic system model $B\langle F, \mathcal{M}\rangle$.

The demand layer contains the entire specification for the desired behavior of a fluidic system. Vier et al. discuss possible demands in detail, such as tolerance constraints, operating restrictions, boundary values, etc. (291). Central elements of $\mathcal{D}$, however, are the cylinder profiles, which prescribe the courses of the forces, the velocities, or the pressure. Implicitly, these profiles characterize particular phases of the working process, such as a speed phase, a slowing down phase, or a press phase. Figure A.3 shows cylinder profiles for a hydraulic system that operates in the low pressure range and that contains two axes, which perform a combined manipulation and pressing task.

A model of fluidic function, $\hat{B}\langle\hat{F}, \hat{\mathcal{M}}\rangle$, is a discrete event model. The gist of $\hat{B}\langle\hat{F}, \hat{\mathcal{M}}\rangle$ is a set of state variables, F_X, along with the discrete state prescription function, Δ. Each state variable in F_X represents the function of a fluidic axis in $\hat{\mathcal{M}}$; Δ characterizes the behavior of the fluidic axes by means of the working phases of the output units, which are cylinders in most cases. A formal specification of $\hat{B}\langle\hat{F}, \hat{\mathcal{M}}\rangle$ is

[3] The same definition is also given in Section D.3, which deals with the functional analysis of fluidic systems.

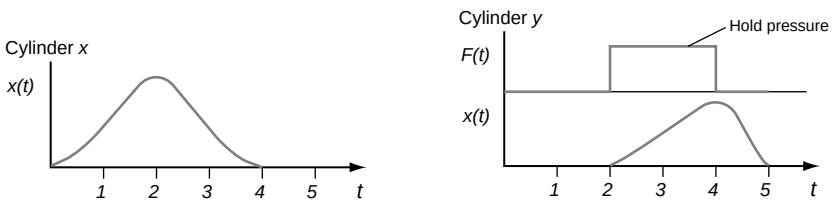


Figure A.3. Demanded cylinder profiles for a hydraulic system.

given in Case Study C.2 on Page 182, where a model of fluidic function is reformulated into a similarity measure. In this place we confine ourselves to an illustration of $\hat{B}\langle\hat{F},\hat{\mathcal{M}}\rangle$ at the example.

	Phase	Phase Type	Value
Axis x	f_1	*const-drive-out*	$0.5\,m/s$
	f_2	*const-drive-in*	$-0.8\,m/s$
Axis y	f_1	*hold-pressure*	
	f_2	*fast-drive*	$-2.0\,m/s$

Global phase schedule

Axis x	f_1	f_2	
Axis y		f_1	f_2
Time	$0s$	$2s$	$4s$

Table A.1. Functional layer that corresponds to the demand layer of Figure A.3; four phases have been identified and scheduled.

Table A.1 shows the functional layer that corresponds to the demand layer of Figure A.3. Here, four phases have been identified and scheduled. The respective fluidic axes must be coupled sequentially to realize $\mathcal{D}$.

An electro-fluidic system model, $B\langle F,\mathcal{M}\rangle$, is a combined discrete event/continuous time model. The elements in $\mathcal{M}$ designate hydraulic or pneumatic components and, for control purposes, electrical components; the elements in F designate fluidic and electrical quantities. $B\langle F,\mathcal{M}\rangle$ is based on local behavior constraints for the components in $\mathcal{M}$, represented by algebraic and differential equations. Typically, $B\langle F,\mathcal{M}\rangle$ contains feedback loops, and the simulation of Δ requires global computation methods. A formal specification of $B\langle F,\mathcal{M}\rangle$ is given in Case Study B.2 on Page 139, where an electro-fluidic system model is compiled into a heuristic rule model. In this place we confine ourselves to an illustration of $B\langle F,\mathcal{M}\rangle$ in the form of the circuit diagram in Figure A.4.

The shown circuit realizes the functional description above; it thus represents a solution of the design task outlined in Figure A.3.

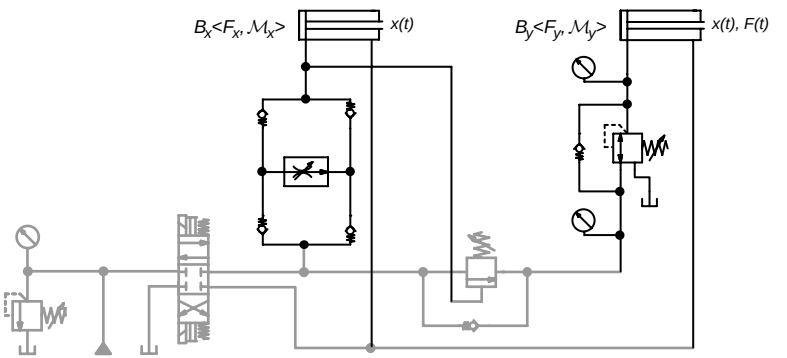


Figure A.4. Circuit diagram with two axes $B_x\langle F_x, \mathcal{M}_x\rangle$ and $B_y\langle F_y, \mathcal{M}_y\rangle$. Putting together the components' local behavior descriptions according to the diagram yields the behavior model $B\langle F, \mathcal{M}\rangle$.

Model Simplification

The design task in question contains creative parts: The structure of a fluidic system is not predefined and has to be created. Moreover, demanding model formulation and inverse simulations must be performed to compute unknown parameters as well as to check the system with regard to the desired demands.

By model simplification the design task can be made tractable—but, in contrast to the far-reaching restrictions applied in Case Study A.2, model simplification here concerns only size and structure of the synthesis space. The behavior fidelity, F, the behavior granularity, $\mathcal{M}$, and the state prescription function, Δ, retain their complexity.

The model simplification idea goes back on the observation that human engineers tackle design tasks not at the component level, but at the level of function. Equation A.1 then reads as follows:

$$\mathcal{D} \longrightarrow \hat{\mathcal{D}} \longrightarrow B\langle F, \mathcal{M}\rangle$$

where $\hat{\mathcal{D}}$ is a functional description of the desired system, corresponding to the model of fluidic function, $\hat{B}\langle \hat{F}, \hat{\mathcal{M}}\rangle$. What makes this observation so interesting here, and not, e. g., for Case Study A.2, is the fact that the mapping $\hat{\mathcal{D}} \longrightarrow B\langle F, \mathcal{M}\rangle$ can be decomposed into rather independent subproblems. This principle, which we call "functional composition" (258), says that

(1) each specification of demands, $\mathcal{D}$, can be translated into a set of fluidic functions, $\hat{\mathcal{D}} = \{D_1, \ldots, D_k\}$,

(2) each function $D \in \hat{\mathcal{D}}$ can be mapped one to one onto a fluidic axis $B_D\langle F_D, \mathcal{M}_D\rangle$ that operationalizes D,

(3) $\mathcal{D}$ can be realized by coupling the respective axes for the functions in $\hat{\mathcal{D}}$, whereas the necessary coupling information can be derived from $\mathcal{D}$, and

(4) the coupling structure is formed by a recursive application of couplings from the type set {*series, parallel, sequential*} (explained below).

While the first point goes in accordance with reality, the Points (2) and (3) imply that a function D is neither realized by a combination of several axes nor by constructional side effects. This assumption, as well as the assumption made in Point (4) represent noticeable simplifications.

Anyway, under this working hypothesis, $\mathcal{D}$ can be transformed towards an electro-fluidic system model, $B\langle F, \mathcal{M}\rangle$, within two design steps: by separately designing a fluidic axis for each fluidic function D in $\hat{\mathcal{D}}$, and by coupling these axes in a qualified way. Figure A.5 depicts this view.

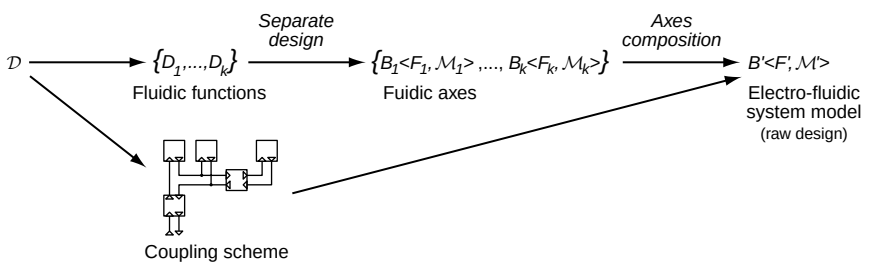
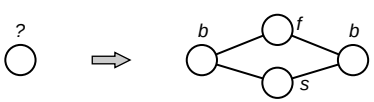


Figure A.5. Automating fluidic circuit design by functional composition.

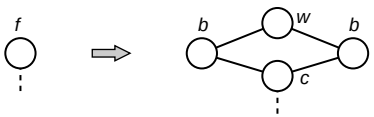
The scope of the structural simplifications that are implied by the principle of functional composition is reflected by the (small) size of the design graph grammar, $\mathcal{G} = \langle \Sigma, P, ? \rangle$, that defines the restricted synthesis space $L(\mathcal{G})$. $\Sigma = \{b, t, f, s, c, w, A, B, C, D\}$ where the nonterminals designate a biconnection, triconnection, fluidic function, supply element, control element, and working element. P contains seven rules of the form $T \rightarrow \langle R, I \rangle$ presented below; apart from rule (1b) only the graphical form is shown.

(1a) Initial rule.

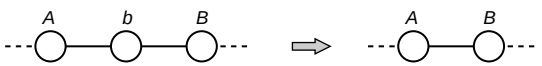


(1b) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1\}, \{\}, \{(1, f)\} \rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{2,3,4,5\}, \{\{2,3\}, \{2,5\}, \{3,4\}, \{4,5\}\},$
$\{(2,b), (3,w), (4,b), (5,c)\} \rangle$
$I = \{((A,f),(A,c))\}$

Function-to-axis conversion.

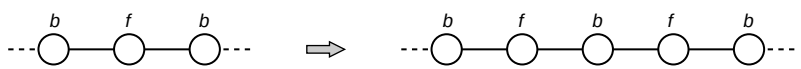


(1c) Biconnection deletion.

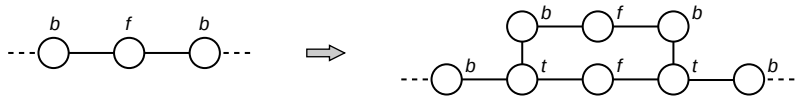


The rules (2a)–(2c) encode the introduction of a series coupling, a parallel coupling, and a sequential coupling respectively.

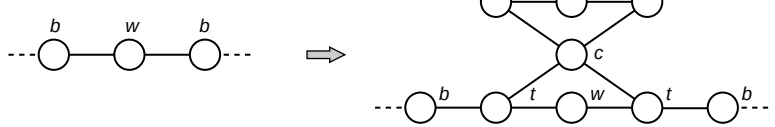
(2a)



(2b)



(2c)



The following display rules change the appearance of the labeled graph into a building block structure.

(3) Display rules.

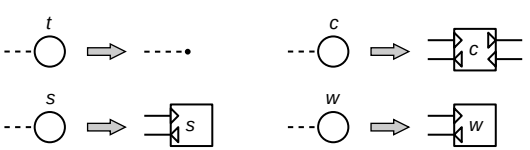


Figure A.6 shows the circuit from the example and its building block structure. The building block structure corresponds to the derivation (1a) $\longrightarrow$ (1b) $\longrightarrow$ (2c) $\longrightarrow$ (1c), followed by an application of the matching display rules.

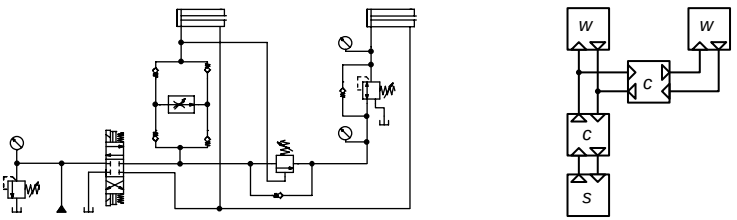


Figure A.6. Circuit diagram and related building block representation.

Remarks. The SCHEMEBUILDER system developed by Oh et al. also pursues a design approach that is based on model simplification (204, 48): A demand set $\mathcal{D}$ is interactively decomposed into subtasks, assuming that each of the subtasks is realized by means of a single hydraulic axis. The axes in turn are retrieved from a database containing carefully selected design prototypes. However, Oh et al.'s approach lacks especially in two respects: (1) Axes can only be connected in parallel at a single nesting depth, which restricts the possible designs to a uniform circuit type. (2) An analysis of a composed circuit in the form of a simulation is not integrated.

Working with Simplified Models: Case-Based Design

Taking the simplifying view of the design process as shown in Figure A.5, the step $\{D_1, \ldots, D_k\} \longrightarrow \{B_1\langle F_1, \mathcal{M}_1\rangle, \ldots, B_k\langle F_k, \mathcal{M}_k\rangle\}$ can be realized by case-based reasoning (short: CBR) methods—provided that the following can be developed: A similarity measure for fluidic functions and an adaptation concept for fluidic axes.

The following paragraphs introduce the case-based design approach in greater detail. We start with a short introduction to design problem solving and CBR; the next but one paragraph develops a similarity measure for fluidic functions $D \in \hat{\mathcal{D}}$. This measure is vital to realize the retrieve step in the CBR approach: For a given function D it identifies the most similar fluidic axis $B_D\langle F_D, \mathcal{M}_D\rangle$ from a collection $\mathcal{C}$ of axes. The two paragraphs thereafter are devoted to revision; it is shown in which way misfitting axes and even entire circuits can be adapted. The last paragraph presents some evaluation results.

Note that the coupling of the selected axes, i. e., the step $\{B_1\langle F_1, \mathcal{M}_1\rangle, \ldots, B_k\langle F_k, \mathcal{M}_k\rangle\} \longrightarrow B'\langle F', \mathcal{M}'\rangle$, does no longer constitute a design (or search) problem, since we strictly follow the coupling scheme defined by the design graph grammar $\mathcal{G}$.

Case-Based Reasoning and Design Let a case combine a description of a problem along with a solution. Basic idea of case-based reasoning (CBR) is to exploit previously solved cases when solving a new problem. I. e., a collection of cases is browsed for the most similar case, which then is adapted to the new situation. The commonly accepted CBR cycle shown in Figure A.7 goes back to Aamodt and Plaza and is comprised of four steps (1).

(1) *Retrieve.* A case relevant for the problem is retrieved.

(2) *Reuse.* Having performed more or less adaptations, the retrieved case may be reused.

(3) *Revise.* Having evaluated the adapted case, additional repair adaptations may be applied.

(4) *Retain.* The new case, consisting of the problem along with a solution, is stored.

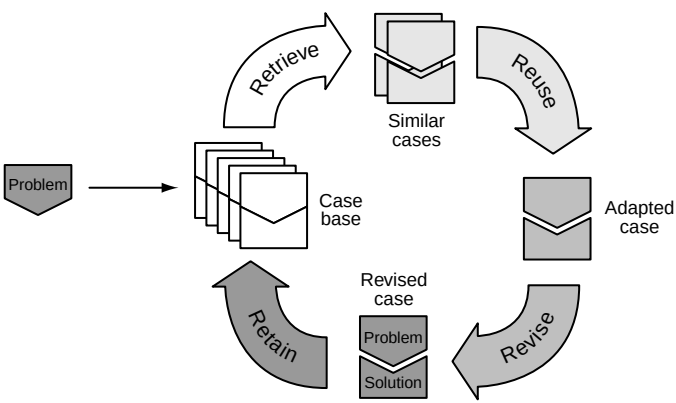


Figure A.7. The classical CBR cycle, as it can be found in (1), for instance.

Configuration, design, synthesis—these terms stand for problem classes where the AI paradigm "generate and test" has been applied rather successfully (34, 46, 170). CBR, however, follows the paradigm "retrieve and adapt" (156). Both concepts can work fine together to solve design problems.

A previously solved design problem that contributes a good deal to the desired solution may bound difficult synthesis and adaptation tasks to a tractable rest problem. Following this idea, the starting position of a design problem should be created with CBR methods, while for the heuristic and search-intensive adaptation tasks other AI paradigms come into play.

For the time being, our focus is on design problems stated in the form of a desired fluidic function D; a solution of such a problem is every electro-fluidic system model $B\langle F, \mathcal{M}\rangle$ of a fluidic axis whose behavior complies with D.

Remarks. There exist two concepts of how a problem's solution can be defined: One of them codes the problem solving *process*, the other codes the *result* of a problem solving process, for example in the form of a system description $B\langle F, \mathcal{M}\rangle$. From this distinction result two analogy concepts in CBR, namely that of derivational analogy (belonging to the former) and that of transformational analogy (belonging to the latter) (36, 95, 109). For reasons of clearness, the considerations of this paper are oriented

at the latter, i. e., at the system description view, but they could be reformulated to the process-centered view as well.

Definition A.2 (Case, Case Base, Query) *Let $\mathcal{D}$ be a set of design problems, and let $\mathcal{S}$ be a set of design solutions. A case C is a tuple $C = \langle D, S \rangle$, $D \in \mathcal{D}$, $S \in \mathcal{S}$, where S constitutes a solution for D. A set $\mathcal{C}$ consisting of cases is called a case base. A case of the form $Q = \langle D, \cdot \rangle$ is called query or problem definition to a case base.*

When given a query $Q = \langle D, \cdot \rangle$ to a case base $\mathcal{C}$, two jobs must be done to obtain a solution to Q: The retrieval of a similar case C, and the adaptation of C such that D is fulfilled.

Wess mentions three approaches to define similarity (298): Similarity based on predicates, similarity based on a preference relation, and the most generic concept, similarity based on a measure. In connection with design problem solving, only the last is powerful enough, and the following definition will formalize a similarity measure for design case bases.

Definition A.3 (Case Similarity, Similarity Measure) *Given is a symmetric function $sim_D : \mathcal{D} \times \mathcal{D} \rightarrow [0;1]$, which additionally has the reflexivity property, $sim_D(D_x, D_y) = 1 \Leftrightarrow D_x = D_y$. Moreover, let $C_x = \langle D_x, S_x \rangle$ and $C_y = \langle D_y, S_y \rangle$, C_x, $C_y \in \mathcal{C}$, be two cases. Then the case similarity $sim : \mathcal{C} \times \mathcal{C} \rightarrow [0;1]$ is defined by means of sim_D in the following way: $sim(C_x, C_y) = sim_D(D_x, D_y)$.*

Remarks. (1) The semantics of sim_D shall be as follows. The more similar two fluidic functions D_x and D_y are, the larger shall be their value $sim_D(D_x, D_y)$. (2) The symmetry property guarantees that $sim(C_x, C_y) = sim(C_y, C_x)$; the reflexivity property defines the self-similarity of a case.

A Similarity Measure for Fluidic Functions Note that a similarity measure for fluidic engineering must anticipate the constructional effort that is necessary to transform a fluidic *system*—and not its functional abstraction: Given two fluidic systems, S_x, S_y, and two related functional descriptions in the form of the cases C_x, C_y, then $sim(C_x, C_y)$ must quantify the effort to reconstruct S_x into S_y, and vice versa (see Section 3.4, Page 81, for a concept to measure transformational efforts in design tasks).

Supposed there is a case base, $\mathcal{C}$, with cases of the form $\langle D, B\langle F, \mathcal{M} \rangle \rangle$, and a query, $\langle D, \cdot \rangle$, for which a suited fluidic axis shall be retrieved from $\mathcal{C}$. Then a mapping, sim_D, with the following properties is required.

(1) sim_D is defined on the domain $\mathcal{D} \times \mathcal{D}$, where $\mathcal{D}$ is a set that comprises the possible fluidic functions.

(2) sim_D is a similarity measure as defined in Definition A.3.

(3) sim_D is semantically reasonable, or formally:

$$\text{“}sim_D(D, D_y) = \max\left\{sim_D(D, D_x) \mid \langle D_x, B\langle F, \mathcal{M}\rangle\rangle \in \mathcal{C}\right\}\text{”}$$

$$\Leftrightarrow$$

$$\text{“}\langle D_y, B_y\langle F_y, \mathcal{M}_y\rangle\rangle \in \mathcal{C} \text{ is the best case in } \mathcal{C} \text{ to realize } D_x\text{”}$$

Remarks. The similarity measure sim_D estimates two fluidic functions respecting their similarity. It maps from the Cartesian product of the domain of fluidic functions onto the interval $[0;1]$. The last property, (3), postulates that sim_D should become maximum only for those cases in $\mathcal{C}$ that realize the demanded function D best.

The elementary characteristic of a fluidic function D is defined by both the sequence and the type of its phases. Valuating two fluidic functions respecting their similarity thus means to compare their phases. However, the phases in turn are characterized by their type, duration, distance, force, or precision, and each of which can be quantified by a special phase similarity measure. Table A.2 gives a quantification for sim_{phase}, the similarity of two phases' types, which is the most important phase characteristic and which is used in the following definition.

Phase type	*Position*	*Constant*	*Accelerate*	*Hold-Pos*	*Hold-Press*	*Press*	*Fast*
Position	1	0.3	0.5	0	0	0.3	0.7
Constant	0.3	1	0	0	0	0.7	0.7
Accelerate	0.5	0	1	0	0	0.2	0.4
Hold-Pos	0	0	0	1	0.6	0.3	0
Hold-Press	0	0	0	0.6	1	0.8	0
Press	0.3	0.7	0.2	0.3	0.8	1	0
Fast	0.7	0.7	0.4	0.0	0	0.0	1

Table A.2. The similarity between two phase types, sim_{phase}.

Definition A.4 (Similarity Measure for Fluidic Functions) *Let the fluidic function D_x designate the phase sequence $(f_1^{(x)}, \ldots, f_p^{(x)}) \in \mathcal{D}$, and let the fluidic function D_y designate the phase sequence $(f_1^{(y)}, \ldots, f_q^{(y)}) \in \mathcal{D}$ for some $p, q \in \mathbf{N}$. A phase-type-based similarity measure for fluidic functions, $sim_D : \mathcal{D}\times\mathcal{D} \to [0;1]$ is defined as follows.*

$$sim_D(D_x, D_y) = \frac{1}{\max\{p, q\}} \sum_{i=1}^{\min\{p,q\}} sim_{phase}(f_i^{(x)}, f_i^{(y)})$$

Remarks. (1) Obviously does sim_D fulfill the conditions of a similarity measure. (2) In the case $p \neq q$ not all phases will match and hence not all phases are considered in the computation of sim_D. A fact, which does reflect our comprehension of similarity in most cases—it does not if, for instance, D_x and D_y are described at different levels of detail. This and other shortcomings have been addressed in the work of (110): He

proposed and realized an algorithm that enables a smart matching between phase sequences varying in length.

In Section C.2 it is shown how a similarity measure for fluidic axes, sim_{axes}, can be constructed automatically by solving a regression problem on a database of similarity samples. Clearly, there is a tradeoff in knowledge acquisition effort and classification quality: While the function sim_{axes} from Section C.2 probably leads to worse similarity estimations, the construction of sim_D required substantial acquisition and implementation time.

Adaptation of Fluidic Axes By means of the above similarity measure for fluidic functions, sim_D, the k most similar cases can be retrieved from a case base of fluidic axes given a query $\langle D, \cdot \rangle$. Note, however, that these cases merely form solution candidates; usually, a retrieved case must be adapted to fulfill the demanded fluidic function D. Case adaptation plays a key role in case-based design (148) and is performed within the reuse step (bring to mind Figure A.7 again).

The following definition specifies the terms "modification" and "adaptation". While each adaptation represents a modification, the inverse proposition does not hold: A modification of some case establishes an adaptation only if the modified object of the case—in our setting a modified fluidic axis $B'\langle F', \mathcal{M}' \rangle$—fulfills the demanded function D to a higher degree than does the unmodified axis $B\langle F, \mathcal{M} \rangle$ of the original case.

Definition A.5 (Modification, Adaptation (264)) *Let $C_x = \langle D_x, B_x\langle F_x, \mathcal{M}_x \rangle \rangle \in \mathcal{C}$ be a case, and let $Q = \langle D, \cdot \rangle$ be a query. A modification of C_x respecting Q is a function $\mu : \mathcal{D}\times\mathcal{C} \to \mathcal{D}\times\mathcal{B}$, with $\mu(D, C_x) = \langle D_y, B_y\langle F_y, \mathcal{M}_y \rangle \rangle$. $\mathcal{B}$ designates the space of behavior models for fluidic axes.*

A modification of C_x is called adaptation of C_x if the following condition holds:[4]

$$sim\big(\langle D_y, B_y\langle F_y, \mathcal{M}_y \rangle \rangle, Q\big) > sim\big(\langle D_x, B_x\langle F_x, \mathcal{M}_x \rangle \rangle, Q\big)$$

Case adaptation can be realized in different ways. A popular approach is the formulation of adaptation knowledge in the form of (heuristic) modification rules (19, 106, 270). In technical domains where the behavior of the system to be adapted is well understood, a particular type of modification rules, called "scaling" rules here, can be employed to encode modification knowledge.

Definition A.6 (Scale Function, Scalable, Scaling (264)) *Given is a query $Q = \langle D, \cdot \rangle$, a subset of the demanded fluidic function $D' \subseteq D$, and a case $C_x = \langle D_x, B_x\langle F_x, \mathcal{M}_x \rangle \rangle \in \mathcal{C}$. A function $\nu : \mathcal{D}\times\mathcal{C} \to \mathcal{D}\times\mathcal{B}$ is called scale function of C_x respecting D' if the following conditions hold.*

(1) $C_y = \nu(D', C_x) = \langle D_y, B_y\langle F, \mathcal{M} \rangle \rangle$ *with* $D' \subseteq D_y$

[4] The condition is equivalent to the following: $sim_D(D_y, D) > sim_D(D_x, D)$.

(2) $sim(C_y, Q) > sim(C_x, Q)$

C_x is called scalable with respect to D, C_y is called scaling of C.

Remarks. (1) In other words, with respect to a part of the desired function, $D' \subseteq D$, there is a case $C_x = \langle D_x, B_x\langle F_x, \mathcal{M}_x\rangle\rangle$ in the case base whose fluidic axis $B_x\langle F_x, \mathcal{M}_x\rangle$ can be modified—say: scaled—towards $B_y\langle F_y, \mathcal{M}_y\rangle$ in such a way that $B_y\langle F_y, \mathcal{M}_y\rangle$ provides D' and C_y is more similar to Q than is C_x. (2) Obviously does each scaling establish an adaptation.

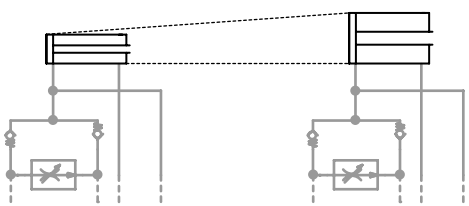


Figure A.8. Scaling a cylinder respecting a desired force.

Example. Consider the design of a lifting hoist where $C_x = \langle D_x, B_x\langle F_x, \mathcal{M}_x\rangle\rangle$, the most similar case found respecting the query $Q = \langle D, \cdot\rangle$, does not fulfill the maximum force constraint $(F_{max}, w) \in D$. Given this situation, C_x can be scaled up to fulfill D if the force difference between the existing and the desired system is of the same order of magnitude (see Figure A.8).

In this simple example the scaling of the force is possible since the responsible underlying physical connections, the balance of forces, can be quantified. A reasonable scale function could utilize this law as follows. It adapts the force value v of C_x according to the demanded value w by scaling the piston area, A_{cyl}, to a new value with respect to the maximum pressure allowed, p_{max}, i. e., $A_{cyl} = w/p_{max}$. The resulting state prescription function is designated with Δ_y.

Formally, the scale function takes two arguments (recall Definition A.6); the first of which defines the subset of D to be achieved by scaling, the second is the case to be modified:

- $C_y = \nu\big(\{(F_{cyl}, w)\}, C_x\big) = \langle D_y, B_y\langle F_y, \mathcal{M}_y\rangle\rangle \in \mathcal{D}\times\mathcal{B}$

where $D_y = D \setminus \{(F_{cyl}, v)\} \cup \{(F_{cyl}, w)\}$, and $B_y\langle F_y, \mathcal{M}_y\rangle = B_x\langle F_x, \mathcal{M}_x\rangle \setminus \{\Delta_x\} \cup \{\Delta_y\}$.

Note that condition (2) of Definition A.6 is fulfilled: The similarity between the scaled case C_y and the query Q is strictly larger than the similarity between the original case C_x and Q.

A Design Language: Adaptation of Entire Circuits The composition of the adapted fluidic axes follows the coupling scheme from Page 95 and yields a preliminary design solution, $B'\langle F', \mathcal{M}'\rangle$, a so-called "raw design". There is a good chance that $B'\langle F', \mathcal{M}'\rangle$ incorporates the potential to fulfill the specification $\mathcal{D}$, say, that a sequence of modification steps can be found that transforms $B'\langle F', \mathcal{M}'\rangle$ into a behavior $B\langle F, \mathcal{M}\rangle$ which complies with $\mathcal{D}$. An example for such a modification is the following piece of design knowledge.

"An insufficient damping can be improved by installing a by-pass throttle."

This measure encodes a lot of implicit engineering know-how, among others: (1) A by-pass throttle is connected in parallel, (2) the component to which it is connected is a cylinder, (3) if there are several cylinders in the system, an engineer knows the best-suited one, (4) a by-pass throttle is a valve. Figure A.9 illustrates the repair rule.

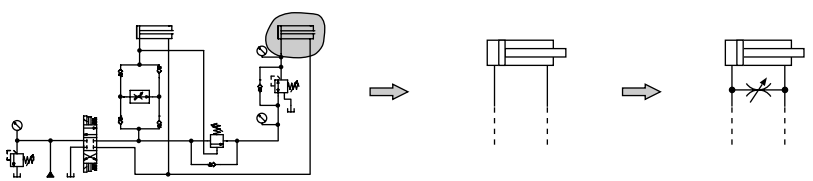


Figure A.9. Illustration of the repair rule *"An insufficient damping can be improved by installing a by-pass throttle."*: Interpret context (cylinder), choose the best context amongst several candidates (left cylinder), apply repair action to context (connect throttle in parallel).

What is more, engineers use design knowledge in a flexible way; i. e., a particular piece of knowledge can be applied to different contexts in a variety of circuits. Flexibility is a major reason which makes it difficult to encode the expressiveness of the above example on a computer. Suppose we were confronted only with systems of the same topological set-up, then measures like the above (*"Install a by-pass throttle."*) could simply become hard-wired within a would-be design algorithm.

To formulate sophisticated modification knowledge as in the example above, a more powerful concept than scaling rules is sought. One possibility getting the knack of the outlined problems is to specify implicit knowledge explicitly. For these purposes, we have been experimenting with a prototypic scripting language tailored to fluidic circuit design.[5] This language was a precursor of the design graph grammar concept, which is introduced in Chapter 3 of this thesis. In this place we will not delve into the design language's syntax and semantics but shortly outline its concepts; details can be found in (238, 270).

- *Action Types.* Figure A.10 shows the basic action types that are available in the design language.

[5]The research was part of the OFT-project, which was supported by DFG grants KL 529/7-1,2,3 and SCHW 120/56-1,2,3.

Figure A.10. All kinds of modifications must be put down to the five basic action types shown in the figure.

- *Separation of Location and Action.* Each action can be equipped with location information that specifies where the action shall take place. This way, knowledge of location and knowledge of action are strictly distinguished—a separation that resembles the distinction of structure and behavior. The semantics of a location specifier corresponds to a matching with context as it is defined in Section 3.1 on Page 61.

- *Modification Schemes.* Actions can be organized within so-called modification schemes, which provide simple macro facilities with local variables, loops, and branching.

- *Meta Knowledge.* To control the application of different adaptation measures, we characterize them with values from $[0; 1]$ indicating the effectiveness, the undesired side effects, called repercussion, and the realization cost, (290, 289). From these values an overall confidence κ can be derived that is conform to application requirements and the designer's preferences κ_{eff}, κ_{rep}, and κ_{cost}. Table A.3 shows an expert's evaluation of measurements to increase a cylinder's damping factor.

  Here, installing a throttle in by-pass to the cylinder (see Figure A.9) is ranked first option; the resulting drain flow through the by-pass moves the eigenvalues of the related transfer function to a higher damping. An extensive theory on the analysis, the assessment, and the adaptation of hydrostatic drives is given in (288).

Modification Measure	Effectiveness	Repercussion	Cost	κ
Throttle in mainstream	0.1	0.4	0.8	0.39
Throttle in side stream	0.4	0.4	0.5	0.44
Throttle in by-pass	0.8	0.4	0.5	0.64
Damping network	0.9	0.8	0.1	0.61

Table A.3. Possible remedies to increase a cylinder's damping factor that has been judged being too low. The computation of κ relies on the preferences $\kappa_{eff} = 0.5$, $\kappa_{rep} = 0.15$, and $\kappa_{cost} = 0.35$, which have been proposed by an domain expert.

Observe that the scaling function ν in Definition A.6 (Page 101) gets by without an extra evaluation step. I. e., the effects of an adaptation can completely be foreseen, or,

at least, they can be estimated within narrow bounds. This cannot be guaranteed for more complex adaptations, such as exemplified in Figure A.9 or as they can specified within our design language. In practice, each modification step must be followed by an evaluation step in order to assess the actual state of design quality.[6] However, evaluability is no matter of course, which leads to the following definition.

Definition A.7 (Evaluable (264)) *Let $\mathcal{D}$ be a set of design problems, and let $\mathcal{S}$ be a set of design solutions. A case $C = \langle D, S \rangle$, $D \in \mathcal{D}$, $S \in \mathcal{S}$, is called evaluable if a function $\varepsilon : \mathcal{S} \rightarrow \mathcal{D}$ with $\varepsilon(S) = D$ can be stated.*

Evaluability prepares the ground for complex adaptations. Using CBR terminology, these adaptations are called "repair" or—along with a preceding evaluation—"revise". Obviously, adaptation plus evaluation can be performed several times, leading to a cycle that renders the design cycle presented at the outset in Figure A.7. Here CBR forms a frame where an approach for design problem solving is embedded.

Remarks. Automatic evaluation often turns out to be a complex job involving demanding reasoning and simulation tasks (94, 259).

Experimental Results Most of the concepts of this subsection have been operationalized. In this regard, Hoffmann has implemented a hydraulic design assistant (110), which is linked to ARTDECO, our drawing and simulation environment for fluidic systems (257, 262). The design assistant enables a user to formulate his demands of the desired circuit as a functional specification $\hat{D}$. For each $D \in \hat{D}$ a sequence of phases can be defined, where for each phase a set of characteristic parameters, such as duration, precision, or maximum values can be stated.

Given some $\hat{D}$, the retrieval mechanism of the assistant searches the case base for axes fitting best the specified functions, according to the measure sim_D. Afterwards, these building blocks are scaled and composed towards a new system, and, finally, a drawing is generated which can directly be simulated and evaluated. Figure A.11 shows a query (top left), the functional description of the generated design (below), and the related drawing.

Clearly, a direct evaluation of generated design solutions must be limited within several respects since

(1) an absolute measure that captures the quality of a design does not exist, and

(2) the number of properties that characterizes a design is large and their quantification often requires a high effort.[7]

[6] Adaptations whose effect cannot be predicted at a sufficient accuracy are called "Level 2 adaptations" in (264).

[7] Characterizing properties include: number of components, reliability, cost, effort for setting into operation, effort for maintenance, degree of standardization, fault-proneness, diagnosability.

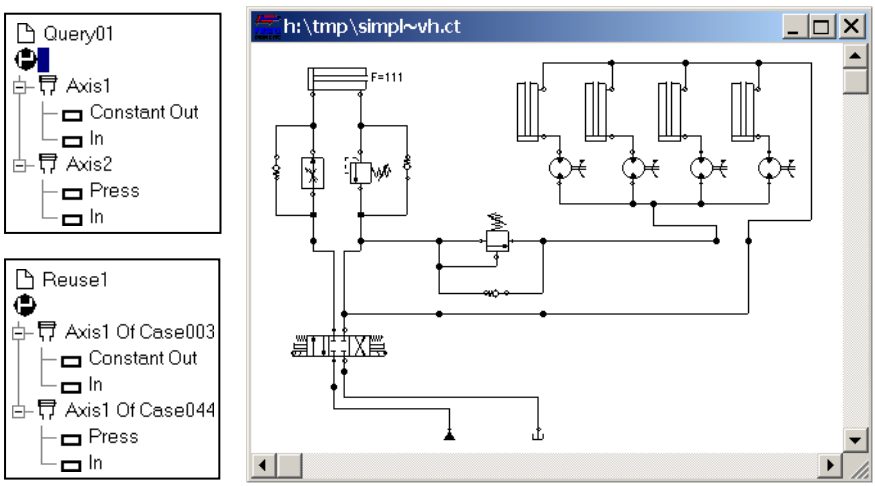


Figure A.11. A design query (top left), the functional description of a solution (below left), and the automatically generated drawing of the design solution.

Anyway, the quality of a generated design can also be rated indirectly, by quantifying its distance to a design solution created by a human expert. The term "distance" is used with the semantics of our theory in Section 3.4, Page 81, and stands for the necessary modification effort to transform the machine solution into the human solution. The experimental results presented in Table A.4 describe such a competition. The underlying experiments have been performed by Hoffmann (110); a more detailed discussion of evaluation concepts can be found at the same place.

$\lvert\mathcal{D}\rvert$	Retrieve	Reuse	$\overline{sim_D} \geq 0,8$	$\overline{sim_D} \geq 0,9$	O.K.	Expert modification
1	$\ll$ 1s	0.10s	17	13	10	10x(+), 6x(o), 1x(–)
2	$\ll$ 1s	0.63s	16	11	9	8x(+), 7x(o), 1x(–)
3	$\ll$ 1s	0.91s	17	10	7	7x(+), 8x(o), 2x(–)
4	$\ll$ 1s	1.43s	15	8	5	3x(+), 10x(o), 2x(–)
5	$\ll$ 1s	2.00s	18	6	1	1x(+), 15x(o), 2x(–)

Table A.4. Runtime results and modification effort for automatically generated designs (see an explanation of the entries below).

Description of the table columns:

- $\lvert\mathcal{D}\rvert$. Number of axes of the queries in the test sets; a test set contains 20 queries.
- *Retrieve.* Average time of a retrieve step in seconds.
- *Reuse.* Average time of a reuse step in seconds.

- $\overline{sim_D} \geq 0.8$. Number of generated designs where $\overline{sim_D}$, the averaged similarity sim_D over all axes, is not smaller than 0.8.
- $\overline{sim_D} \geq 0.9$. Number of generated designs where $\overline{sim_D}$, the averaged similarity sim_D over all axes, is not smaller than 0.9.
- *Simulation O.K.* Number of generated designs whose simulation results fulfill the demands of the query.
- *Expert Modification.* Evaluation of a human expert. The symbols (+), (o), and (–) designate a small, an acceptable, and a large modification effort to transform the machine solution into a solution accepted by the human expert.

  Note that this evaluation follows the paradigm of structure model distance, which is introduced and formalized in Section 3.4, Page 81.

Test environment was a Pentium II system at 450 MHz with 128 MB main memory; the operating system was Microsoft Windows NT 4.0.

Synopsis

Problemclass Design of technical systems.

Problem Solving Method Design by functional composition: Synthesis of functional building blocks that have been retrieved and adapted by CBR methods.

Source Model Discrete event/continuous time model $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$, defined over $\langle F, \mathcal{M}\rangle$.

- *Fidelity Level F.* Fluidic, mechanical, and electrical quantities.
- *Granularity Level $\mathcal{M}$.* Fluidic and electrical components.
- *Input F_U.* Driving profiles for the cylinder forces.
- *State Prescription Function Δ.* Explicit local differential equations; implicit nonlinear algebraic equations.
- *Output Function Λ.* Courses of selected physical quantities.
- *Behavior Model Processing.* Differential-algebraic system solving.

Simplified Model Collection of single fluidic axes. Each fluidic axis is a discrete event/continuous time model $B\langle F, \mathcal{M}\rangle$ similar to the source model. The simplification results from the isolated processing and simulation of the axes, neglecting feedback, constructional side effects, and functionality that results from structure.

Knowledge Source Principle of "functional composition" copied from human designers (258).

A.2 Structure Synthesis in Chemical Engineering

This section introduces an approach to synthesize structure models in the field of chemical engineering. The approach may be applied to various kinds of chemical design problems, but by now we focus on a particular type of chemical processes only, the design of plants for the food processing industry (241, 242).

A chemical plant can be viewed as a graph where the nodes represent the devices, or unit-operations, while the edges correspond to the pipes responsible for the material flow. Typical unit-operations are mixing (homogenization, emulsification, suspension, aeration etc.), heat transfer, and flow transport. The task of designing a chemical plant is defined by the given demand D, in the form of properties of various input substances, along with the desired output substance. The goal is to mix or to transform the input substances in such a way that the resulting output substance meets the imposed requirements.

The design happens by passing through (and possibly repeating) the following five steps: Preliminary examination, choice of unit operations, structure definition, component configuration, and optimization. An automation of the steps at a behavioral level would be very expensive—if possible at all. Present systems limit design support to isolated subjobs; they relieve the human designer from special simulation or configuration tasks, and the effort involved there is high enough (33, 144).

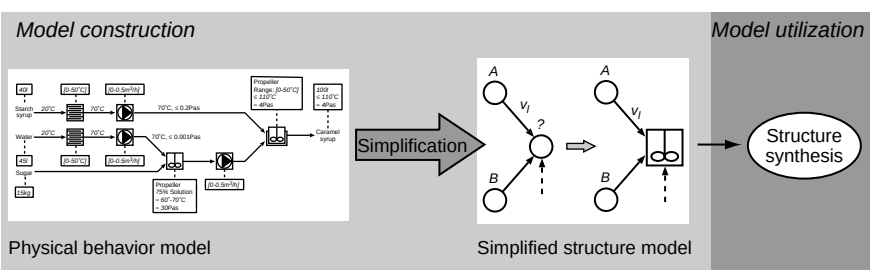


Figure A.12. The behavior model is abstracted to a structure model, say a design graph grammar $\mathcal{G}$, where physical quantities are represented as linguistic variables. This way, the synthesis space is restricted to the set of graphs $L(\mathcal{G})$.

Primary concern of our project was the investigation of possibilities to support the design process as a whole, with the long-term objective to operationalize step-spanning optimization knowledge.[8] The result of our research is comprised best at Figure A.1 on Page 89: (1) The properties of the input and output substances, D, are abstracted towards linguistic variables, $\hat{D}$. (2) At this functional level a structure

[8] The section describes a cooperative work with André Schulz in the WIMOM project; the project was supported by DFG grants PA 276/23-1 and KL 529/10-1.

model $S\langle F, \mathcal{M}\rangle$ is synthesized that fulfills $\hat{D}$ and that is used as a solution candidate for D. (3) $S\langle F, \mathcal{M}\rangle$ is completed towards a tentative behavior model $B'\langle F', \mathcal{M}'\rangle$, (4) which is then repaired, adapted, and optimized.

Clearly, the mentioned steps will not happen at the highest level of behavioral and structural details, and the underlying model is simplified in several respects. The remainder of this section outlines Step (1) and Step (2) of our work as it is illustrated in Figure A.12: The next subsection introduces the source model as it is used by the human designer, the next but one subsection describes the model simplification measures taken, while the last subsection specifies, in extracts, the design graph grammar that accomplishes the structure model synthesis.

Note that this section cannot tell a success story; Step (3) and Step (4) are not developed far enough to make a meaningful evaluation of our ideas possible[9], and, consequently, emphasis is put on modeling here. However, this new approach to the complex design problems in chemical engineering seems promising to us, and, its most interesting property is its flexibility:

- Due to the use of design graph grammars, design problem solving is not restricted to variations of a fixed plant structure. As well as that, new design knowledge on structure can easily be integrated, at the human designer's desired level of granularity.
- The completion of a structure model towards a behavior model (Step (3) in Figure A.1, Page 89) is not prescribed. Hence, behavior can be modeled at an arbitrary level of fidelity—falling back on existing simulation tools for chemical behavior models, such as ASCEND, MODEL.LA, or SPEEDUP (211, 274, 208, 173).

Underlying Models

The design of a food processing system as meant here grounds on behavior descriptions of intermediate complexity: Most of the partial and ordinary differential equations have been abstracted to purely algebraic relations or to rules of thumb. This description level is sufficient for standard design tasks in chemical engineering and is approved.

Definition A.8 (Stationary Chemical Engineering Model) *Let S be a chemical engineering system and let $\langle F, \mathcal{M}\rangle$ be a model of S. A stationary chemical engineering model over $\langle F, \mathcal{M}\rangle$ is a memoryless behavior model, $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$, whose elements are defined as follows.*

(1) $F = F_U \cup F_Z \cup F_P$ is a set of functionalities, described below. The elements in $\mathcal{M}$ designate different types of mixers, heaters, and pumps.

[9] Actually, a prototypic realization of the concept was presented on the ACHEMA 2000 fair in Frankfurt.

(2) The input variables, F_U, the constraint variables, F_Z, and the output variables, F_Y, describe the same physical quantities, i. e., substance properties such as state of aggregation, temperature, granularity, uniformity of the mixture, or viscosity. In particular, F_U defines the substances' initial properties, and F_Z defines the substance properties at various places within the system S. $F_Y \subseteq F_Z$ specifies the properties of the output substances in S. $F_P = F \setminus (F_U \cup F_Z)$ is the set of component properties.

(3) The sets U_f and Z_f designate the domains of the variables f in F. Likewise, $\mathcal{U}$, $\mathcal{Z}$, and $\mathcal{Y}$ designate the Cartesian products of the input variable domains, the constraint variable domains, and the output variable domains. $\mathcal{V}$ comprises the domains of all functionalities.

(4) The set of state variables, F_X, is empty; i. e., $B\langle F, \mathcal{M}\rangle$ is memoryless. The algebraic state prescription function $\Delta : \mathcal{U} \to \mathcal{Z}$ maps an input vector onto the global vector of substance properties, F_Z.

$B\langle F, \mathcal{M}\rangle$ is a local behavior model (cf. Page 31): The behavior constraints $\delta \in \Delta$ can be mapped one-to-one onto the components in $\mathcal{M}$.

(5) Λ is the identity mapping.

Remarks. In a design problem only the substance properties of the input and output substances, F_U and F_Y, are given, and the task is to complete these sets towards a structure model $S\langle F, \mathcal{M}\rangle$. The next paragraph provides an example.

Example Given is the task specification excerpt in Table A.5, describing the properties of three input substances (starch syrup, water, sugar) and the desired properties of the output substance, caramel syrup.

Name	State	Mass	Temperature	Viscosity
Starch syrup	liquid	36.63%	20°C	0.2-1.6 Pas
Water	liquid	15.75%	20°C	0.0010012 Pas
Sugar	solid	47.62%	20°C	–
Caramel syrup	liquid	100.00%	110°C	?

Table A.5. Substance properties of the input quantities sugar, water, starch syrup, and the output substance caramel syrup.

Based on this specification, the five design steps of the general procedure mentioned at the outset are as follows.

(1) *Preliminary Examination.* The first observation made by the engineer is that one of the substances, sugar, is a solid and must be dissolved within one of the other input liquids. Since water has a lower viscosity than starch syrup,

it will be better to mix sugar and water first and then add the starch syrup to the solution. Depending on the mass ratios the water may have to be heated beforehand to increase solubility.

(2) *Choice of Unit-Operations.* The comparison of the mass ratios of sugar and water leads to the conclusion that heating is necessary; thus, a heat transfer unit-operation is needed to heat the water. The heated water and the sugar are then mixed—for this purpose a mixing unit-operation for lower viscous substances is appropriate. To avoid recrystallization, the starch syrup should also be heated, thereby making another heat transfer unit-operation necessary. Finally, the heated sugar solution and the heated starch syrup are mixed. In order to reach the required temperature of 110°C, another heat transfer unit-operation will be needed. Furthermore, pump unit-operations are required to transport the substances between devices.

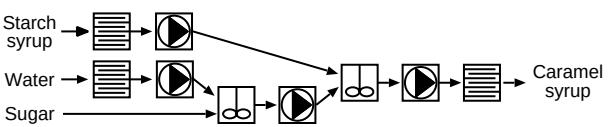


Figure A.13. A possible design of the example process.

(3) *Structure Definition.* The choice of unit-operations, although having no direct impact on the structure, allows for certain conclusions pertaining to the ordering of the unit-operations. In this case this ordering is relatively evident; Figure A.13 depicts the chosen topology. Note that this is one possible structure complying with the simplified demands; the remaining steps determine the behavior of the design and make some corrections, if applicable.

(4) *Configuration of Devices.* Based on the mass, the volume, and the other properties of the involved substances, matching devices are chosen from databases or data sheets. For the sake of simplicity we will refrain from a detailed description here and refer to Figure A.14, where the abstract design with additional data from the underlying model is shown.

(5) *Optimization.* The computed properties of the plant design usually represent feasible values, but improvement may still be possible. With this goal in mind, the parameterization process is repeated and parameters adjusted accordingly. In our case the last heat transfer unit-operation of the process chain represents an overkill—the last mixing unit-operation is then slightly changed so that only devices with a built-in heat transfer unit are considered. This change shortens the process chain, thereby reducing costs and mixing time.

Remarks. Alternatively, another design with fewer devices is conceivable. For instance, water and starch syrup could be mixed first, and the resulting solution used to dissolve sugar. This structure choice would require one heat transfer unit-operation

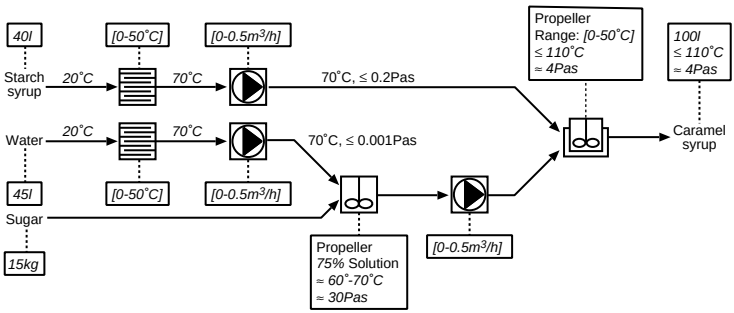


Figure A.14. Design solution showing part of the underlying model.

less than the proposed design because both water and the starch syrup have to be heated to the same temperature, which is best done if both substances are mixed together beforehand. However, this alternative would cause a longer mixing time, since the sugar must be dissolved in a more viscous solution (compared to pure water).

Model Simplification

Instead of deriving a concrete solution at the modeling level that is imposed by the supplied demands, the physical model is simplified to an abstract model. On this abstract level, a solution can be efficiently calculated and transferred back to the physical level, although some adjustments may be necessary at this point (recall the paradigm illustrated in Figure A.1). The following model simplification steps lead to a more tractable design problem; they are oriented, in as much as it is possible, at the taxonomy of model abstraction techniques compiled by Frantz (84).

Model Boundary Assumptions pertaining to the external features of the model—input variable space, global restrictions etc.—are made.

- *Single Task Assumption.* It is general practice to combine different chemical processes in order to share by-products or partial process chains. Such a combined design corresponds to the solution of different tasks simultaneously—a procedure which belongs to optimization. Here, design is restricted to $n : 1$ case, and overlapping plant structures are split and dealt with separately as shown in Figure A.15. Say, F_Y comprises the properties of a single substance.

- *Model Context.*The way models, or parts of models, are embedded into a context is clearly defined. Pumps, for example, have a strict structural relationship; they have one input and one output, i. e., the degree is fixed, and the predecessor of a pump must be another device.

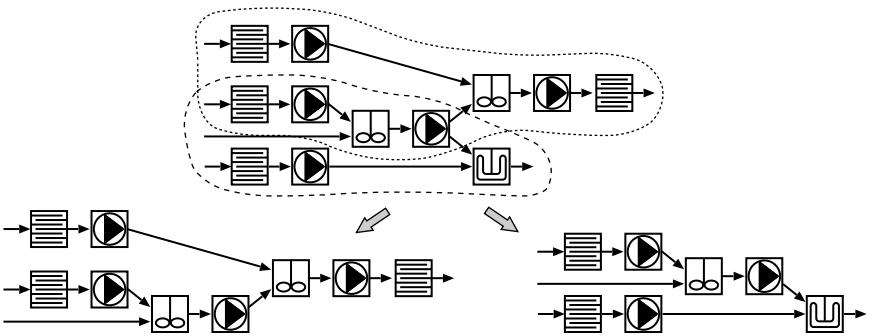


Figure A.15. Single task: Overlapping plant structures are split and treated separately.

- *Limited Input Space.* Firstly, we concentrate on tasks of the food processing branch of the chemical engineering domain. This restriction results in further simplifications: The chemical plants to be designed do not exceed a certain magnitude, as well as the range of some variables, such as temperature, which is limited to "small" values, say, below 200° C.

  Furthermore, the focus is laid on liquid mixtures. This means that at least one input has to be a fluid.

  Another restriction is the number of relevant substance properties. During design generation, decisions are taken based on the abstract values of a small set of substance properties, such as temperature, viscosity, density, mass and state. Properties such as heat capacity, heat conductivity or critical temperature and pressure are neglected at this point.

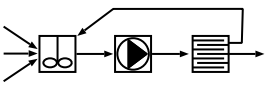
- *Approximation.* Instead of using different functions and formulas that apply under different conditions, only one function or formula covering the widest range of restrictions is used in each case. For example, there are more than 50 different formulas to calculate the viscosity of a mixture, most of which are very specialized versions and only applicable under very rare circumstances. The formula $ln(\eta) = \sum_i \varphi_i \cdot ln(\eta_i)$, however, is very often applicable and delivers a good approximation, even in the complicated cases.[10]

Behavioral Simplification Now the focus is shifted to aspects within the model, where the behavior of components is simplified:

- *Causal Decomposition.* To prevent components from exerting influence on themselves, feedback loops are ruled out. This simplification step makes structural manipulation and behavioral analysis easier.

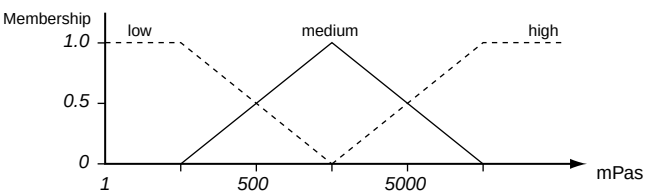
[10]The symbols φ_i and η_i designate the volume portion and the viscosity of input i.

The situation depicted below shows a cycle within a chemical plant. The purpose of the backward edge is to transport evaporated substance back into the mixer; this gaseous substance condenses on the way.



- *Numeric Representation.* Although the use of crisp values leads to exact results, fuzzy sets are used to represent essential value ranges. This simplification diminishes the combinatorial impact on our graph grammar approach, since substance properties are coded into edge labels and the use of crisp values would lead to an excessive number of rules.

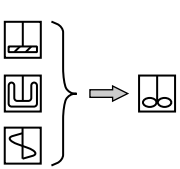
  The following figure shows the fuzzy representation of the substance property "viscosity":



- *State Aggregation.* In general, the material being processed within a device is not in one state, but actually in various different ones. For instance, inside a heat transfer device a fluid may be, depending on the reading point, cold, warm, in liquid form or gaseous. This behavior is simplified by assuming that, inside any device, a material will be in one single state.

- *Temporal Aggregation.* Time is neglected, making any statements about continuous changes to material properties no longer possible; changes to material properties are connected to entry and exit points within the plant structure.

- *Entity Aggregation by Function.* Different device entities are represented by one device performing a function common to all devices. For example, all different mixer types could be described by one special mixer, as shown below.



- *Entity Aggregation by Structure.* Devices usually consist of different parts that can be configured separately. For instance, a plate heat transfer device is composed of a vessel and a variable number of plates. The arrangement of the plates within the vessel is a configuration task.

  The following figure sketches the composition of a plate heat transfer device and of an anchor mixer.

- *Function Aggregation.* In contrast to entity aggregation by function, where devices are represented by a special device, we aggregate here functions. For instance, mixers are capable of performing different functions, such as homogenization, emulsification, aeration, suspension etc.

Derived Relationships Some fields of chemical engineering still remain unveiled and are dealt with as black boxes. In such cases one has to resort to look-up tables and interpolation, as far as sufficient information is available. For example, the output of a mixer, measured in terms of the Reynolds and Newton numbers, has to be determined experimentally, and this data is usually only available as tables.

Some model aspects of minor consequence are ignored, thereby simplifying the model as a whole. One good example are pipes used to transport the substances within the plant: In general, the length and the material of a pipe determine the degree of thermal loss of a substance being conveyed.

Remarks. The model simplification steps listed above can be classified into two different types: Steps pertaining to the model structure and steps belonging to the model behavior. Note that steps may be connected to both structure and behavior.

The steps that simplify the model structure—simple task assumption, model context, limited input space, causal decomposition, numeric representation, entity aggregation by function—yield a model that is fitting to be processed with graph grammars. These steps restrict the graph structure and specify the types and granularity of node and edge labels.

The steps belonging to the model behavior—limited input space, approximation, causal decomposition, numeric representation, state and temporal aggregation, entity aggregation by structure, function aggregation, derived relationships—result in a model suitable for qualitative simulation.

The model simplification steps performed here relate to the domain of chemical engineering. For other domains not necessarily the same steps will be appropriate; the modeler has to determine which model simplification steps are fitting individually.

Working with Simplified Models: Structure Synthesis

This section presents excerpts of a design graph grammar, $\mathcal{G}$, for the synthesis of structure models in chemical engineering; the grammar has been developed within the WIMOM project. The algorithm SYNTHESISSTEP operationalizes the search in the structure space $L(\mathcal{G})$. Starting point of SYNTHESISSTEP is the grammar $\mathcal{G}$ and some labeled graph G encoding a partial structure model of the desired system. Initially, G consists of a center node labeled "?" and adjacent nodes for the input and output substances. Figure A.16 shows the start graph related to the previous example.

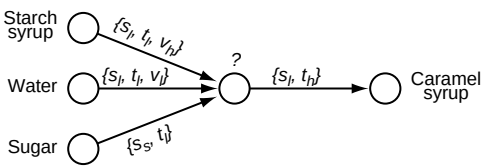


Figure A.16. Start graph for the algorithm SYNTHESISSTEP related to the example. The edge labels are sets abbreviating the substance properties from Table A.5: s_s, s_l for solid and liquid state, t_l, t_h for low and high temperature, and v_l, v_h for low and high viscosity.

SYNTHESISSTEP
Input. A graph grammar $\mathcal{G}$ and a structure model $G = S\langle F, \mathcal{M}\rangle$.
Output. A structure model containing only terminal nodes or the symbol *fail*.

(1) **if** terminal-p(G) **then** result = G
(2) **else**
(3) result = *fail*
(4) rules = find-matchings($\mathcal{G}$, G)
(5) **while** rules $\neq \emptyset \wedge$ result = *fail* **do**
(6) rule = select-rule(rules, G)
(7) rules = rules $\setminus$ {rule}
(8) result = SYNTHESISSTEP ($\mathcal{G}$, apply-rule(rule, G))
(9) **end**
(10) **end**
(11) **return** result

The algorithm employs calls to four subfunctions:

(1) *Terminal-P.* A Boolean function that checks whether G contains a node labeled with "?". If not, terminal-p(G) returns *true*.

(2) *Find-Matchings.* Searches for all possible matchings of rules in $\mathcal{G}$ in the partial structure model G, according to Definition 3.2, Page 61. In this regard, Section 3.4, Page 78, provides some complexity remarks.

(3) *Select-Rule.* The most knowledge-sensitive function with respect to search efficiency and design quality. By now, in the WIP system the rules are selected according to static rule precedences. However, the static precedences shall be replaced by a dynamic, context-sensitive precedence function, that has been computed by means of the model compilation method described in Section B.1.

(4) *Apply-Rule.* Applies a rule in compliance with the semantics specified in Definition 3.4 (Page 66) and 3.5 (Page 68) respectively.

Chemical Engineering Grammar In the sequel, rules of the design graph grammar for the structural design of food processing plants are shown; a complete listing and a formal specification can be found in (240).

- *Available Unit-Operations.* There are many devices that can be used for the same task (heating, conveying, mixing). In this place the rule set is limited to one heat transfer unit-operation, one conveying unit-operation, and two mixing unit-operations for low and high viscous substances. Additionally, two combined mixing and heat transfer unit-operations are allowed (see Figure A.17).

Figure A.17. available unit-operations of the graph grammar rule set.

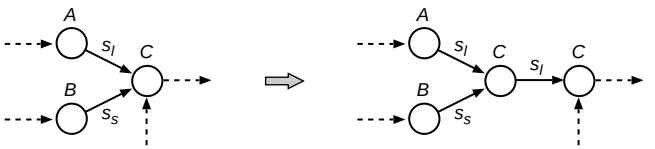
- *Substance Properties.* The most relevant substance properties are temperature, viscosity, and state. Further properties of importance are density, heat capacity etc., but we refrain from taking these into consideration here.

- *Label Class Granularity.* For each scalar substance property we choose to use a label consisting of two different variants: "low" and "high". Thus, temperature is represented by t_l and t_h, and viscosity by v_l and v_h. The property state is represented by s_s, s_l, and s_g, corresponding to the three states "solid", "liquid" and "gaseous".

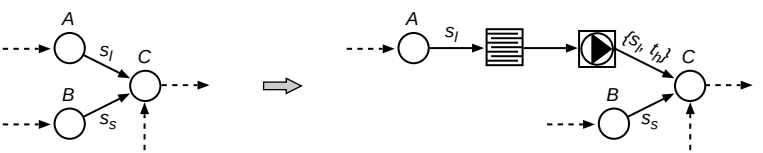
Let $\mathcal{G} = \langle \Sigma, P \rangle$ be a design graph grammar for the synthesis of chemical plants with $\Sigma = \Sigma_{\text{unit}} \cup \Sigma_{\text{substance}}$. $\Sigma_{\text{unit}} = \{p, h, m_{\text{propeller}}, m_{\text{anchor}}, hm_{\text{propeller}}, hm_{\text{anchor}}\ A, B, C, D, E\}$ where the nonterminals designate a pump, a heater, mixers, and heater-mixer combinations; $\Sigma_{\text{substance}} = \{t_l, t_h, v_l, v_h, s_s, s_l, s_g\}$. P is the set of graph transformation rules of the form $T \rightarrow \langle R, I \rangle$; the rules are divided into groups related to the substance properties state, temperature, and viscosity.

State-related rules: (1) Splitting of solids and fluids, and (2) improvement of solubility by heating (only graphical).

(1) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3\}, \{(1,3),(2,3)\},$
$\{(1,A),(2,B),(3,C),((1,3),s_l),((2,3),s_s)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5,6,7\}, \{(4,6),(5,6),(6,7)\},$
$\{(4,A),(5,B),(6,C),(7,C),((4,6),s_l),((5,6),s_s)\}\rangle$
$I = \{((D,A,E),(D,A,E)),((D,B,E),(D,B,E)),((D,C,E),(D,7,E))\}$

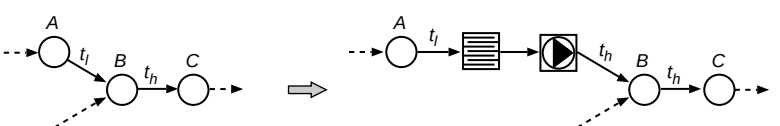


(2)

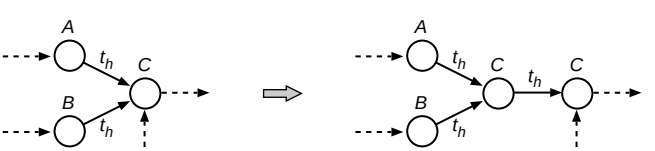


Temperature-related rules: (1) Improvement of mixing properties by heating an input, (2) improvement of mixing properties by dealing with warm inputs separately (only graphical), and (3) aggregation of mixer and heating chain into combined device (only graphical).

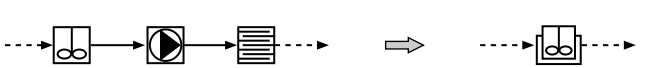
(1) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3\}, \{(1,3),(2,3)\},$
$\{(1,A),(2,B),(3,C),((1,2),t_l),((2,3),t_h)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5,6,7,8\}, \{(4,5),(5,6),(6,7),(7,8)\}, \{(4,A),(5,h),$
$(6,p),(7,B),(8,C),((4,5),t_l),((6,7),t_h),((7,8),t_h)\}\rangle$
$I = \{((D,A,E),(D,A,E)),((D,B,E),(D,B,E)),((D,C,E),(D,C,E))\}$



(2)

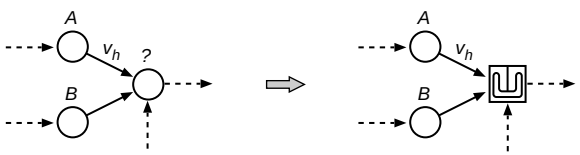


(3)

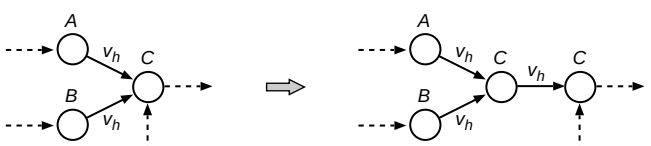


Viscosity-related rules: (1) Choice of mixer for lower viscous inputs, (2) improvement of mixing properties by dealing with high viscous inputs separately (only graphical), and (3) improvement of mixing properties by heating high viscous input (only graphical).

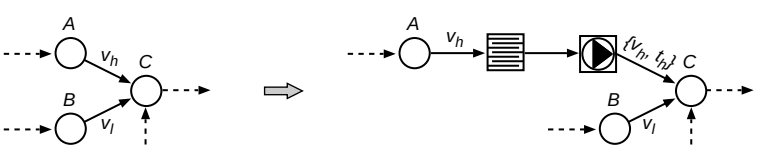
(1) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3\}, \{(1,3),(2,3)\},$
$\{(1,A),(2,B),(3,?),((1,2),v_h)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5,6\}, \{(4,6),(5,6)\},$
$\{(4,A),(5,B),(6,m_{\text{anchor}}),((4,6),v_h)\}\rangle$
$I = \{((D,A,E),(D,A,E)),((D,B,E),(D,B,E)),((D,C,E),(D,m_{\text{anchor}},E))\}$



(2)



(3)



Remarks. Observe that the substance property under consideration determines the types of structural manipulation that are possible. For instance, all substance properties lead to splitting rules, the property "temperature" is connected to optimization rules, and the property "viscosity" implies choice rules. Likewise, the other substance properties not covered by the above design graph grammar are also associated with specific structural transformations.

Synopsis

Problemclass Design of technical systems.

Problem Solving Method Design by functional abstraction: Synthesis of a structure model, which is then completed and adapted with respect to its physical behavior.

Source Model Memoryless behavior model, $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$, defined over $\langle F, \mathcal{M}\rangle$.

- *Fidelity Level F.* Substance properties such as state of aggregation, temperature, granularity, uniformity of the mixture, or viscosity, which are specified as numbers from $\mathbf{R}$.
- *Granularity Level $\mathcal{M}$.* Chemical engineering devices like mixers, heaters, and pumps.
- *Input F_U.* Properties of the input substances.
- *Output F_Y.* Properties of the single output substance.
- *State Prescription Function Δ.* Algebraic behavior descriptions; locality principle fulfilled: The constraints $\delta \in \Delta$ can be mapped one-to-one onto the components in $\mathcal{M}$.
- *Behavior Model Processing.* Local and global constraint processing methods.

Simplified Model Memoryless behavior model, $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$, defined over $\langle F, \mathcal{M}\rangle$.

- *Fidelity Level F.* Elimination of less significant component parameters and substance properties. Abstraction of the remaining numerical substance properties towards linguistic variables.
- *Granularity Level $\mathcal{M}$.* Corresponds widely to the source model.
- *Input F_U.* Fuzzified properties of the input substances.
- *Output F_Y.* Fuzzified properties of the single output substance.
- *State Prescription Function Δ.* Design graph grammar. Simplified algebraic behavior descriptions.
- *Behavior Model Processing.* Derivation based on design graph grammars. Local and global constraint processing methods.

Knowledge Source Simplifications proposed by domain experts, relating the model boundary, the synthesis space, and the behavior.

B

Model Compilation

The aim of model compilation is to shift processing effort from the model utilization phase to the model construction phase. Model compilation is a powerful construction approach to address a model's analytic complexity, its constraint complexity, or the search space complexity. The idea behind several model compilation approaches is to break global connections of the source model down to local connections. In the compiled model the local connections may be encoded as (1) exploratory short cuts within a large search space or as (2) computational short cuts bridging long-winded calculations. See Page 48, Section 2.4, for a comparison to other model construction approaches.

This chapter presents a synthesis and an analysis case study. In Section B.1, model compilation is used to automatically generate control knowledge that guides the search when solving complex resource-based configuration problems. In Section B.2, model compilation (along with model simplification) is used to derive a heuristic diagnosis model from a deep behavior model that is based on state prescription constraints.

In this sense, the first case study aims at the identification of exploratory short cuts while the second case study depends on the possibility to formulate computational short cuts. Both compilation approaches are scalable. Scale factor in the configuration case study is the depth of the analyzed search space; scale factor in the diagnosis case study it is the precision at which simulations are performed.

B.1 Generating Control Knowledge in Configuration Tasks

Configuration is the process of composing a model of a technical system from a predefined set of components. The result of such a process is called configuration too and has to fulfill a set of given constraints. Aside from technical restrictions a customer's demands constitute a large part of these constraints (34, 46, 92, 141, 165).

A configuration process may rely on both a structure model and a behavior model. If emphasis is on the former model, the configuration approach is called skeleton configuration or structure-based configuration. If a behavior model forms the backbone of the configuration process, only very few standardized configuration approaches exist. An important representative in this connection is the so-called resource-based configuration approach, which becomes center in this place.

Within the resource-based configuration approach the components involved are characterized by simplified functional dependencies, so-called resources. On the one hand, a resource-based modeling provides for powerful and user-friendly mechanisms to formulate configuration tasks (140). On the other hand, the solution of resource-based configuration problems is NP-complete, which means that no efficient algorithms exist to solve a generic instance of that problem (257).

Actually, when given a real-world configuration problem formulated within a resource-based description, the search for an optimum configuration can often be realized efficiently by means of heuristics that were developed by domain experts. Stated another way, a concrete resource-based system description can be compiled by enriching it with control knowledge. This section picks up that observation: It presents a method to automatically generate control knowledge that guides the search when solving complex resource-based configuration problems.

The compilation approach emphasizes the view that a resource-based configuration problem can be attacked at two different scenes: At a preprocessing stage, where heuristics are generated, and at a configuration stage, usually at the customer's site, where a concrete configuration problem is solved. Figure B.1 illustrates this view.

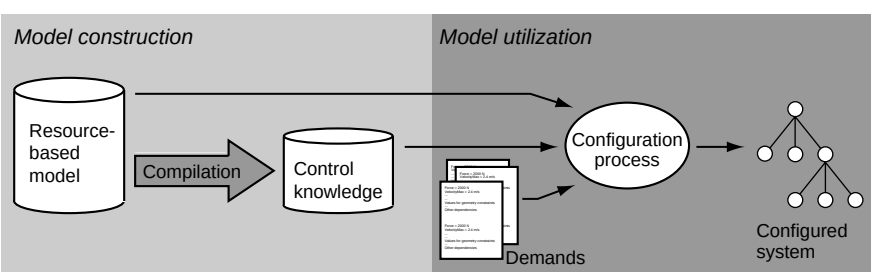


Figure B.1. Partitioning the resource-based configuration process.

That a partitioning of the configuration process is possible is in the nature of

most configuration problems: Input of the preprocessing step is the entire configuration model, and preprocessing must be re-applied whenever this model is changed. Such changes come along when new components are added or when functionalities of existing components are modified. Input of the configuration step are demand sets, which are customer-dependent. The ratio, p, of component model changes and demand set changes satisfies $p \ll 1$.

The remainder of the section is organized as follows.

(1) In the next subsection, resource-based configuration is introduced, and the balance algorithm, a method to process resource-based component models, is outlined.

(2) The performance of the basic balance algorithm can be significantly improved by exploiting heuristics that provide a decision base during the search. This is the starting point of the next but one subsection, where we show in which way such heuristics can be derived by model compilation within a preprocessing step.

The concepts presented here have been operationalized and evaluated at a real-world real-world setting(141), which is outlined in "Implementation" subsection.

Resource-Based Models and Configuration

There exist a lot of methodologies that describe in which way configuration problems can be tackled. Their adequacy depends on the configuration task, the domain, and, of course, the description of the configuration building blocks. Especially when configuring modular technical systems, resource-based configuration is an important configuration methodology.

Introduction to Resource-Based Models The resource-based model establishes an abstraction level that reduces a domain to a finite set of functionality-value-pairs (105). More precisely, the functionalities that are involved in the configuration process are divided into equivalence classes, the resources. Functionalities of the same resource class can be charged against each other, rendering the components to suppliers and demanders of resources.

E. g. when configuring a small technical device such as a computer, one functionality of a power supply unit could be its power output, f_1, and one functionality of a plug-in card could be its power consumption, f_2. Both functionalities are elements of the resource "power": A power supply unit *supplies* some power value $f_1 = a_1$, while each plug-in card *demands* some power value $f_2 = a_2$. In its simplest form, demanded values are represented by negative numbers, which are summed up with the supplied, positive numbers. Within a technically sound model the balance of each resource must be zero or positive. Figure B.2 depicts a resource-based descriptions of computer components.

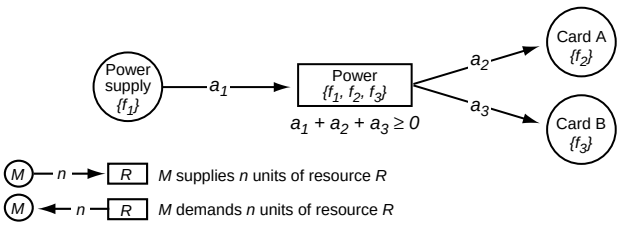


Figure B.2. Resource-based modeling of simple computer components.

With respect to the example, $\langle F, \mathcal{M} \rangle$ is defined as follows: $F = \{f_1, f_2, f_3\}$, $\mathcal{M} = \{\{f_1\}, \{f_1, f_2, f_3\}, \{f_2\}, \{f_3\}\}$. Note that the set $\mathcal{M}$ contains both components and resources. Also note that a dependency network as shown in Figure B.2 represents a structure model of $\langle F, \mathcal{M} \rangle$. Resource-based configuration means the instantiation of such structure models and the simulation of the underlying behavior models.

In the simplified resource-based configuration model all components have only a single state. As a consequence, the state description function, Δ, is constant. The output of a resource-based model is a vector comprising the summed up resources. If a component is used k times, its functionalities are k-fold supplied or demanded. Hence, the output function, Λ, is a linear form.

It is the aim of a configuration process to synthesize a model that fulfills a set of demands. Under the resource-based model, demands are charged against the output vector. Figure B.3 extends the example by introducing the resource "RAM-extension", which comprises the functionalities f_4 and f_5. For a given demand d at resource "RAM-extension" both conditions, $a_1 + k \cdot a_2 + a_3 \geq 0$ and $k \cdot a_4 + a_5 \geq d$, must be fulfilled.

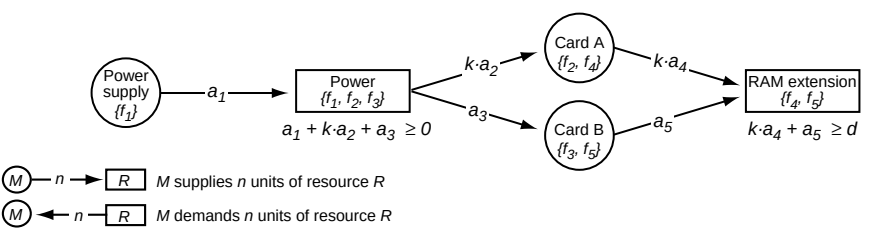


Figure B.3. In the example, $\langle F, \mathcal{M} \rangle$ is defined as follows: $F = \{f_1, f_2, f_3, f_4, f_5\}$, $\mathcal{M} = \{\{f_1\}, \{f_1, f_2, f_3\}, \{f_2, f_4\}, \{f_3, f_5\}, \{f_4, f_5\}\}$.

The resource-based model is suitable for a configuration problem if the following conditions are fulfilled:

- Structural information plays only a secondary role.

- The components can be characterized by functionalities that are supplied or demanded.
- The components' functionalities are combined in order to provide the system's global behavior.

In the following we give a precise specification of the simplified resource-based configuration problem and its solution.[1] As introduced by (114), these definitions specify the state prescription function, Δ, in a normal form as a matrix $\mathbf{A}$. Each column of $\mathbf{A}$ represents the functionality vector of a component; likewise, the rows of $\mathbf{A}$ represent the resources. I. e., an element a_{ij} defines the amount at resource i of component j. A configuration problem then is a linear inequation system defined on $\mathbf{A}$. For the example of Figure B.3 the matrix and the inequation system are given below.

$$\mathbf{A} = \begin{pmatrix} a_1 & a_2 & a_3 \\ 0 & a_4 & a_5 \end{pmatrix}, \qquad \mathbf{A} \begin{pmatrix} 1 & k & 1 \end{pmatrix}^T \geq \begin{pmatrix} 0 \\ d \end{pmatrix}$$

Definition B.1 (Resource-Based Model) *Let $\langle F, \mathcal{M} \rangle$ be a model. A simplified resource-based model over $\langle F, \mathcal{M} \rangle$ is a constant behavior model, $B\langle F, \mathcal{M} \rangle = \langle F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$, whose elements are defined as follows.*

(1) F is a set of functionalities; the elements in $\mathcal{M}$ are called objects (or components) and resources.

(2) F_Z is the set of constraint variables, and F_Y is the ordered set of output variables. The domain for each $f \in F$ is $\mathbf{Z}$. $\mathcal{V}$ comprises the domains of all functionalities.

(3) The set of state variables, F_X, is empty. The state prescription function Δ is constant, it assigns each functionality in F_Z a value from $\mathbf{Z}$. Δ can be written as an $r \times o$ matrix $\mathbf{A}$, whose rows and columns define the resources and objects in $\mathcal{M}$ respectively. In the following we will use the symbols $\mathcal{O} \subset \mathcal{M}$ and $\mathcal{R} \subset \mathcal{M}$ to designate the set of objects and resources respectively.

Between a resource $R \in \mathcal{R}$, an object $O \in \mathcal{O}$, and a matrix element a_{ij} the following relation holds:

$$\begin{array}{lcl} a_{ij} = 0 & \Leftrightarrow & O \cap R = \emptyset \\ a_{ij} > 0 & \Rightarrow & O \cap R = 1, \text{ Semantics: } O \text{ supplies } a_{ij} \text{ units of } R \\ a_{ij} < 0 & \Rightarrow & O \cap R = 1, \text{ Semantics: } O \text{ demands } a_{ij} \text{ units of } R \end{array}$$

[1] Compared to the original definitions in (192), the definitions here are weakened within the following respects: (1) The value domain of all functionalities is $\mathbf{Z}$, (2) the only way to combine two functionalities is the addition of their values, and (3) supplies and demands are compared with the "$\leq$"-operator. As a consequence, symbolic functionalities or sophisticated configuration constraints cannot be formulated straightforwardly. However, the definition reflects a great deal of the required modeling power for typical resource-based configuration problems.

(4) The output function Λ is given by $\mathbf{Ak}$, say, $F_Y = \mathbf{Ak}$. In this connection $\mathbf{k} \in \mathbf{N}^o$ is called the vector of object occurrences, or simply, configuration.

Remarks. Note that the output function Λ depends on the number of object occurrences.

Definition B.2 (Resource-Based Configuration Problems) *Given is a simplified resource-based model $\langle F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$ over $\langle F, \mathcal{M} \rangle$, several configuration problems can be stated (192, 257).*[2]

- FINDCONF
  Let $\mathbf{d}$ be a vector from $\mathbf{N}^r$, called the demand vector.
  Problem: Determine a configuration $\mathbf{k} \in \mathbf{N}^o$ such that the inequation system $\mathbf{Ak} \geq \mathbf{d}$ is fulfilled.

- FINDOPTCONF
  Let $\mathbf{d}$ be a demand vector from $\mathbf{N}^r$, and let $c : \mathcal{O} \rightarrow \mathbf{R}^+$ be a cost function that assigns some cost to each object in $\mathcal{O}$.
  Problem: Determine a configuration $\mathbf{k} \in \mathbf{N}^o$ that fulfills $\mathbf{Ak} \geq \mathbf{d}$ and that is cost-minimum.

In the following we abbreviate the problem FINDCONF with Π and the problem FINDOPTCONF with Π^*.

Processing Resource-Based Models If there exists a configuration $\mathbf{k}$ that solves the resource-based configuration problem Π, $\mathbf{k}$ can be determined with the balance algorithm. This algorithm operationalizes a generate-and-test strategy and has been implemented in the configuration systems COSMOS, CCSC, AKON, and MOKON (105, 155, 294, 273). The generate part, controlled by propose-and-revise heuristics or simply by backtracking (171), is responsible for selecting both an unbalanced (or unsatisfied) resource R and a set of components that supply $f \in R$. The test part simulates a virtual balance. A resource is called unsatisfied, if the sum of its demanded functionalities exceeds the sum of its supplied functionalities.

Basically, configuration works as follows. First, the demand set of the virtual balance is initialized with all demanded functionalities, and $\mathbf{k}$ is set to the zero-vector. Second, with respect to some unsatisfied resources R, an component set is formed; its functionalities are added to the corresponding resources of the balance, and $\mathbf{k}$ is updated by the component set. Third, it is checked whether all resources are satisfied. If so, $\mathbf{k}$ establishes a solution of the configuration problem Π. Otherwise, the configuration process is continued with the second step.

Consider a simple configuration problem where an initial demand vector $\mathbf{d}^T = (R_1, R_2) = (6, 0)$ is given. Two components, O_1 and O_2, can be used to fulfill the demand vector; they are defined in table B.1.

[2] At the same place the authors introduce also a number of decision problems.

Component	Functionalities
O_1	$\{f_1 = 2, f_2 = 1\}$
O_2	$\{f_3 = 4, f_4 = -1\}$

Resource	Functionalities
R_1	$\{f_1, f_3\}$
R_2	$\{f_2, f_4\}$

Table B.1. Resource model of the example.

After initialization, the balance has the entry $\{(R_1, -6), (R_2, 0)\}$. Now the configuration algorithm has to choose a component that fulfills the unsatisfied demand at resource R_1. If we assume that component O_2 is chosen, the balance and the actually explored search space will look as depicted in Figure B.4.

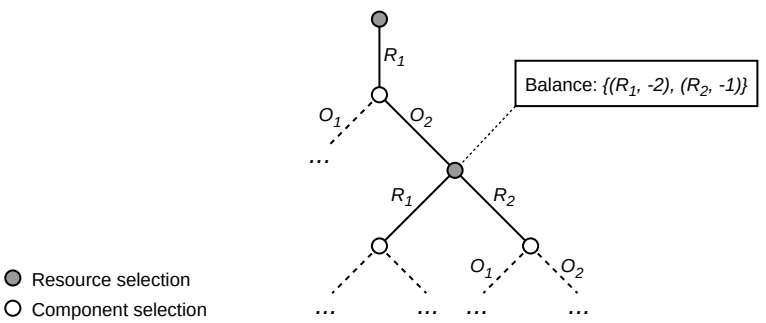


Figure B.4. Configuration situation after the first decision.

Note that in the configuration situation of Figure B.4, both a decision regarding component selection and resource selection must be made. A solution of the example is given with $\mathbf{k}^T = (O_1, O_2) = (1, 1)$ or $\mathbf{k}^T = (3, 0)$.

Compiling Resource-Based Models

Resource-based component models provide great knowledge acquisition support since the configuration knowledge consists of local connections to the very most part (257). However, the basic balance processing algorithm is of exponential time complexity.

In many real-world configuration problems the *optimum* solution must be found for a given demand vector, say, some instance of Π^* must be solved. Hence, if no powerful heuristics are at hand that control the functionality and component selection steps, merely small configuration problems can be tackled by resource-based configuration.

Resource selection is related to the search space's total depth in first place; component selection affects the effort necessary for backtracking. An "intelligent" strategy within these selection steps is the major engine of efficient balance processing (278).

In this connection the preprocessing idea comes into play. By preprocessing the resource-based component model, *implicit knowledge* relating a selection strategy *can be made explicit*, e. g. in the form of an estimation function. The following subsections outline preprocessing techniques. The considerations are not of a purely theoretical nature but have been operationalized within the configuration system PREAKON (141).

Computing a Precedence Order for Resources Let $\langle V, E\rangle$ be a graph of a resource-based model. Note that, by definition, $\langle V, E\rangle$ is bipartite, each edge $e \in E$ corresponds one-to-one to a non-zero element in $\mathbf{A}$, and that the incident nodes of e correspond to one object and one resource.

Definition B.3 (Component-Resource Graph) *Let $B\langle F, \mathcal{M}\rangle = \langle F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$ designate a resource-based model over $\langle F, \mathcal{M}\rangle$. A directed, bipartite graph $\langle V, E\rangle$ of $\langle F, \mathcal{M}\rangle$ is called component-resource graph, if for each edge $(v, w) \in E$ with matrix entry $a \in \mathbf{A}$ the following holds.*

$a > 0 \Leftrightarrow v$ *corresponds to a component and* w *corresponds to a resource*
$a < 0 \Leftrightarrow v$ *corresponds to a resource and* w *corresponds to a component*

Component-resource graphs reflect the supply and demand dependencies of a resource-based model. Consider the component-resource graph in Figure B.5. A demand in **d** should only be processed, if all components of the system that also need this resource 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.

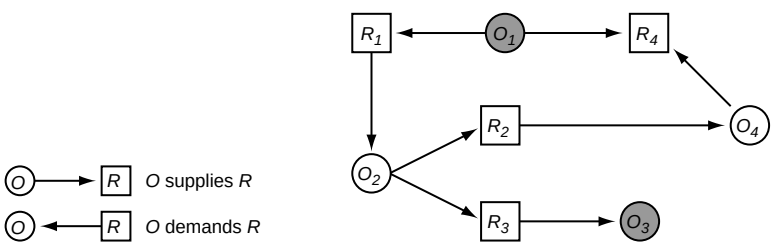


Figure B.5. A sample component-resource graph.

Obviously, the number of O's instances required to satisfy a resource can be determined without backtracking, if O's outdegree in the component-resource graph is zero (on condition that the components selected and the resources 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 resources occur in this sorting defines the succession by which unsatisfied resources should be selected from the balance.

Remarks. Given the case that $\langle F, \mathcal{M} \rangle$ represents a non-causal model, the related component-resource graph $\langle V, E \rangle$ will contain cycles. To compute a topological sorting, all strongly connected components must be detected first, a computation that can be done in $\mathcal{O}(|E|)$ for a connected directed graph (9). Then, the condensed graph can be constructed where each strongly connected component is represented by one node, and a topological sorting based on this graph can be computed.

Computing a Precedence Order for Components To satisfy an open demand at a resource R, a simple selection strategy is to choose from all components that supply R the cost-minimum one. Certainly such a greedy strategy is often too shortsighted, and we are looking for a strategy that has global configuration knowledge compiled in. Such a strategy can be operationalized by means of a function that computes a reliable *estimation of the follow-up costs* bound up with the selection of a particular component.

The subsequent simplifications are a reasonable compromise when constructing such an estimation function:

(1) A configuration situation is solely characterized by those resources that are currently unsatisfied.

(2) A resource is satisfied by components of the same type.

(3) Components are regarded as suppliers of a single resource.

(4) There exist no restrictions between components.

Remarks. Point 1 neglects that unused resources in a partial configuration may be exploited in a further course of the configuration process. Point 2 neglects that a combination of different components may constitute a more adequate solution for an unfulfilled resource than a set of components of the same type. Point 3 neglects that a component may supply several resources each of which is demanded in the partial configuration. Point 4 neglects that other constraints than supplies and demands must be satisfied by a configuration (mounting restrictions for instance).

Based on the above simplifications an estimation function $h(O, R, n)$ for the computation of follow-up costs can be directly constructed.[3] R denotes the demanded resource, n denotes the amount to which R is demanded, and O denotes a component that supplies R and that is used to satisfy the open demand. We will construct h within three steps:

(1) Each component $O \in \mathcal{O}$ has some local cost $c(O)$, but it also causes particular follow-up costs. Together they make up a component's total cost c_t.

[3] The estimation function discussed here was proposed and operationalized by Curatolo (140).

(2) A component's follow-up costs result from its demands. More precisely: Let O be a component and $d(O)$ the set of resources demanded by O. Then, of course, we would like each demand $v_d(O, R')$ of component O at resource $R' \in d(O)$ to be satisfied at minimum costs. Note that all components that will be selected to satisfy R' entail follow-up costs on their turn. I. e., if we selected component O in order to satisfy a required demand R, we would expect the following total cost c_t:

$$c_t(O, R) := c(O) + \sum_{R' \in d(O)} \min_{O' \in s(R')} \{c_t(O', R')\},$$

where

$c(O) \in \mathbf{R}^+$	local cost of component O
$d(O)$	demanded resources of O
$s(R)$	components that supply R

Note that the term for c_t assumes that each required demand can be satisfied by exactly one component. This shortcoming is addressed within the next step.

(3) The following term computes for a given amount n at resource R the number of components O that are necessary to satisfy R:

$$\left\lceil \frac{n}{v_s(O, R)} \right\rceil$$

Putting the pieces together results in an estimation function h that computes for a component O and a demand R at the amount of n the total costs:

$$h(O, R, n) := \left\lceil \frac{n}{v_s(O, R)} \right\rceil \cdot \left(c(O) + \sum_{R' \in d(O)} \min_{O' \in s(R')} \left\{ h\big(O', R', v_d(O, R')\big) \right\} \right),$$

where

$c(O) \in \mathbf{R}^+$	local cost of component O
$d(O)$	demanded resources of O
$s(R)$	components that supply R
$v_s(O, R) \in \mathbf{N}$	supply at resource R of component O
$v_d(O, R) \in \mathbf{N}$	demand at resource R of component O

Example. Recall the example from Page 129, where a simple knowledge base containing two components, O_1 and O_2, and two resources, R_1 and R_2 was given. In this place we elaborate on the same example, but we define aside from the components' resources also their local costs c.

According to the formula previously derived, h is defined as follows.

$h(O_1, R_1, n)$	$=$	$\lceil \frac{n}{2} \rceil \cdot 100$	(no follow-up costs)
$h(O_1, R_2, n)$	$=$	$n \cdot 100$	(no follow-up costs)
$h(O_2, R_1, n)$	$=$	$\lceil \frac{n}{4} \rceil \cdot (10 + 100)$	(follow-up costs for R_2)
$h(O_2, R_2, n)$	$=$	∞	(R_2 can never be satisfied by O_2)

Component	Functionalities	Local cost
O_1	$\{f_1 = 2, f_2 = 1\}$	100
O_2	$\{f_3 = 4, f_4 = -1\}$	10

Resource	Functionalities
R_1	$\{f_1, f_3\}$
R_2	$\{f_2, f_4\}$

Table B.2. Resource model of the example with local cost.

Again, the demand vector at the system searched for is $\mathbf{d}^T = (R_1, R_2) = (6, 0)$. The resulting search tree is depicted in Figure B.6.

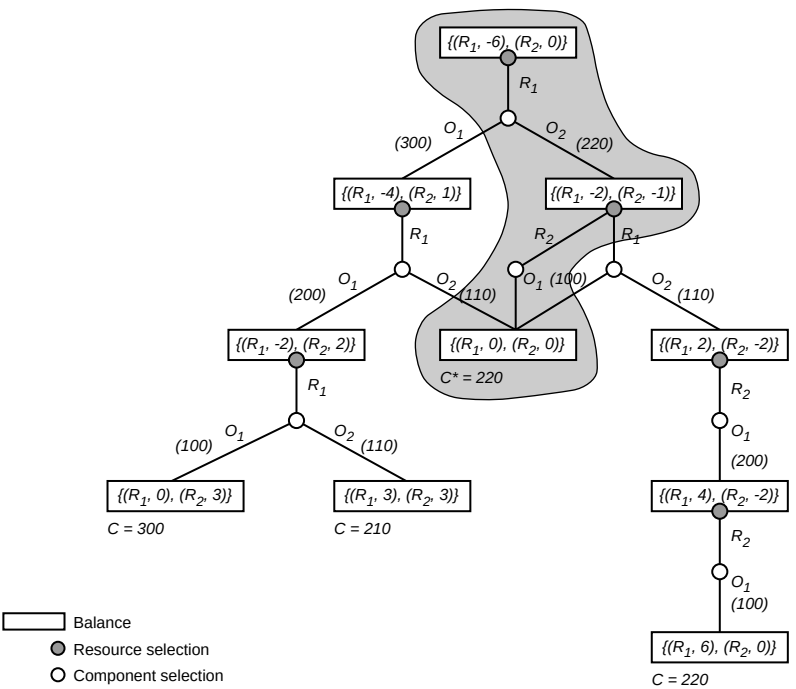


Figure B.6. Search tree of the configuration example.

The search tree is two-layered and consists of two types of nodes. Filled nodes establish choice points regarding the resource to be satisfied next. The related balance is shown framed above the node. Outlined nodes establish choice points regarding the component to be selected next. The edges of the search tree are labeled with the configuration decisions. Below the actually selected components, put in parentheses, the estimation function's values are annotated.

The tree shows in which way the control information of h is exploited. If alternative components are given to satisfy an open demand, h defines an order by which

the nodes are expanded. In the example, component O_2 is chosen at the first choice point, while O_1 is chosen at the next. The earlier a solution is found, the earlier its cost information c can be used to cut off partial configurations exceeding c.

Properties of the Precedence Order Functions The precedence order for resource selection is optimum if $\langle F, \mathcal{M} \rangle$ is causal. Following up this order will correctly inform the balance algorithm respecting the total demand at some resource. If $\langle F, \mathcal{M} \rangle$ is not causal, the tie amongst the resources involved in a dependency cycle must be broken by some heuristic: For instance, the resources could be ranked according to their size, which minimizes the number of backtracking candidates amongst the components.

The precedence order for component selection is based on the cost estimation function h. The function h cannot be used to realize an admissible Best-First search of type A^*, since h is neither consistent nor optimistic.

Remarks. (1) An algorithm is admissible if it is guaranteed to return an optimal solution whenever a solution exists (209). (2) Consistency implies that the estimation function is optimistic, and, consistency is equivalent to monotonicity (202).

Observe that the computation of the sum cost of all unsatisfied resources of a balance by means of h clearly leads to an overestimation of a configuration's total cost. In fact, even if a balance is assessed by the cost of only one unsatisfied resource, h cannot be considered as an optimistic estimation function for a configuration's total cost. This is shown now.

Let $h(O, R, n)$ be the estimated cost for a configuration that uses O to fulfill a demand of n at resource R, and let $h^*(O, R, n)$ be the optimum cost for such a configuration. h is optimistic if the inequation $h(O, R, n) \leq h^*(O, R, n)$ holds for all $O \in s(R)$, R, and n. Consider the sum term of h:

$$\sum_{R' \in d(O)} \min_{O' \in s(R')} \left\{ h\big(O', R', v_d(O, R')\big) \right\}$$

For each resource $R' \in d(O)$ separate cost estimations are performed and accumulated. But, there may be some interactivity between two objects O_1 and O_2 that are used to satisfy two resources $R_1, R_2 \in d(O)$: Object O_1 may also supply the demanded resource R_2, rendering a separate treatment of R_2 superfluous. Given this kind of interactivity between objects, $h > h^*$.

Depending on both the actual configuration problem and specialties of the domain, h can be improved or constructed according to other paradigms:

- In place of a preprocessing that first selects a resource R and then decides which of the components is suited best, a combined consideration of resources and components is conceivable. This way restrictions between components can be taken into account.
- Instead of assessing the cost with respect to a single resource, estimation functions for the simultaneous treatment of resources can be developed.

Implementation at a Real-World Problem

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 technical constraints. Customer demands, which form the starting point of each configuration process, include various telephone extensions, digital and analog services, or software specifications. Typically, there exist a lot of alternative systems that fulfill a customer's demands from which—with respect to some objective—the optimum has to be selected.

For this kind of domain and configuration problem the resource-based component model establishes the right level of abstraction: Technical constraints can be reduced to a finite set of functionality-value-pairs, which are supplied or demanded from the components. In a nutshell, the configuration problem in hand forms an instance of Π^*.

To cope with their huge and increasing number of telecommunication system variants and to reduce the settling-in period for their sales personnel, Bosch Telenorma, Frankfurt, started along with our working group the development of the resource-based configuration system PREAKON (141). Early versions of PREAKON showed the necessity of a heuristic search space exploration if optimum configurations should be found in an acceptable time, given realistic demand vectors $\mathbf{d}$.

For the following reasons we refrained from a *manual* integration of control knowledge:

(1) The control knowledge of domain experts is usually contradictory and incomplete.

(2) The additional effort bound up with maintenance of heuristic knowledge complicates a configuration system's introduction.

(3) Each modification of the knowledge base (e. g. because of new components) can potentially invalidate existing heuristics.

Instead, the model compilation concept presented in this place was pursued—the automatic generation of control knowledge by means of preprocessing.

A precise computation or a complete representation of h is impossible here, since an exhausting search for all values in O, R, and n had to be performed. As a way out, aside from the simplifying assumptions already made on Page 131, h is approximated in the following way.

Firstly, depending on both the resources, $R \in \mathcal{R}$, and the components, $O \in \mathcal{O}$, the recursion depth of h is bound to some number k. If a search depth of k is reached while the balance is still unsatisfied, an approximate value estimating the remaining cost is assumed. Secondly, $h(O, R, n)$ is computed only for a few points n, including typical minimum and maximum demand values. The sampling points are used to

construct for each component O and each demanded resource a function that interpolates h. The result is a family of $|F|$ functions $h_f(n)$, which can be evaluated at configuration runtime.

The implementation of such estimation functions $h_f(n)$ within PREAKON led to a significant speed-up for realistic instances of Telenorma's configuration problem. PREAKON was the first configuration system at Telenorma that provided realistic means for being used at the customer site. Technical details, an evaluation, and a comparison of the outlined as well as of related estimation functions can be found in (278).

Synopsis

Problemclass Synthesis of modular technical systems according to a demand vector $\mathbf{d}$.

Problem Solving Method Balance configuration.

Source Model Constant behavior model $\langle F_Y, F_Z, \mathcal{V}, \Delta, \Lambda \rangle$, defined over $\langle F, \mathcal{M} \rangle$.

- *Fidelity Level F.* Aggregated physical properties. No compatibility or continuity constraints.
- *Granularity Level $\mathcal{M}$.* Definition of components, $\mathcal{O} \subset \mathcal{M}$, and resources, $\mathcal{R} \subset \mathcal{M}$. $\mathcal{O} \cap \mathcal{R} = \emptyset$, $\mathcal{O} \cup \mathcal{R} = \mathcal{M}$. Let $r = |\mathcal{R}|$ and $o = |\mathcal{O}|$
- *State Prescription Function Δ.* The functionalities $f \in F_Z$ are constants, $F_X = \emptyset$ (memoryless). Δ defines both object-functionality-value triples, (O, f, v), with $O \in \mathcal{O}$, $f \in F_Z$, $v \in \mathbf{Z}$ and linear inequations on the elements in $\mathcal{R}$. Δ can be represented as an $r \times o$ matrix $\mathbf{A}$. The definition of Δ guarantees locality but not causality.
- *Output Function Λ.* Λ is a linear combination $\mathbf{Ak}$, $\mathbf{k} \in \mathbf{N}^o$. $\mathbf{k}$ is the vector of component occurrences, also called configuration.
- *Order of Magnitude of the Application.* $|\mathcal{O}| \approx 100$ and $|\mathcal{R}| \approx 200$; the size of a particular resource $R \in \mathcal{R}$ varies from 1 (specialty) to o (ubiquity). On average, resource size and component size are of the same order of magnitude and smaller than 20.
- *Behavior Model Processing.* Systematic generate-and-test, realized by backtracking on an And-Or-graph.

Compiled Model $\langle F, \mathcal{M} \rangle$ and $\langle F_Y, F_Z, \mathcal{V}, \Delta, \Lambda \rangle$ correspond to the source model.

- *Resource Precedence Function.* Reversed topological sorting of the component-resource graph of $\langle F, \mathcal{M} \rangle$.
- *Component Precedence Function.* Realized by $h(O, R, n)$, a cost estimation function that recursively considers follow-up costs. For runtime and space efficiency reasons, h is bound in depth and interpolated at selected sampling points.
- *Behavior Model Processing.* General best-first search with delayed termination and non-monotonic cost estimation function.

Knowledge Source Graph-theoretical analysis. Search space exploration. Selective evaluation and interpolation of cost estimation functions.

B.2 Flattening Deep Models for Diagnosis Purposes

Let a technical system S and a related behavior model $B\langle F, \mathcal{M}\rangle$ over $\langle F, \mathcal{M}\rangle$ be given. If $B\langle F, \mathcal{M}\rangle$ describes the behavior of the correctly working system, then deviations between the predicted behavior of the simulated model $B\langle F, \mathcal{M}\rangle$ and the observed behavior of the faulty system are called symptoms. Diagnosis means to explain symptoms in terms of one or more misbehaving components in $\mathcal{M}$.

The diagnosis of systems that are described by complex behavior models is a challenge (57): Both the heuristic approach and the model-based approach entail large difficulties. Following the heuristic paradigm requires the elicitation of diagnosis rules by questioning domain experts or evaluating records from the past—a road which is insecure, fault-prone, and tedious.

Following the model-based paradigm means to synthesize and simulate a behavior model and to compare predicted behavior to observed behavior (221). Model-based diagnosis, e. g. in the form of the GDE (51, 81), requires excellent simulation capabilities because several inverse simulation runs (from symptoms back to causes) are necessary til all symptoms are covered. Among others, following problems are connected to diagnosis based on first principles. (1) Long paths of interaction between components result in a large number of diagnosis candidates, which in turn result in a large number of measurements to be carried out. (2) Many technical systems have a feedback structure; as a consequence, cause-effect chains, which form the base for GDE's reasoning process on violated assumptions, do not exist.

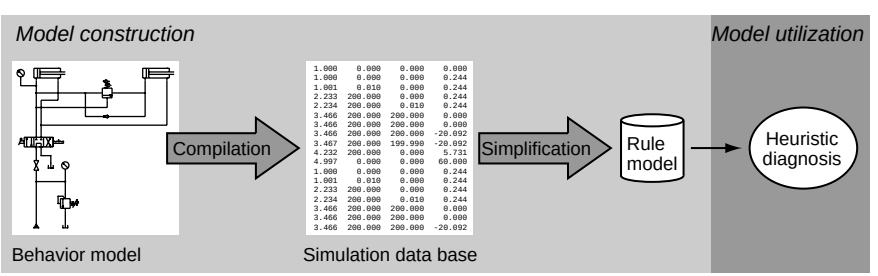


Figure B.7. Simulating a fluidic circuit in various fault modes yields a database from which, in turn, a rule-based diagnosis model is constructed.

In this section we present an approach for the automatic generation of tailored diagnosis systems for fluidic systems—it shall disburden us from long expert interviews and from consistency problems bound up with large rule bases. Clearly, an automatically generated diagnosis system cannot be expected to detect very particular faults, but it can provide a reasonable support with respect to frequently occurring component defects.

Our diagnosis approach is based on model compilation: By simulating a fluidic

circuit in various fault modes and over its typical input range a simulation database is compiled. From this database a simplified rule-based behavior model is constructed where the long cause-effect chains have been replaced with much simpler associations and which is optimized for a heuristic classification of the interesting faults (see Figure B.7). It thus may have the potential to combine the advantages of a model-based approach, such as behavior fidelity and modularity, with the efficiency and robustness of a heuristic approach.

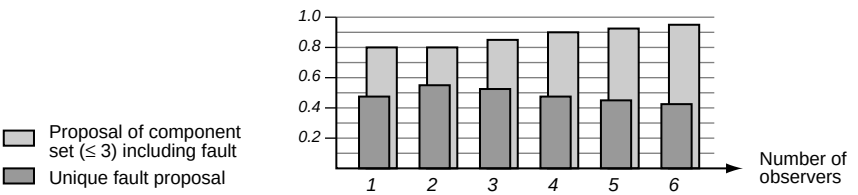


Figure B.8. Classified faults depending on the number of installed measuring devices (observers). Dark bars indicate a unique proposal of the faulty component, light bars an ambiguous proposal (≤ 3 components including fault).

An implementation of the approach and tests with medium-sized hydraulic systems, from which Figure B.8 shows an extract, are promising. The diagnosis task was to detect unseen instances of realistic component faults, which established variations of 12 different fault models.

Underlying Models

Definition B.4 (Electro-Fluidic System Model) *Let S be a fluidic system and let $\langle F, \mathcal{M} \rangle$ be a model of S. A electro-fluidic system model over $\langle F, \mathcal{M} \rangle$ is a combined discrete event/continuous time model, $B\langle F, \mathcal{M} \rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$, whose elements are defined as follows.*

(1) $F = F_U \cup F_Z$ is a set of functionalities, described below. The elements in $\mathcal{M}$ designate fluidic, i. e., hydraulic or pneumatic elements and, for control purposes, electrical elements. To the former count cylinders, motors, different types of valves, pumps, tanks, and measuring devices. To the latter count relays, switches, and power supplies, among others.

(2) F_U is the set of input variables, defining the extrinsic forces at the cylinders, but also the switching events signaled by a numerical control; F_Z is the set of constraint variables for the fluidic and electrical quantities. $F_Y \subseteq F_Z$ is the set of output variables, i. e., quantities in F_Z that shall be observed: Velocities of the cylinder pistons and pressure and flow variables at various places.

(3) The sets U_f, U_f^T, and Z_f designate the domains of the variables f in F. Likewise, $\mathcal{U}$, $\mathcal{U}^T$, $\mathcal{Z}$, and $\mathcal{Y}$ designate the Cartesian products of the input variable

domains, the constraint variable domains, and the output variable domains. The time base T is a subset of $\mathbf{R}^+$. $\mathcal{V}$ comprises the domains of all functionalities.

(4) *Δ declares a set of state variables, $F_X \subseteq F_Z$, and a state space, $\mathcal{X}$, which is the projection of $\mathcal{Z}$ with respect to F_X. If S does not contain structural singularities, there is a state variable (continuous or discrete) for each reactive element. Given a state vector $\mathbf{x} \in \mathcal{X}$, a vector of functions $\mathbf{u}(t) \in \mathcal{U}^T$, and some $t \in T$, Δ determines a constraint vector $\mathbf{z} \in \mathcal{Z}$ including a new state, say, $\Delta : \mathcal{X} \times \mathcal{U}^T \times T \to \mathcal{Z}$.*

(5) *Λ is the identity mapping.*

The precedent definition can be regarded as a correct behavior model specification for an electro-fluidic system. For our model-based diagnosis approach we need also a model of fault behavior in the sense of the GDE+ (275, 277, 54). A fault behavior model is an extension of the above electro-fluidic system model: There is an additional state variable set F_D and a second state prescription function Δ'. F_D defines fault states of the components $M \in \mathcal{M}$, such as resistance deviations, leaking amounts, or decreases in performance. Consequently the domain of Δ' is $\mathcal{D} \times \mathcal{X} \times \mathcal{U}^T \times T$; Δ' specifies the fault behavior relations of the components in $\mathcal{M}$.

Remarks. (1) Δ defines a nonlinear differential-algebraic system; details respecting behavior laws for fluidic components can be found in (15, 22, 70, 174, 175). (2) Both the electro-fluidic system model and its fault model extension are local behavior models. That is to say, a special one-to-one correspondence between the relations in Δ (Δ') and the components $M \in \mathcal{M}$ can be stated (cf. Section 2.1, Page 31). In particular, we claim that the fault behavior model complies with the no-function-in-structure principle (same Section, Page 34).

Fault Behavior in Fluidics In the following, we list important faults of fluidic components; moreover, exemplary fault behavior relations from Δ' are stated for the check valve and the cylinder.

- *Check Valve Faults.* Jamming, leaking, broken spring.

  These faults affect the resistance characteristic of the valve in first place. Let p_1 and p_2 be the pressure values at the two valve connections, let q be the flow through the valve, and let R its hydraulic resistance. Then, the pressure drop at a turbulent flow is

  $$\Delta p = R \cdot q^2 \quad \text{where} \quad \Delta p := p_1 - p_2.$$

  The resistance law is given in Table B.3 for both the correct and the faulty behavior. If the valve is operating in its range of control, the fractions are well defined and $\Delta p > p_O$.

Correct resistance behavior	Faulty resistance behavior
$R = \frac{m^2 \cdot \Delta p}{(\Delta p - p_0)^2}$	$R = \frac{m^2 \cdot \Delta p}{(\Delta p - p_0 \cdot (1 + \varepsilon_{\text{valve}}))^2}$

Table B.3. Behavior law for the resistance of a correct and a faulty check valve operating in its control range. The deviation coefficient $\varepsilon_{\text{valve}} \in F_D$ specifies a component-specific random variable according to the characteristic shown in Figure B.9 (left).

The variable $\varepsilon_{\text{valve}}$ in the faulty resistance equation is a state variable from F_D. Since the resistance of a malfunctioning valve is a continuous quantity, $\varepsilon_{\text{valve}}$ is modeled as a continuous random variable. The left-hand side of Figure B.9 shows a characteristic density function for $\varepsilon_{\text{valve}}$, the right-hand side classifies the fault behavior with respect to its seriousness based on two Fuzzy membership functions for the linguistic variables "simple fault" and "severe fault". See (24, 59) for an introduction to Fuzzy modeling.

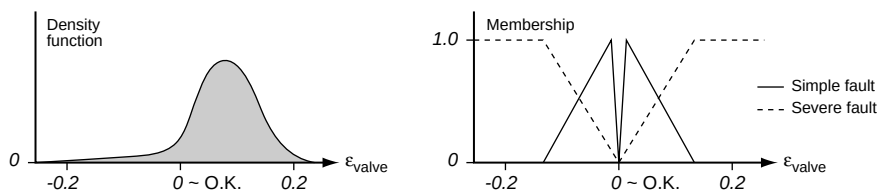


Figure B.9. A sample distribution of the resistance deviation of a faulty check valve (left) and the fault classification with respect to fault seriousness (right). Note that the left curve does nothing say about the valve's failure rate.

- *Cylinder Faults.* Slipping, interior or exterior leaking.

  The next but one equations from Δ' model a pressure-dependent leakage flow resulting from a defect sealing. Let p_1 and p_2 be the pressure values at the two cylinder connections, and let q_1 and q_2 be the respective flows. Then the balance of forces at the cylinder always is

$$F = a_k \cdot p_1 - a_r \cdot p_2 + d \cdot \dot{x} + m \cdot \ddot{x}$$

  where a_k, a_r, d, and m designate the piston area, the ring area, the coefficient of friction, and the moved mass. The flow equations take a leakage flow q_l into account:

$$q_1 = a_k \cdot \dot{x} + q_l \qquad q_2 = -a_r \cdot \dot{x} - q_l \quad \text{where} \quad q_l := \sqrt{|p_1 - p_2| \cdot \varepsilon_{\text{cyl}}}$$

  As before, a component-specific density function and a fault behavior classification must be stated for the deviation coefficient $\varepsilon_{\text{cyl}} \in F_D$.

- *Throttle Valve Faults.* Incorrect clearance, sticking.
- *Directional Valve Faults.* Defect solenoid, contaminated lands.
- *Pump Faults.* Decrease in performance.

Flattening a Deep Model by Compilation

This subsection introduces a powerful diagnosis approach for systems that are described by behavior models $B\langle F, \mathcal{M}\rangle$ as specified in Definition B.4—i. e., complex, continuous time models, which may also contain feedback loops.[4] Main idea of the diagnosis approach is model compilation. However, model simplification does also play a role: Simplification techniques are employed to generalize the compiled model and to make it manageable. The entire model construction process comprises five steps.

(1) *Behavior Model Contamination.* The local behavior specification $\Delta_M \subseteq \Delta$ of some component $M \in \mathcal{M}$ is replaced with its faulty counterpart $\Delta'_M \subseteq \Delta'$, resulting in a fault behavior model $B'\langle F', \mathcal{M}'\rangle$. Whichever of the single fault or multiple fault assumption is pursued, the behavior contamination happens either for one component at a time or for several components at once.

(2) *Systematic Simulation.* The objective is to learn as much as possible about $B'\langle F', \mathcal{M}'\rangle$. For this purpose $B'\langle F', \mathcal{M}'\rangle$ is excited with meaningful values for the input variables in F_U.

Step (1) and (2) are repeated for all fault candidates $M \in \mathcal{M}$. Together, they realize a compilation of $B'\langle F', \mathcal{M}'\rangle$: Processing effort, i. e., the simulation in our case, is shifted from the model utilization phase to the model construction phase (cf. Section 2.4, Page 2.4). Result of the compilation is a simulation database $\mathcal{C}$. The subsequent steps, (3)–(5), make up the model simplification. Figure B.7 on Page 138 illustrates this division of the model construction process.

(3) *Data Abstraction.* Abstraction means to generalize the simulation database $\mathcal{C}$ by mapping its values onto a small number of deviation intervals (the symptoms), without loosing too much of the discrimination information. Result of this step is an interval database $\mathcal{C}_I$.

(4) *Focusing.* By a simulation, all physical quantities of a model can be observed. In contrast, at a real system, symptoms can be observed only at the few points where measuring devices have been installed. To ensure that these points are useful for diagnosis purposes we determine them—following the objective to maximize their fault discrimination capability. Result of this step is the observable subset of $\mathcal{C}_I$, the observer database $\mathcal{C}_O$, where $\mathcal{C}_O \ll \mathcal{C}_I$.

[4] The approach was operationalized and verified in a close cooperation with Uwe Husemeyer (115).

(5) *Rule Generation.* By means of data mining methods, which treat the different intervals as propositional symbols, associations, say, rules between symptoms and faults are sought within $\mathcal{C}_O$. Result of this step is a rule database $\mathcal{C}_R$ where $\mathcal{C}_R \ll \mathcal{C}_O$.

The remainder of this subsection is devoted to selected aspects of the model construction steps (2)–(5). Clearly, since all construction steps are fairly complex their succinct presentation cannot come up to all respects, and references for further reading are given.

Systematic Simulation The evaluation results, a few of which are presented later on, significantly prove the success of our ideas. This paragraph outlines reasons for this success from the modeling and simulation standpoint.

One pillar of our approach is the approximation of a huge amount of simulation data by a comparatively small rule set. This is fruitful only if the simulation data can be generalized, that is to say, learned. Recall in this connection that $B\langle F, \mathcal{M}\rangle$ (or $B'\langle F', \mathcal{M}'\rangle$) establishes a combined discrete event/continuous time model. I. e., the trajectories of the constraint variables in F_Z can be considered as piecewise continuous segments, which are called "phases" here. The discrete constraint variables such as valve positions, relays, and switches are constant within a phase, and between each two consecutive phases one or more of them change their values. The continuous constraint variables such as pressures, flows, velocities, positions, etc., which are the target of our learn process, follow continuous but diverse curves. Due to the dynamic nature of $B\langle F, \mathcal{M}\rangle$ the curves show a decreasing, increasing, oscillating, or some superimposed characteristic.

From the viewpoint of diagnosis, the (quasi-)*stationary* values (in each phase) of the continuous constraint variables are in the role of symptoms, since they can be observed at the measuring devices. Our working hypothesis is that between the continuous input variables and many of the continuous constraint variables a monotonic characteristic can be assumed—as long as a single phase is considered. The simulation procedure reflects this "single-phase-monotonicity hypothesis" as follows.

Given are an initial state vector $\mathbf{x}_0 \in \mathcal{X}$ or $\mathbf{x}_0 \in \mathcal{D}\times\mathcal{X}$, a vector of—typically constant—input functions $\mathbf{u}(t) \in \mathcal{U}^T$, and some $t \in T$. Then, during simulation, samples $\mathbf{z}$ of the resulting vector of constraint trajectories are drawn at those points in time τ, $\tau \leq t$, where a state change is imminent. Each constraint vector $\mathbf{z} = (z_1, \ldots, z_{|F_Z|})$ extracted this way is taken as a vector of stationary values representing the phase where it was drawn. With respect to the compilation process it is enriched by further information: A unique number $\pi \in \mathbf{N}$ designating its phase, the responsible vector of input function values at time τ, $\mathbf{u}(\tau) = (u_1, \ldots, u_{|F_U|})$, and a vector $\mathbf{d}$ encoding component faults $d \in D$ along with membership values specifying the seriousness of the faults. For the time being we commit ourselves to the single-fault case, and $\mathbf{d}$ is of the form (d, μ_d). Altogether, a fault-simulation vector $\mathbf{c}$ is constructed:

$$\mathbf{c}(\pi, \mathbf{u}, \mathbf{d}) := (\pi, u_1, \ldots, u_{|F_U|}, z_1, \ldots, z_{|F_Z|}, d, \mu_d)$$

If the behavior model was not contaminated, that is to say, a faultless simulation has been performed, we write $\mathbf{c}(\pi, \mathbf{u})$ instead of $\mathbf{c}(\pi, \mathbf{u}, \mathbf{d})$.

Under the single-fault assumption, the total number of simulations, n, depends on the number of input quantities, $|F_U|$, the desired resolution of the input range, r, the number of component faults, $|D|$, and, for the faulty component currently chosen, its number of fault behavior graduations, s. Hence, the input sample number is in $\mathcal{O}(r^{|F_U|})$, the fault behavior variations are in $\mathcal{O}(|D| \cdot s)$, and n is in $\mathcal{O}(r^{|F_U|} \cdot |D| \cdot s)$. The n samples of simulation vectors c make up the simulation database $\mathcal{C}$.

Remarks. (1) Note that each simulation entails for each phase π a model synthesis problem followed by a solution of the resulting state-space or steady-state model. (2) Related to our purposes, the systematic simulation of a combined discrete event/continuous time model provides some pitfalls. Depending on the input functions $\mathbf{u}(t) \in \mathcal{U}^T$ or a component's fault type, phases may be dropped. Clearly, the application of the $\ominus$-operation within the data abstraction step (see below) does only make sense for vectors $\mathbf{c}$ that stem from the same phase. Therefore, the system DÉJÀVU, which operationalizes the diagnosis approach, contains heuristics for both the detection of missing phases and the introduction of dummy phases.[5]

Data Abstraction Most probably, the highest importance in the presented diagnosis approach comes up to the skillful abstraction of the simulation data, which transforms the raw data $\mathcal{C}$ towards the interval database $\mathcal{C}_I$. It is realized within three steps.

- *Difference Computation.* Based on the operator "$\ominus$", for each fault-simulation vector $\mathbf{c}(\pi, \mathbf{u}, \mathbf{d}) \in \mathcal{C}$ the difference of its constraint variables $\mathbf{z}$ to the faultless simulation vector $\mathbf{c}(\pi, \mathbf{u})$ with same $\mathbf{u}$ in the same phase π is computed.

  The measuring instruments in the real system S give information about effort variables and flow variables. The former are undirected, and a difference between two values of this type is computed straightforwardly. The latter contain directional information, and their difference computation needs a more thorough analysis.

  Let $z^{(x)}$ and $z^{(y)}$ be two values of flow variables. Then the operation "$\ominus$" is defined as follows.

$$z^{(x)} \ominus z^{(y)} = \delta_z = \begin{cases} z^{(x)} - z^{(y)} & \text{if } z^{(x)},\ z^{(y)} \text{ are unidirectional} \\ \text{“0/+”} & \text{if } z^{(x)} = 0,\ z^{(y)} > 0 \\ \text{“0/-”} & \text{if } z^{(x)} = 0,\ z^{(y)} < 0 \\ \text{“+/0”} & \text{if } z^{(x)} > 0,\ z^{(y)} = 0 \\ \text{“+/-”} & \text{if } z^{(x)} > 0,\ z^{(y)} < 0 \\ \text{“-/0”} & \text{if } z^{(x)} < 0,\ z^{(y)} = 0 \\ \text{“-/+”} & \text{if } z^{(x)} < 0,\ z^{(y)} > 0 \end{cases}$$

[5] A more elaborate presentation of this system is given in (115) under the name ARGUS. The employed fluid simulator is ARTDECO (257, 262).

The strings such as "0/+" read as "zero instead of positive", etc. Since "$\ominus$" does not define an injective operation, the difference computation is bound up with a loss of information.

- *Generalization.* If $z^{(x)}$ and $z^{(y)}$ are unidirectional flow values or if they are values of two effort variables, the $\ominus$-operation maps into $\mathbf{R}$. We now generalize these difference values from $\mathbf{R}$ towards intervals—acting on the maxim of the previously introduced single-phase-monotonicity hypothesis.

  If no event occurs, we expect that Δ_u, the restriction of the state prescription function with respect to a single input variable $u \in F_U$, behaves monotonically for significantly large variations of u. In particular, we expect that Δ'_ε, the state prescription function in dependence on a single deviation coefficient ε at constant input $\mathbf{u}$, behaves monotonic within the standard deviation interval of ε (see again the left-hand side of Figure B.9).

  If the conditions are fulfilled for several constraint variables, the generalization of the single simulation differences towards intervals is justifiable. In this connection, we collect for each constraint variable $z \in F_Z$ and for each phase π its difference values δ_z (short for $\delta_{z,\pi}$) and construct a one-to-one mapping onto a set of intervals $\mathcal{I}_z$ (short for $\mathcal{I}_{z,\pi}$). Except for the direct neighbors of zero, the difference values form the center of their associated intervals; moreover, the intervals do not overlap each other, and they cover the entire domain of z. Let $\delta_z^{(i)}$ and $\delta_z^{(i+1)}$ be the interval centers of adjoining intervals and $sign(\delta_z^{(i)}) = sign(\delta_z^{(i+1)})$, then the interval border between these intervals lies in the middle, at $(\delta_z^{(i)} + \delta_z^{(i+1)})/2$.

  Figure B.10 exemplifies for some constraint variable z and some phase π eight simulations that result from several inputs and fault variations. The letters below the $\delta_z^{(i)}$ specify the behavior model contaminations, say, the chosen component faults; above the δ_z-axis the associated intervals in $\mathcal{I}_z$ are shown.

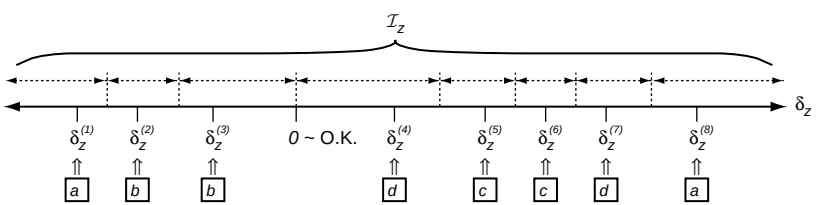


Figure B.10. Eight difference values $\delta_z^{(i)}$, which result from one faultless simulation and eight fault simulations $z^{(i)}$ for some constraint variable z within a certain phase; the differences have been generalized towards nine intervals (including the zero interval). The letters below encode the fault contaminations in the behavior model $B\langle F', \mathcal{M}'\rangle$ that led to the constraint variable values $z^{(i)}$.

Note that the computation of adjoining interval borders according to the rule $(\delta_z^{(i)} + \delta_z^{(i+1)})/2$ implies a linearity assumption on Δ and Δ': Variations $\delta_{\mathbf{u}}$ and

δ_ε of the input vector and the deviation coefficient vector are independent of the actual amount of the varied variables. Clearly, this linearity postulation further tightens the assumptions of the single-phase-monotonicity hypothesis.

That the further tightening is admissible becomes clear if one takes a closer look at the functions in Δ and Δ'. Many of the nonlinear connections are analytical functions of square characteristics, whereas each difference $\delta_z^{(i)}$ between the faultless simulation and a sample $z^{(i)}$ of the fault simulations can be considered as an operating point. And, a linear approximation close to some operating point corresponds to an approximation of the functions in Δ and Δ' by the first term of a Taylor series, which is bound up with a small error for polynomials of degree two (237).

- *Interval Reduction.* Actually, we could stop at this point and use the phase-specific intervals $\mathbf{I} \in \mathcal{I}_z$, $z \in F_Z$, to learn rules of the following or similar form:

$$\text{If} \quad \delta_{o_1} \in \mathbf{I}_{o_1} \ \wedge \ \delta_{o_2} \in \mathbf{I}_{o_2} \ \wedge \ \delta_{o_3} \in \mathbf{I}_{o_3} \quad \text{Then} \quad \text{“Fault } a \text{ is occurred”}$$

where the δ_{o_i} are symptoms that have been observed at the real system S within phase π. There are two important aspects that advised us to subtly reduce the sets of intervals. Firstly, our intervals are not purposefully constructed but result from a large number of simulations; i. e., they may be too small or come along with ill-formed interval boundaries when compared to human read-off practices. Secondly, a large number of small intervals is opposed to the generalization thought; it may lead to highly specialized rules that never become applied in practice. For instance, look at the difference values $\delta_z^{(2)}$ and $\delta_z^{(3)}$ in Figure B.10: Their associated intervals can be unified without loosing discrimination information.

In our diagnosis system DÉJÀVU this shortcoming is addressed as follows. Depending on the interesting physical quantity $z \in F_Z$, the favorable number r_z of read-off intervals is acquired, assuming that commercial measuring devices will be employed. Given some quantity z with interval set $\mathcal{I}_z$, a lower bound ρ_z for the interval width is determined by dividing the range (the sum of the smallest negative and the largest positive value of the δ_z) by r_z. To guarantee plain interval boundaries for the human sense of esthetics, ρ_z is approximated by the function $\widetilde{\rho}_z$:

$$|\rho_z - \widetilde{\rho}_z(n)| \to \min \quad \text{where} \quad \widetilde{\rho}_z(n) = n \cdot 10^{\lfloor \log_{10} \rho_z \rfloor}, \quad n \in \{1, \tfrac{5}{2}, 5, 10\} \quad \text{(B.1)}$$

The interval boundaries in $\mathcal{I}_z$ are moved to the closest integer multiple of the solution of approximation (B.1); thereafter, all adjoining intervals that contain the same set of faults are unified. Figure B.11 illustrates the procedure at the previous example.

Due to the abstraction step we leave the domain of real numbers, $\mathbf{R}$, and continue on a symbolic (propositional-logical) level with weak ordinal information: For each

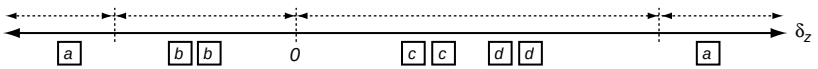


Figure B.11. The DÉJÀVU interval reduction applied to the example in Figure B.10. The interval widths are multiples of a common basis $\tilde{\iota}_z$; intervals with equal faults have been unified.

constraint variable $z \in F_Z$ and for each phase π, a new domain I_z (short for $I_{z,\pi}$) is introduced. I_z is the union of the special symbol set $\{0/+, 0/-, +/0, +/-, -/0, -/+\}$ and a set of interval names, which map in a one-to-one manner onto the reduced set of real-valued intervals $\mathbf{I} \in \mathcal{I}_z$. In the sequel, the symbolic interval database that emerges from $\mathcal{C}$ by data abstraction is denoted with $\mathcal{C}_I$. Note that the number of simulation vectors has not been reduced, say, $|\mathcal{C}| = |\mathcal{C}_\mathcal{I}|$.

Remarks. From its nature, the data abstraction step can be compared to discretization methods that transform a cardinal domain to an ordinal domain in order to make a classification or learning approach applicable (216, 56). According to Dougherty et al. such transformation methods can be distinguished with respect to locality, supervision, and interdependency. The presented method is global, since the interval formation is applied to the entire range of a variable; it is supervised, since it exploits classification knowledge (the faults within an interval); however, it does not consider dependencies between variables. Anyway, note that we are not solving a true discretization problem, since we are working on a sample database and not on continuous functions.

Focusing Based on the state prescription function Δ' of the fault behavior model $B'\langle F', \mathcal{M}'\rangle$, values for *all* constraint variables in F_Z are computed. In fact, restricted to a handful of measuring devices, only a small subset O of F_Z, $|O| = k$, can be observed at the real system S. The objective of the focusing step is to determine the most informative constraint variables in F_Z—or, speaking technically, to place a set of k measuring devices such that as much faults as possible can be classified by interpreting the k displayed values. In the sequel, the observed constraint variables are also called observers.

Diagnosis Background. The deviations found when comparing the predicted behavior of the simulated model $B\langle F, \mathcal{M}\rangle$ to the observed behavior of the system S are called symptoms. These symptoms have to be explained in terms of one or more misbehaving components, so to speak, they have to be reproduced by a simulation of some contaminated model $B'\langle F', \mathcal{M}'\rangle$. A set of correctly modeled components whose simulated behavior does not correspond to the observed behavior contains at least one faulty component and is called a conflict. A faulty component that does not contradict any of the observed symptoms is called a diagnosis candidate, or short, a candidate. A candidate that is affirmed to be the cause of all symptoms is called a diagnosis.

A model-based diagnosis approach such as the GDE[6] is organized as a cycle of the

[6]GDE stands for "General Diagnostic Engine" (51, 81). It is the most popular model-based di-

following tasks: behavior simulation, conflict identification, candidate generation, and candidate discrimination. Observations at the system S play a key role because they are the driving force within the cycle.

At this point, the model compilation idea becomes apparent: Model compilation breaks open the diagnosis cycle. All possible observations have already been made, namely offline; they are stored in the simulation database $\mathcal{C}$ and $\mathcal{C}_I$ respectively.[7] And, the restated question from above is: Which of the quantities in F_Z can be used to form meaningful diagnosis rules?

We will answer this question by analyzing for each phase π the sets $I_z, z \in F_z$ of symbolic intervals with respect to their correlations to the set D of (symbolic) component faults. The analysis covers both dependency aspects and information-theoretical considerations.

- *Observer Dependency.* Clearly, observers that depend on each other correlate in their diagnosis information and must be excluded from further examination. Because of good-natured domain properties and, in particular, the subsequent multivariate rule generation step, the dependency analysis here is narrowed to the bivariate case. Since the observers' domains are nominally scaled for the most part, the contingency coefficient of Pearson is used. It relies on the χ^2 contingency, which measures the association between two variables in a two-way table. Table B.4 shows the generic structure of such a two-way table (left) and a concrete example (right).

	$\tau_1 \in I_{o_2}$	…	$\tau_{r_2} \in I_{o_2}$	Σ
$\iota_1 \in I_{o_1}$	$h(\iota_1, \tau_1)$	…	$h(\iota_1, \tau_{r_2})$	$h(\iota_1, \circ)$
⋮	⋮	⋱	⋮	⋮
$\iota_{r_1} \in I_{o_1}$	$h(\iota_{r_1}, \tau_1)$	…	$h(\iota_{r_1}, \tau_{r_2})$	$h(\iota_{r_1}, \circ)$
Σ	$h(\circ, \tau_1)$	…	$h(\circ, \tau_{r_2})$	n

	$\delta_{p_3} < 20$	$\delta_{p_3} \geq 20$	Σ
$\delta_{q_7} < 1.5$	a b c	a d	5
$\delta_{q_7} \geq 1.5$	d	b c	3
Σ	4	4	8

Table B.4. Generic structure of a two-way table for the variables o_1, o_2 (left); r_1 and r_2 define the number of symbolic intervals, $|I_{o_1}|, |I_{o_2}|$, in the domains of o_1 and o_2. The right table shows for observed differences at pressure p_3 and flow q_7 the distribution of four component faults.

Given two observers, $o_1, o_2 \in F_Z$ and the set of all component faults, D, then $h(\iota, \tau)$ designates the phase-specific frequency of tuples (ι, τ, d), $\iota \in I_{o_1}$, $\tau \in I_{o_2}$, and $d \in D$:

$$h(\iota, \tau) = \left| \left\{ (\iota, \tau, d) \mid \mathbf{c} \in \mathcal{C}'_I \wedge \mathbf{c} = (\iota, \tau, d) \right\} \right|$$

agnosis approach and has been varied, improved, and adapted within many projects. From a logical viewpoint, the GDE implements diagnosis as a reasoning task grounded on O.K. and not-O.K. assumptions for components.

[7] Diagnosis systems that follow this paradigm have everything of S already seen—a fact, which advised us to name our system DÉJÀVU.

where $\mathcal{C}_I'$ denotes the projection of the interval database $\mathcal{C}_I$ regarding some phase π, two constraint variables, o_1, o_2, and the component faults. Since one is interested in the association between two variables related to *different* component faults, the computation of $h(\iota, \tau)$ disregards multiple occurrences of the same variable instantiations (ι, τ) associated with the same fault.

Let $o_1, o_2 \in F_Z$ be two observers with the symbolic interval domains I_{o_1} and I_{o_2} for a certain phase π. Then, their χ^2 contingency is defined as follows (188, 103).

$$\chi^2(o_1, o_2) = \sum_{\iota \in I_{o_1}} \sum_{\tau \in I_{o_2}} \frac{\left(h(\iota,\tau) - \widetilde{h}(\iota,\tau)\right)^2}{\widetilde{h}(\iota,\tau)} \quad \text{where} \quad \widetilde{h}(\iota,\tau) = \frac{h(\iota,\circ) \cdot h(\circ,\tau)}{n}$$

The observed frequencies $h(\iota, \tau)$ are compared to the frequencies $\widetilde{h}(\iota, \tau)$ that one would expect if there were no association between the variables; n designates the number of considered tuples. Two observers are independent from each other if $\chi^2(o_1, o_2) = 0$ holds. To obtain the unique range $[0; 1]$ for χ^2-values of dependent variables we compute the contingency coefficient $C^*(o_1, o_2)$ after Pearson.

$$C^*(o_1, o_2) = \sqrt{\frac{\chi^2(o_1, o_2)}{n + \chi^2(o_1, o_2)}} \cdot \sqrt{\frac{r}{r-1}} \quad \text{where} \quad r = \min\{|I_{o_1}|, |I_{o_2}|\}$$

Note that the number of intervals for some constraint variable $z \in F_z$ is significantly smaller than 10, whereas the number of potential observers, $|F_Z|$, depends on the size of the system S, and it is in the magnitude of 10^2 for middle-sized systems yet. Hence, the effort for the observer dependency analysis is assessed with $\mathcal{O}(|F_Z|^2)$. Within our experiments we gave a limit of 0.6 for the contingency coefficient C^*, resulting in the exclusion of 40-50% of the blindly positioned observers.

- *Observer Information.* Only very few observers $O \subset F_Z$ are actually installed in a real system: $|O| < |F_Z| \cdot 10^{-1}$. This fact underlines the importance that comes up to an intelligent construction of O. At heart, all considerations presented here are based on the idea of hypothetical measurements; the idea goes back on the work of Forbus and de Kleer who argue as follows.

> *"If every device quantity were observable and measurements were free, then the best diagnostic strategy would be to measure everything."*
>
> Forbus and de Kleer, 1993, pg. 631

Clearly, measurements are not free, and Forbus and de Kleer try to estimate the measuring cost hidden in a particular diagnosis situation. They stipulate

on the following setting. (1) Every measurement can be made at equal cost. (2) Amongst the set of possible diagnosis, only those of minimum cardinality are of interest—a principle which is also known as Occam's razor.

Reasoning by hypothetical measurements means to evaluate for all $z \in F_Z$, i. e., for all potential observer places, how an observed difference δ_z would reduce the set of possible diagnosis, D. For instance, assuming that $D = \{a, \ldots, h\}$ and that we are given the simulation results shown in Table B.4, a measurement of q_7 resulting in the symptom "$\delta_{q_7} \geq 1.5$" complies only with the component faults b, c, and d. However, the measurement could also result in the symptom "$\delta_{q_7} < 1.5$" where the component faults a,...,d come into question.

With respect to the database $\mathcal{C}_I$ and a given phase π, let $\kappa(z, \iota) \subseteq D$ designate the set of diagnoses that comply with symptom "(z, ι)"; i. e., diagnoses entailing a difference δ_z at quantity $z \in F_Z$, which is characterized by the interval ι. Related to the example, $\kappa(q_7, \text{"}\geq 1.5\text{"}) = \{b, c, d\}$. If one presumes that all diagnoses (component faults) in the set D occur equally likely[8], then the likelihood that a particular symptom "(z, ι)" will occur can be estimated by $|\kappa(z, \iota)|/|D|$, the fraction of diagnoses that comply with the symptom.

We are now in a position to state for every observer place z both (1) the likelihood that an observed difference δ_z lies within some $\iota \in I_z$ and (2) the possible diagnoses $\kappa(z, \iota)$ one of which must have caused δ_z. If we also knew the measurement effort to discriminate amongst the remaining diagnoses $\kappa(z, \iota)$, the most informative observer place in F_Z could be determined. For this purpose, the simplifying assumption is made that the diagnoses D are equally distributed over the $|I_z| = r$ intervals in I_z, $z \in F_Z$. Henceforth, $\log_r \kappa(z, \iota)$ defines a lower bound for the number of measurements that are necessary to isolate each of the faults in $\kappa(z, \iota)$. Putting all together, we obtain formula (B.2), which estimates the measuring (discrimination) effort to identify a component fault from D using observer z when given the diagnosis situation described by the interval database $\mathcal{C}_I$.

$$e(z) = \sum_{\iota \in I_z} \frac{|\kappa(z, \iota)|}{\sum_{\tau \in I_z} |\kappa(z, \tau)|} \cdot log_r |\kappa(z, \iota)| \quad \text{where} \quad r = |I_z| \tag{B.2}$$

Within the DÉJÀVU system, the minimization of formula (B.2) over all $z \in F_Z$ is used as a heuristic to determine the most informative observer places $O \subset F_Z$. Known a-priori probabilities $P(d)$ for the component faults d in D can be easily integrated by replacing the likelihood estimator in (B.2), yielding formula (B.3).

$$e(z) = \sum_{\iota \in I_z} \left(\sum_{d \in \kappa(z, \iota)} P(d) \right) \cdot log_r |\kappa(z, \iota)| \tag{B.3}$$

Formula (B.2) is similar to that of Forbus and de Kleer; it differs in the respect that in the simulation situation of the GDE (or GDE+) each component fault leads exactly to one symptom, and thus $\sum_{\tau \in I_z} |\kappa(z, \tau)| = |D|$.

[8] A fact which is not explicitly stated in (81).

Let $O \subset F_Z$ be the set of selected observers. Then, the database that emerges from the interval database $\mathcal{C}_I$ by eliminating all variables in $F_Z \setminus O$ is called observer database $\mathcal{C}_O$; it is much smaller than $\mathcal{C}_I$. However, the number of elements is unchanged, $|\mathcal{C}_I| = |\mathcal{C}_O|$.

Remarks. The abstraction from the real-valued simulation database $\mathcal{C}$ towards the symbolic database $\mathcal{C}_I$ provides the ground for the application of the information-theoretical considerations. In the next subsection, starting on Page 153, the true power of the model compilation idea related to the classical model-based diagnosis approaches becomes apparent: It results from the combination of behavior model contaminations and the anticipation of simulation runs.

Rule Generation The observer database $\mathcal{C}_O$ emerges from the interval database $\mathcal{C}_I$ by focusing, which in turn emerges from the simulation database $\mathcal{C}$ by abstraction. The vectors $\mathbf{c}$ in the databases undergo the modifications shown in Table B.5.

Operation	Result			
	Phase	Input	Behavior	Fault
Simulation $\Rightarrow \mathcal{C}$	$(\pi,$	$u_1, \ldots, u_{\lvert F_U \rvert},$	$z_1, \ldots, z_{\lvert F_Z \rvert},$	$d, \mu_d)$
Difference computation	$(\pi,$	$u_1, \ldots, u_{\lvert F_U \rvert},$	$\delta_1, \ldots, \delta_{\lvert F_Z \rvert},$	$d)$
Generalization	$(\pi,$		$\mathbf{I}_1, \ldots, \mathbf{I}_{\lvert F_Z \rvert},$	$d)$
Interval reduction $\Rightarrow \mathcal{C}_I$	$(\pi,$		$\iota_1, \ldots, \iota_{\lvert F_Z \rvert},$	$d)$
Elimination of dependencies	$(\pi,$		$\iota_{\widetilde{o_1}}, \ldots, \iota_{\widetilde{o_m}},$	$d)$
Selection of observers $\Rightarrow \mathcal{C}_O$	$(\pi,$		$\iota_{o_1}, \ldots, \iota_{o_k},$	$d)$

Table B.5. Modifications that the elements in the different databases undergo during the abstraction operations and the focusing operations.

The aim of the rule generation step is to extract reliable diagnosis rules from the observer database $\mathcal{C}_O$. Clearly, the rules will have a propositional-logical semantics and are of the form

$$\iota_{o_1} \wedge \ldots \wedge \iota_{o_k} \rightarrow d \quad \text{with} \quad \iota_{o_i} \in I_{o_i}, \quad d \in D, \quad \text{and} \quad k \leq |O|,$$

where $o_i \in O$; $O \subset F_Z$ is the set of the chosen observers; the symbols of a rule form a subset of a single vector $\mathbf{c} \in \mathcal{C}_O$. The left and right sides of the rule are called premise and conclusion respectively.

The semantics of such a rule $\mathbf{r}$ is defined by means of two propositional-logical truth assignment functions, $\alpha : \bigcup_{I_z,\, z \in F_Z} \rightarrow \{0,1\}$ and $\beta : D \rightarrow \{0,1\}$. For some constraint variable $z \in F_Z$, let $\mathbf{I} \in \mathcal{I}_z$ be the real-valued interval associated with the interval symbol $\iota \in I_z$, and let δ_z be a symptom. Then α and β are defined as follows.

$$\alpha(\iota) = \begin{cases} 1, & \text{If } \delta_z \in \mathbf{I} \\ 0, & \text{otherwise.} \end{cases} \qquad \beta(d) = \begin{cases} 1, & \text{If component fault } d \text{ is occurred.} \\ 0, & \text{otherwise.} \end{cases}$$

A truth assignment function α matches a rule $\mathbf{r}$ if its premise, $\mathbf{r}^-$, becomes true under α. If also the rule conclusion becomes true under a truth assignment function β, then $\mathbf{r}$ is called positive.

Note that the inference direction of the above rules is reverse to the cause-effect computations when simulating a behavior model $B'\langle F', \mathcal{M}'\rangle$: We now ask for symptoms and deduce faults, and—as opposed to the simulation situation—this inference process must not be unique. Perhaps there is a unique mapping from symptoms to faults in the original simulation database $\mathcal{C}$. However, it is very likely that the rigorously simplified observer database $\mathcal{C}_O$ encodes ambiguities. As a consequence, we may get ambiguous rules, i. e., rules with the same premise (symbolic intervals) that are associated with different faults.

To cope with this form of uncertain knowledge we forget about a strictly logical interpretation and characterize each rule $\mathbf{r}$ by its confidence, c, and its support, s:

$$c(\mathbf{r}) = \frac{h(\mathbf{r})}{h(\mathbf{r}^-)} \qquad \text{and} \qquad s(\mathbf{r}) = \frac{h(\mathbf{r})}{|\mathcal{C}_O|}$$

where $h(\mathbf{r})$ denotes the frequency of $\mathbf{r}$ in $\mathcal{C}_O$, while $h(\mathbf{r}^-)$ denotes the frequency of the rule's premise in $\mathcal{C}_O$.

The rule generation can be realized straightforwardly and is a combinatorial problem at heart. In the DÉJÀVU system, the rule generation step is realized with data mining methods, and strategies are employed with respect to confidence-thresholds and subsumption tests in order to avoid computational overhead (115).[9] Note that, as in the abstraction and focusing steps before, rule generation happens separately for each phase π, and we obtain the rule database $\mathcal{C}_R$ with phase-specific rule.

The last aspect of the presented model compilation/simplification procedure relates to the processing of the rules in $\mathcal{C}_R$. We just introduced definitions for a rule's confidence and its support but provided no operational semantics. The classics amongst the rule-based systems that employs rules with confidences is MYCIN (250, 35). However, MYCIN's formula for the confidence computation of a diagnosis is not applied in DÉJÀVU: The computation scheme used in MYCIN is designed for the accounting of a handful of rules—it fails in our setting where confidences of 10-100 rules predicting the same a diagnosis candidate $d \in D$ must be reckoned up.

Due to its successive confidence update the MYCIN formula quickly leads to confidence values close to 1, even for lower confident rules. In our situation, we know for every set of symptoms that is delivered from the observers $O \in F_Z$, which rules in $\mathcal{C}_R$ match. Clearly, a confidence computation should exploit this global view, and we developed the better suited formula (B.4). It computes for each fault $d \in D$ its confidence in "$\beta(d) = 1$"[10], when given a rule database $\mathcal{C}_R$ and a truth assignment α. The formula consists of two terms: (1) A base term, where the impact of a positive

[9] Rules of the described form are called "association rules" in the data mining jargon (6, 254).
[10] Under the single fault assumption this means $\beta(x) = 1 \Leftrightarrow x = d, x \in D$.

rule with maximum confidence cannot be weakened and, (2) , an update term, where the confidences of the positive rules are weighted with all matching rules.

$$c(\text{“}\beta(d) = 1\text{”}) \;\equiv\; c(d) = c(\mathbf{r}^*) + \big(1 - c(\mathbf{r}^*)\big) \cdot \frac{1}{|\mathcal{R}^-|} \sum_{\mathbf{r} \in \mathcal{R}} c(\mathbf{r}) \tag{B.4}$$

where $\mathcal{R}^- \subset \mathcal{C}_R$ comprises the matching rules, $\mathcal{R} \subset \mathcal{C}_R$ comprises the positive rules, and $\mathbf{r}^*$ denotes a positive rule of maximum confidence.

Remarks. The diagnosis results mentioned at the outset were achieved with automatically constructed rule databases $\mathcal{C}_R$ that have not been manually revised. The source models are medium-sized hydraulic systems consisting of 20-50 components.

How to Assess Observability and Diagnosability

This subsection continues the considerations related to the observer selection step from Page 150. The next paragraph shows, how the selection heuristic (B.2) based on (81) can be turned into an optimum placement strategy. Based on this consideration, the next but one paragraph introduces a new concept for assessing the diagnosability of a system S. The key idea is to relate the information gain of increasing sets of observers, which are optimally placed in S, to the theoretical optimum. This relation can be expressed in a concentration measure, which is called a systems discrimination entropy.

Model Compilation Enlarges the Observability Horizon The equation (150), which estimates the effort to discriminate between several diagnoses in D when using observer $z \in F_z$, has a look-ahead of 1: For each observable interval ι, discrimination must go further on, amongst the remaining set $\kappa(z, \iota) \subseteq D$ of diagnoses. A global selection strategy would determine a set of observers $O \subset F_Z$ such that the overall discrimination effort is minimum.

Within the diagnosis setting of the GDE, a global selection strategy can only be employed, if additional hypothetical simulation runs are performed: Let ι be the observation resulting from a hypothetical measurement at some observer z. Then ι is interpreted as additional system input, and for each component M of the conflict set (cf. Page 147), a simulation is performed with the reduced state prescription function $\Delta' := \Delta \setminus \delta_M$.[11] Since such a symptom-driven, hypothetical simulation concept is computationally very expensive, Forbus and de Kleer do not follow this idea. Moreover, the execution of symptom-driven simulations for diagnosis purposes in real-valued behavior models is questionable.

Within our compiled model setting, $\mathcal{C}_I$, a large database with simulation scenarios is to our disposal, which can be exploited for a global selection strategy. In this regard,

[11] This construction of Δ' reveals that the GDE is able to detect faults that were never anticipated.

we introduce the conditional probability $P_z(\iota|D)$ which specifies the probability that the symptom ι can be observed at observer $z \in F_Z$ under the condition that some fault from D has been occurred. We use the frequency distribution of D in the database $\mathcal{C}_I$ to estimate the probabilities $P_z(\iota|D)$:

$$P_z(\iota|D) = \frac{\left|\{d \in k(z,\iota) \mid d \in D\}\right|}{\sum_{\iota \in I_z} \left|\{d \in k(z,\iota) \mid d \in D\}\right|}$$

where $k(z,\iota)$ is the multiset counterpart of $\kappa(z,\iota)$; i. e., $k(z,\iota)$ is the set of diagnosis that comply with symptom "(z,ι)", and multiple occurrences of the same interval-fault combination are counted multiply. Related to the example in Table B.4, $k(q_7,\text{"}< 1.5\text{"}) = \{\boxed{a}, \boxed{a}, \boxed{b}, \boxed{c}, \boxed{d}\}$, and $P_{q_7}(\text{"}< 1.5\text{"}|\{\boxed{a}, \boxed{b}\}) = 3/4$.

Now Equation (B.2) from Page 150 can be extended to exploit a-priori knowledge about the diagnoses D amongst which the observer $z \in F_Z$ shall discriminate:

$$e(z,D) = \sum_{\iota \in I_z} P_z(\iota|D) \cdot log_r|D \cap \kappa(z,\iota)| \quad \text{where} \quad r = |I_z| \tag{B.5}$$

The minimization of Equation (B.5) over F_Z yields the most informative observer for a look-ahead of 1. By a recursive application of (B.5) to the remaining sets of diagnoses $D \cap \kappa(z,\iota)$, we can extend the observation horizon—until a unique fault classification is achieved. Each recursion step corresponds to a new observation.

Given a number of observations allowed, k, we define the discrimination effort for a system as the number of observations that must additionally be made to discriminate between all diagnoses. Clearly, this makes sense only if the k observations are optimum with respect to the expected information gain. The following definition affords the demanded; it provides a lower bound for the expected number of additional observations.

Definition B.5 (Expected Discrimination Effort) *Let S be a system that is characterized by an interval database $\mathcal{C}_I$. $\mathcal{C}_I$ defines the set of diagnoses, D, the set of possible observers, F_Z, the conditional probabilities, P_z, and the function κ. Then the expected discrimination effort of S with respect to a maximum number of observations $k > 0$ is defined as*

$$e(D,k) = \begin{cases} \min\limits_{z \in F_z} \left(\sum\limits_{\iota \in I_z} P_z(\iota|D) \cdot e(D \cap \kappa(z,\iota), k-1) \right), & \text{if } k > 0 \text{ and } |D| > 1 \\ \log_r(|D|), & \text{if } k = 0 \text{ or } |D| = 1 \end{cases}$$

where I_z comprises the intervals of an observer z, $z \in F_Z$, $r = |I_z|$, and the function κ returns for an observer z and an interval $\iota \in I_z$ the set of complying diagnoses.

When setting $k = 1$ and employing the relative frequency instead of the conditional probability, $e(D, k)$ becomes the original formula of Forbus and de Kleer (81).

Remarks. The definition of the expected discrimination effort implies several assumptions. (1) The set of diagnoses, D, is complete, (2) the diagnoses in D are equally distributed, and (3) the available observers, F_Z, are independent from each other. The presented formula uses the same resolution r for all observers but can be easily extended to allow for observer-specific resolutions r_z.

The formula for expected discrimination effort $e(D, k)$ does not explicitly exclude that an observer is used twice during the recursive descend. The following lemma closes this gap; it shows under which conditions the observer assignment is unique.

Lemma B.1 (Unique Observer Assignment) *Let $|D| > 1$, $k > 0$, and $z \in F_Z$. If D is no subset of some $\kappa(z, \iota), \iota \in I_z$ then the observer z has not been used by now during the determination of $e(D, k)$.*

Remarks. The condition of the Lemma precludes that $\kappa(z, \iota) \cap D = D$ holds for some interval ι. Given such a case, the related observer, z, cannot provide discrimination information respecting D in the interval ι, and the related discrimination effort is $P(\iota|D) \cdot \log_r(|D|)$, if no further observer is chosen.

Proof of Lemma B.1. For a given set of diagnoses, D, some $k > 0$, and a set of observers, F_Z, let $e(D, k)$ designate the expected discrimination effort.
Assume that observer z has been chosen twice within the computation of $e(D, k)$. The first time z is chosen, the current set D is split into—not necessarily disjoint—sets $D_1^{(1)}, \ldots, D_z^{(1)}$. For each of these sets the relation $D_i \subseteq \kappa(z, \iota_i)$ is fulfilled. If z is chosen a second time, it has to discriminate amongst the diagnoses of a particular $D_j \in \{D_1^{(2)}, \ldots, D_z^{(2)}\}$. In the course of observer selection the diagnosis sets either become smaller or remain unchanged, so $D_j \subseteq \kappa(z, \iota_j)$ still holds. This contradicts the condition of the Lemma.

Quantifying a System's Diagnosability There is the interesting question how to assess the difficulty to diagnose a system. In the following we will present the necessary considerations and develop such measure. Starting point is the formula for the expected discrimination effort, $e(D, k)$.

If $k = 1$ then $e(D, k) = \min_{z \in F_Z} (\sum_{\iota \in I_z} P(\iota|D) \cdot \log_r(|\kappa(z, \iota)|))$. The term $\sum_{\iota \in I_z} P(\iota|D) \cdot \log_r(|\kappa(z, \iota)|)$ becomes minimum if the diagnoses are distributed equally amongst the r intervals in I_z. This, in turn, allows us to factor out the term $\log_r(|\kappa(z, \iota)|)$, and $e(D, 1)$ simplifies to $\log_r(|\kappa(z, \iota)|) \cdot \sum_{\iota \in I_z} P(\iota|D) = \log_r(\frac{|D|}{r})$.
Repeating the same assumptions for $k = 2$ yields:

$$\begin{aligned} e(D,k) &= \min_{z \in F_Z} \left(\sum_{\iota \in I_z} P(\iota|D) \cdot e(\kappa(z,\iota),1) \right) \\ &= e(\kappa(z,\iota),1) \\ &= \log_r \left(\frac{|\kappa(z,\iota)|}{r} \right) \\ &= \log_r \left(\frac{|D|}{r^2} \right) \end{aligned}$$

Note that the minimum number of observations totally required depends on both the observers' resolution, say, their number of intervals, r, and the number of diagnoses $|D|$. The infimum number of observations necessary to discriminate between each diagnosis is $\lfloor \log_r |D| \rfloor$. It is used to specify E^*, the accumulated ideal discrimination effort of a system S as follows.

Definition B.6 (Accumulated Ideal Discrimination Effort) *The accumulated ideal discrimination effort of a system S with respect to a set of diagnoses D and an observer resolution r is defined as*

$$E^*(D) := \sum_{i=1}^{\lfloor \log_r |D| \rfloor} \log_r \frac{|D|}{r^i}$$

The difference between the accumulated expected and the accumulated ideal discrimination effort can be used as a measure for the difficulty to diagnose a system. The larger this difference is the more does a faulty system behave agnostic. Note that this measure gives an estimation that is independent of the number of possible observers, thus providing a system-specific characteristic. At the best, the difference between the expected and the ideal discrimination effort is zero. Figure B.12 illustrates the difference between the discrimination efforts pictorially; the accumulated difference is called discrimination entropy here.

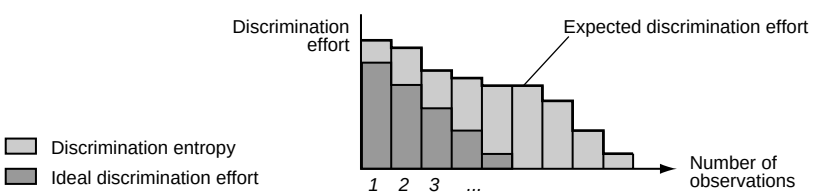


Figure B.12. Discrimination entropy: The difference between the accumulated expected and the ideal discrimination effort.

Definition B.7 (Discrimination Entropy) *The discrimination entropy E of a system S with respect to a set of diagnoses, D, is defined as*

$$E := \left(\sum_{k=1}^{\infty} e(D,k) \right) - E^*(D)$$

Synopsis

Problemclass Diagnosis of continuous technical systems.

Problem Solving Method Generation of a heuristic diagnosis model from first principles.

Source Model Discrete event/continuous time model $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda\rangle$, defined over $\langle F, \mathcal{M}\rangle$.

- *Fidelity Level F.* Fluidic, mechanical, and electrical quantities.
- *Granularity Level $\mathcal{M}$.* Fluidic and electrical components.
- *Input F_U.* Extrinsic cylinder forces and valve switching signals.
- *State Prescription Function Δ.* Explicit local differential equations; implicit nonlinear algebraic equations.
- *Output Function Λ.* Trajectories of all physical quantities.
- *Behavior Model Processing.* Differential-algebraic system solving.

Compiled Model Compiled, heuristic simplification of $B\langle F, \mathcal{M}\rangle$.

- *Fidelity Level F.* Selected and coarsened physical quantities of the source model.
- *Granularity Level $\mathcal{M}$.* Monolithic global model.
- *Input F_U.* Input quantities of the source model plus additional physical quantities that represent observed symptoms.
- *State Prescription Function Δ.* Propositional-logical rules with confidence values.
- *Output Function Λ.* None.
- *Behavior Model Processing.* Rule processing with confidence computation.

Knowledge Source Systematic simulation of both correct and faulty source models. Generalization of the simulation data; elimination of quantities by information-theoretical methods. Rule generation by data mining.

C

Model Reformulation

We designate a model construction process as reformulation, if the model is transformed from one representation into another—while leaving the model's accuracy and granularity unbiased. The reasons for a reformulation can be multifaceted: There may be requirements related to processing (the model processor does not accept the source model's form), security (the source model's internals are to be hidden), knowledge transfer (the source model's essence is needed within another application), processing properties (the reformulated model can be easier processed), model handling (the reformulated model can be easier maintained, understood, or communicated), and other reasons. See Page 52, Section 2.4, for a comparison to other model construction approaches.

Just as much as it can serve different purposes, the model reformulation process can take different forms; the case studies of this chapter give an idea of this spectrum. In Section C.1, model reformulation happens within a special electrical engineering application, that is to say, the transfer of an electrical circuit from the voltage/current domain into the wave domain. Driving force are processing advantages: The reformulated model comes up with excellent numerical properties. In Section C.2, model reformulation means to make an expert's problem-solving expertise, which is encoded in the form of an object classification, explicit in the form of a similarity measure. Here, the driving force is knowledge transfer: Similarity measures are used within various knowledge-based analysis and synthesis tasks.

C.1 Constructing Wave Digital Structures from Electrical Circuits

This section is subjected to wave digital structures, short: WDS. Wave digital structures have their origins in the field of filter design, where they are designated more specifically as wave digital filters (72, 73). They can be considered as a particular class of signal flow graphs whose signals are linear combinations of the electric current and flow, so-called a/b-waves. The translation of an electrical circuit from the electrical v/i-domain into the a/b-wave-domain establishes a paradigm shift with respect to model processing; it is bound up with profound numerical advantages. However, since neither the modeling accuracy nor its granularity is affected, the translation into a wave digital structure establishes a model reformulation.

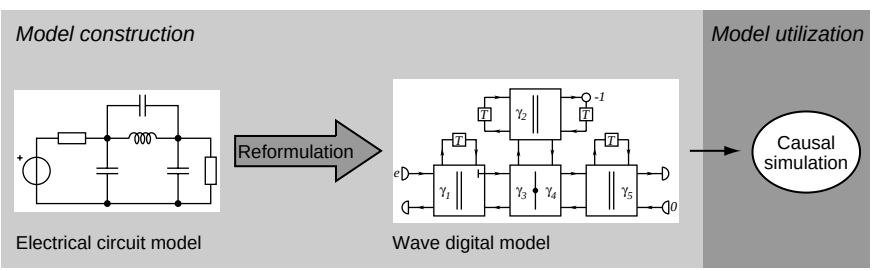


Figure C.1. Reformulation of an electrical circuit model as wave digital structure for model processing reasons.

When migrating from a voltage/current description of an electrical circuit S towards a wave digital structure, the model is completely changed: The structure model of S is interpreted as a series-parallel graph with closely connected components and transformed into an adaptor structure (cf. Figure C.1). This reformulation aims at the analysis, say, simulation of S, as illustrated at Gero's design cycle in Figure C.2.

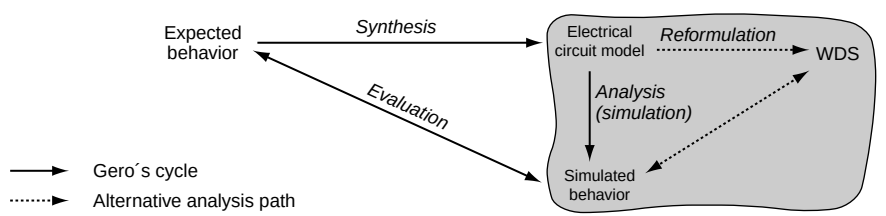


Figure C.2. An extension of Gero's widely-accepted model of the human design process (92); the automatic construction of WDS aims the analysis step (shown gray).

Note however, that the construction of a wave digital structure constitutes a syn-

thesis task at heart: The design of a sophisticated algorithmic model. Since this design task is not trivial and demands experience, its automation is a worthwhile undertaking. In this place, the necessary concepts and algorithms for a WDS design automation are prepared. In particular, we present the algorithm ADAPTORS which computes for a given electrical circuit the optimum wave digital structure in linear time. Note that this result cannot be further improved.

Underlying Models

Definition C.1 (Electrical Circuit Model) *Let S be a passive electrical circuit and let $\langle F, \mathcal{M} \rangle$ be a model of S. An electrical circuit model over $\langle F, \mathcal{M} \rangle$ is a dynamic, continuous time model, $B\langle F, \mathcal{M} \rangle = \langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$, whose elements are defined as follows.*

(1) $F = F_U \cup F_Z$ is a set of functionalities, described below. The elements in $\mathcal{M}$ are called one-ports[1], elements, or components, where capacitances and inductivities are the only reactive elements.

(2) F_U is the set of input variables, specifying voltage and current signals; F_Z is the set of constraint variables for the electrical quantities. $F_Y \subseteq F_Z$ is the set of output variables, i. e., quantities in F_Z that shall be observed.

(3) The sets U_f, U_f^T, and Z_f designate the domains of the variables f in F. Likewise, $\mathcal{U}$, $\mathcal{U}^T$, $\mathcal{Z}$, and $\mathcal{Y}$ designate the Cartesian products of the input variable domains, the constraint variable domains, and the output variable domains. The time base T is a subset of $\mathbf{R}^+$. $\mathcal{V}$ comprises the domains of all functionalities.

(4) Δ declares a set of state variables, $F_X \subseteq F_Z$, and a state space, $\mathcal{X}$, which is the projection of $\mathcal{Z}$ with respect to F_X. If S does not contain structural singularities, there is a state variable for each reactive element. Given a state vector $\mathbf{x} \in \mathcal{X}$, a vector of functions $\mathbf{u}(t) \in \mathcal{U}^T$, and some $t \in T$, Δ determines a constraint vector $\mathbf{z} \in \mathcal{Z}$ including a new state, say, $\Delta : \mathcal{X} \times \mathcal{U}^T \times T \to \mathcal{Z}$.

(5) Λ is the identity mapping.

Definition C.2 (Corresponding Graph of an Electrical Circuit) *Let S be an electrical circuit and let $\langle F, \mathcal{M} \rangle$ be a model of S. The corresponding (electrical) graph $G(S)$ of S is a structure model $\langle V, E, \sigma \rangle$ over $\langle F, \mathcal{M} \rangle$; it is defined as follows.*

(1) V is the set of segments of the interconnecting network in S that form areas of equal potential.

[1] We restrict ourselves to one-port elements here.

(2) E is a set of two-element sets $\{v, w\} \in \mathcal{P}(V)$, $|E| = \mathcal{M}$, whose elements correspond in a one-to-one manner to the components in $\mathcal{M}$. $\{v, w\}$ is in E iff the potential areas v and w are connected by the component in $\mathcal{M}$ that corresponds to $\{v, w\}$.

(3) σ is some labeling function.

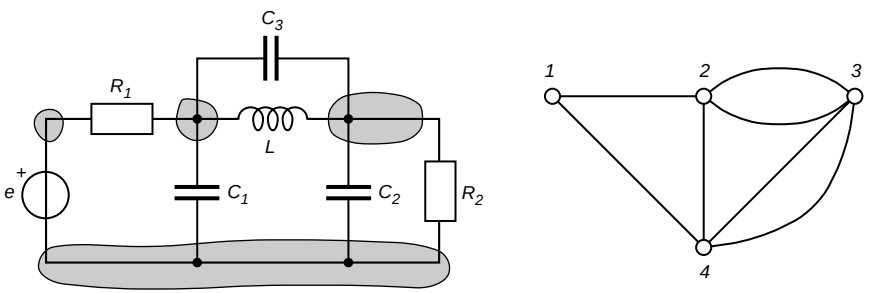


Figure C.3. Drawing of an electrical circuit with its corresponding graph (right). The shaded regions in the circuit indicate the areas of equal potential in the interconnecting network.

Remarks. Definition C.2 enables us to disburden our considerations from electrical circuits and use their graph equivalents instead.

Example. Given the drawing of an electrical circuit S depicted in Figure C.3; a corresponding graph G of S is defined on the set of points $V = \{1, 2, 3, 4\}$ and has the edge set $E = \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 4\}\}$.

Reformulating a Circuit as Wave Digital Structure

Targeted especially on readers with less background knowledge in classical network theory or signal theory, this subsection introduces the underlying ideas of wave digital structures.

Overview Let $B\langle F, \mathcal{M}\rangle$ be an electrical circuit model. Related to $B\langle F, \mathcal{M}\rangle$, an algorithmic model in the form of a wave digital structure can be created, which is a rather complex reformulation where several constraints are to be met. The reformulation involves the following principal steps.

(1) Topology reformulation of the Kirchhoff interconnecting network.

(2) Description of component behavior in the a/b-wave domain.

(3) Discretization by numerically approximating the differential equations.

Remarks. The above reformulation steps divide into local operations (Step 2 and 3), which act on the components of the electrical circuit in an isolated manner, and into the global topology reformulation in Step 1. Note that Step 2 and Step 3 are orthogonal to each other; i. e., their order of application can be interchanged.

In a nutshell, a wave digital structure is a particular kind of signal flow graph. Its topology is constructed by means of series and parallel connectors; the signals that are processed when traveling along the signal flow graph are wave quantities. In the remainder of the subsection, examples to each of the above reformulation steps will be presented and the rationale will be discussed.

Topology Reformulation Let S be an electrical circuit and G its corresponding graph. The reformulation of the Kirchhoff interconnecting network of S grounds on the identification of subgraphs in G that are either connected in series or in parallel to each other. Both series connections and parallel connections are specializations of a concept called "port", as much as each component with two terminals establishes a (one-)port as well. A port fulfills the port condition, which claims that the currents at terminal 1 and terminal 2, i_1 and i_2, fulfill the constraint $i_1 = -i_2$ at any point in time.

Objective of the topology reformulation is the replacement of series and parallel subgraphs by special connectors. They guarantee that Kirchhoff's laws are fulfilled and permit a special network analysis approach.

Common network analysis approaches are based on mesh equations, node equations, or state equations (74, 37, 286). Following a common approach means to set up and transform matrices, in a way the mesh-incidence matrix, the branch-impedance matrix, the node-incidence matrix, the branch-admittance matrix, or the state space matrix. Computations on matrices are global computations in the sense that a system of equations must be treated at the same time to find the equations' solutions. By contrast, a computation is local if a single equation at a time is sufficient to compute a solution of that equation, and if this solution is coded explicitly in the equation.

If the topology of S is realized solely by means of series and parallel connections, model processing effort can decisively be decreased: Due to the special topology, computational effort can be made up front—during model construction time—resulting in a new behavior model whose equations can be processed locally. Note that a behavior model where all equations can be processed in a local manner, e. g. by local propagation, establishes a causal behavior model (cf. Definition 2.5, Page 32). Such a behavior model represents the most efficient algorithmic model possible.

Transfer to the a/b-Wave Domain and Discretization The electrical quantities voltage, v, and current, i, can be expressed in terms of other quantities, e. g. by so-called wave quantities, a, b, which are linear combinations of v and i. The transformation pursued here is defined as follows.

$$a = v + Ri \qquad b = v - Ri \tag{C.1}$$

The wave quantities defined in the equations (C.1) are called voltage waves, where a and b represent the incident and reflected wave respectively. R is called port resistance; R must be positive, but apart from that its value can be chosen arbitrarily for each port Fettweis (73, pg. 273).

We outline now the transfer from the v/i-domain to the a/b-domain at a reactive element, the capacitance. Starting point is the following differential relationship between the current and the voltage at a capacitance, where $v(t_{k-1})$ designates the known voltage at t_{k-1}.

$$v(t_k) = v(t_{k-1}) \; + \; \frac{1}{C}\int_{t_{k-1}}^{t_k} i(\tau)d\tau \quad \text{where } t_k := t_0 + kT, k \in \mathbf{N} \tag{C.2}$$

To translate equation (C.2) into the discrete-time domain, we approximate the integral by means of the trapezoid rule.

$$v(t_k) \approx v_k := v_{k-1} \; + \; \frac{T}{2C}\big(i_k + i_{k-1}\big) \tag{C.3}$$

where v_{k-1}, i_k, and i_{k-1} denote the approximate values for the respective exact values $v(t_{k-1})$, $i(t_k)$, and $i(t_{k-1})$.

Equation (C.3) can be translated to the a/b-wave domain, for instance by using the identities (C.1). Moreover, because of the special (and simple) form of the trapezoid rule, it can directly be expressed in terms of a and b:

$$v_k - \frac{T}{2C}i_k = v_{k-1} + \frac{T}{2C}i_{k-1} \quad \Leftrightarrow \quad b_k = a_{k-1} \quad \text{with } R = \frac{T}{2C} \tag{C.4}$$

Choosing $R = \frac{T}{2C}$ as port resistance for a capacitance obviously leads to the simplest overall expression. Figure C.4 shows the capacitance in the v/i-time domain and the related wave flow diagram.

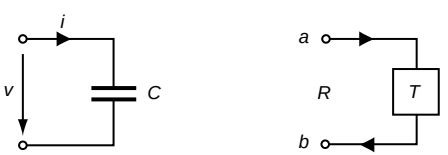


Figure C.4. Capacitance and related wave flow diagram with $R = \frac{T}{2C}$.

Remarks. (1) Equation (C.4) shows that a reformulation of the electrical quantities in terms of wave quantities is bound up with the fact that an implicit integration in v and i by means of the trapezoid method becomes explicit in a and b. (2) Note that the other electrical elements can be transfered to a/b-wave domain in a similar way (72). Figure C.5 shows the wave flow diagrams for a resistance and a resistive voltage source.

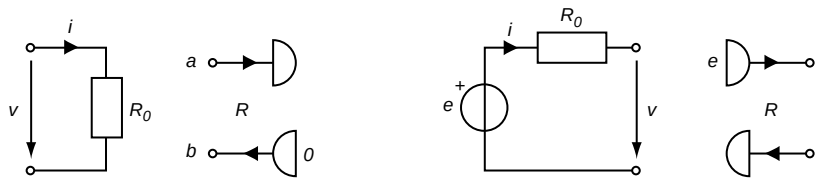


Figure C.5. Resistance (left) and resistive voltage source (right) and their related wave flow diagrams with $R = R_0$.

The Role of Adaptors If the topology reformulation of a circuit model happens in the a/b-wave domain, the connectors that are used to model series and parallel subgraphs get a special name—they are called series adaptor and parallel adaptor respectively.

Adaptors come along with ports where the a/b-equivalents of electrical components or other adaptors can be connected. An adaptor can be understood as a mathematical building block that introduces constraints on the a/b-waves of its connected elements such that in the original circuit Kirchhoff's voltage and current law are fulfilled. Clearly, these constraints depend on the elements, and they are considered in the form of the elements' port resistances that are used to compute the vector of adaptor coefficients $\boldsymbol{\gamma}$. Figure C.6 shows an electrical circuit and its related wave digital structure containing one series adaptor.

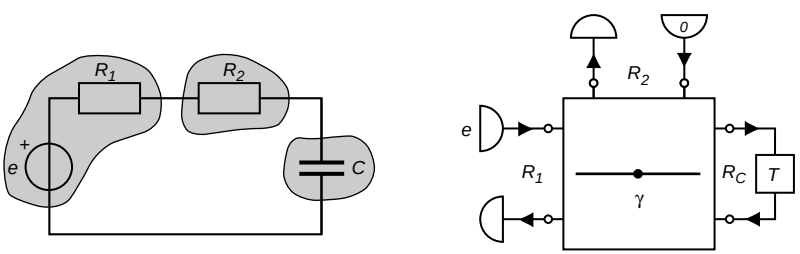


Figure C.6. Electrical circuit with two resistances and a capacity (left) and the related wave digital structure (right). The shaded areas in the circuit indicate the decomposition into three ports.

Automating the Topology Reformulation

For an electrical circuit model $B\langle F, \mathcal{M}\rangle$, the a/b-equivalents of the electrical components in $\mathcal{M}$ can be constructed straight away—a situation which does not hold for the reformulation of the circuit's interconnecting network. This subsection addresses the problem and, consequently, makes the automatic reformulation of $B\langle F, \mathcal{M}\rangle$ as a wave digital structure possible: It introduces the theoretical underpinning to generate the optimum adaptor structure for a given electrical circuit.

This adaptor structure is encoded as a special decomposition tree; the algorithm for its generation is based on the graph-theoretical concepts of connectivity, independent subnetworks, triconnected components, series-parallel graphs, and tree decomposition. The presented considerations are not restricted to electrical systems but can be transferred to mechanical, fluidic, and other—here called—effort-flow-systems.

Graph-Theoretical Concepts for Effort-Flow-Systems A coupling between two subsystems of an effort-flow-system can be represented by a pair of variables whose product is the instantaneous power being transmitted trough an energy port Wellstead (297, pg. 12). For each port these system variables divide into one intensive flow variable (current, fluid flow, velocity, etc.) and one extensive effort variable (voltage, pressure, force, etc.). Figure C.7 illustrates such a generic port.

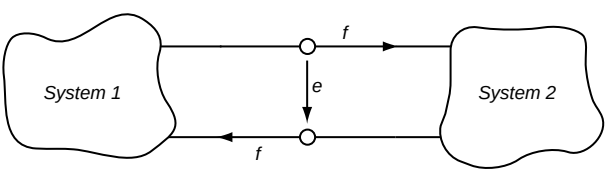


Figure C.7. Energy transmittal in effort-flow-systems is realized by means of ports each of which being characterized by an effort variable, e, and a flow variable, f.

When joining together ports, connection constraints are introduced that relate to the subsystems' effort variables and flow variables. The constraints are called compatibility and continuity constraints and are of the following form:

$$\begin{array}{lll} e = e_1 + e_2 & f = f_1 = f_2 & \text{(series connection of two ports)} \\ e = e_1 = e_2 & f = f_1 + f_2 & \text{(parallel connection of two ports)} \end{array}$$

Remarks. (1) For electrical systems, compatibility and continuity constraints are known as Kirchhoff's voltage and current law respectively. (2) Observe that both the series and the parallel connection of two ports again yields a port. This fact, along with the plain form of the connection constraints, enables one to easily combine the characteristics of the ports to a total value. And, as already pointed out before, this gives rise to an algorithm that computes the quantities for a given system S by means of local propagation.

Recall that a prerequisite for the design of a wave digital structure from the model $B\langle F, \mathcal{M}\rangle$ of a system S is the detection of the ports within $G(S)$. Clearly, if S is constructed from bottom-up by applying only series and parallel connections, $G(S)$ will be isomorphic to a series-parallel graph, and all ports can be easily found. However, typically this is not the case, and S contains "closely connected" subsystems.

A solution of this problem is described in (257) as part of a network preprocessing approach: The port concept is extended towards so-called independent subnetworks, and the relation between independent subnetworks and triconnected components is exploited to identify all ports within a flow network.[2] We will follow the same avenue here; the remainder of this paragraph presents the necessary definitions.

Definition C.3 (Two-Terminal Graph, Flow) *A two-terminal labeled graph is a triple $\langle G, s, t\rangle$, where $G = \langle V, E\rangle$ is a (multi)graph and $s, t \in V$, $s \neq t$. s and t are called source and sink of G respectively.*

A mapping $f : E \to V \times \mathbf{R}$, $f(\{v, w\}) \mapsto (u, x)$, $u \in \{v, w\}$, on a two-terminal labeled graph $\langle G, s, t\rangle$ is called flow on G if the conservation law holds for every point $v, v \neq s, t$ in G:

$$\sum_{e \in E_v} y = 0 \quad \textit{with} \quad y = \begin{cases} x & \textit{if } f(e) = (v, x) \\ -x & \textit{otherwise} \end{cases}$$

$E_v \subset E$ comprises the edges incident to v. If the function f does also depend on the parameter time, t, the conservation law must hold for any element in the domain of t.

Remarks. Standard flow definitions refer to directed graphs and a positive flow function f. In the presented definition the flow function prescribes both flow direction and flow value since we are dealing with undirected graphs. Of course, a non-positive flow function can be made positive by partially redefining it: $(\{v, w\}, (v, x))$ is replaced with $(\{v, w\}, (w, -x))$ if $x < 0$.

Ports are characterized by the property that they possess two terminals where for each point in time the related flow values are of equal amount and opposite direction. In this sense, a terminal of a port corresponds to the graph-theoretical concept of an edge. For our analysis of graphs it is necessary to extend the port concept towards so-called independent subnetworks whose terminals correspond to nodes.

Definition C.4 (Independent Subnetwork (257)) *Let $G = \langle V, E\rangle$ be a graph, and let H be a subgraph of G induced by $V_H \subset V$ with $|V_H| > 2$. A two-terminal labeled graph $\langle H, s_H, t_H\rangle$ is called independent subnetwork of G, if the following condition holds:*

[2] There, the ports are identified to reformulate a global model of a fluidic network into a new model that can be processed by local propagation. However, with respect to its runtime $\mathcal{O}(|E| \cdot |V|)$ the used detection algorithm for triconnected components is suboptimum.

(1) Every walk from a point in $V \setminus V_H$ to a point in V_H contains either s_H or t_H.

An independent subnetwork H will be called minimum, if there exists no independent subnetwork which is induced on a proper subset of V_H.

Let $\langle H, s_H, t_H \rangle$ be an independent subnetwork of a two-terminal labeled graph $\langle G, s, t \rangle$. Observe that the topology of H guarantees that an energy exchange between H and G can happen only via the nodes s_H and t_H. Moreover, Kirchhoff's node rule states the conservation of the electric current, which thus defines a flow in the sense of Definition C.3. From this conservation property follows that for each current flow on H the sum of all in-going currents at s_H equals the sum of all outgoing currents at t_H Jungnickel (126, pg. 106). Together both aspects enable us to enclose independent subnetworks with a hull, say, to investigate them in an isolated manner. An important consequence is that the concepts "port" and "independent subnetwork" can be used interchangeably (see Figure C.8).

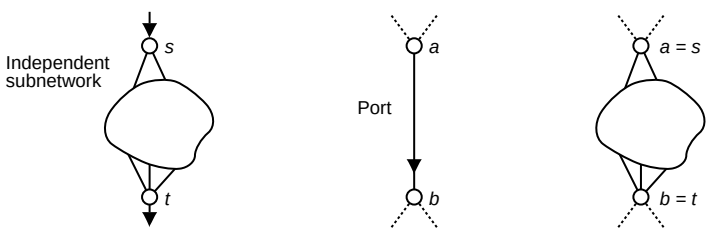


Figure C.8. Substituting an independent subnetwork (left) for a port (middle) does not violate the conservation law.

Remarks. (1) The parts of a circuit model that correspond to independent subnetworks of the circuit graph $G(S)$ have to be simulated by a global numerical procedure (257). Consequently, we are interested in a decomposition of $G(S)$ into minimum independent subnetworks. (2) Independent subnetworks are not multiports. The physical concept of a multiport can be entirely reproduced by adding to the definition of $G(S)$ a decomposition $\mathcal{C}$ of the edge set E. Each set $C \in \mathcal{C}$ stands for a subset of E and defines a multiport in a definite way.

Definition C.5 (Series-Parallel Graph (26, 31)) *Let $\langle G, s, t \rangle$ be a two-terminal labeled graph. G is called two-terminal series-parallel with source s and sink t if it can be built by means of the following three rules:*

(1) Base Graph. Any graph of the form $G = \langle \{s, t\}, \{\{s, t\}\} \rangle$ is a two-terminal series-parallel with source s and sink t.

Let $G_1 = \langle V_1, E_1 \rangle$ be two-terminal series-parallel with source s_1 and sink t_1, and let $G_2 = \langle V_2, E_2 \rangle$ be two-terminal series-parallel with source s_2 and sink t_2.

(2) Series Composition. The graph formed from G_1 and G_2 by unifying t_1 and s_2 is two-terminal series-parallel, with source s_1 and sink t_2.

(3) Parallel Composition. The graph formed from G_1 and G_2 by unifying s_1 and s_2 and unifying t_1 and t_2 is two-terminal series-parallel, with source $s_1 = s_2$ and sink $t_1 = t_2$.

Two-terminal series-parallel graphs can be represented by decomposition trees, also called sp-trees, cf. (50), which generalize the series and the parallel composition to more than two operands.

Definition C.6 (sp-Tree) *An sp-tree $T_{\langle G,s,t\rangle}$ of a two-terminal series-parallel graph $\langle G, s, t\rangle$ is a rooted tree whose nodes are either of type s-node, p-node, or leaf-node. Each node is labeled by a pair (u, v), $u, v \in V$; the children of an s-node are ordered; the leafs of $T_{\langle G,s,t\rangle}$ are of type leaf-node and correspond one-to-one to the edges of G.*

Every node of an sp-tree corresponds to a unique two-terminal series-parallel graph $\langle H, u, v\rangle$, where H is a subgraph of G and (u, v) is the label of the node. The root of $T_{\langle G,s,t\rangle}$ has label (s, t) and corresponds to the graph $\langle G, s, t\rangle$. The two-terminal series-parallel graph defined by an s-node is the result of the series composition applied to its children in their given order. The two-terminal series-parallel graph defined by a p-node is the result of the parallel composition applied to its children.

Figure C.9 exemplifies the definition.

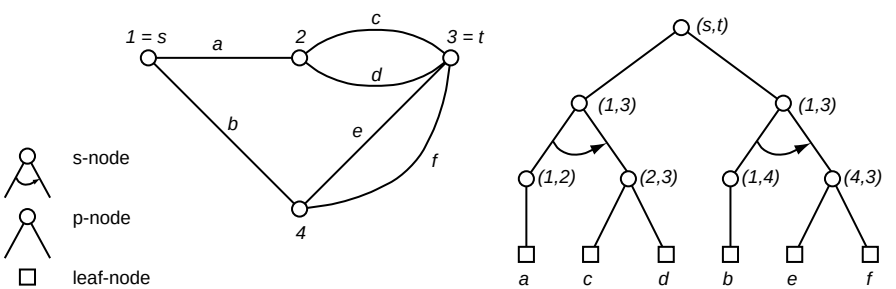


Figure C.9. Series-parallel graph (left) and its sp-tree representation (right).

The composition rules laid down in Definition C.5 make apparent that a series-parallel graph whose sp-tree has a root node label of series-node and parallel-node type has a vertex connectivity of one and two respectively. Graphs of a higher vertex connectivity are the result of either connecting more than three two-terminal graphs at the same time or by connecting two-terminal graphs by a different rule. Formally, the vertex connectivity of a graph is defined as follows.

Definition C.7 (Vertex Connectivity $\kappa(G)$) *$\kappa(G)$ is called vertex connectivity of G and is defined as follows: $\kappa(G) = min\{|T| \mid T \subset V$ and $G \setminus T$ is not connected $\}$. G is called k-connected, if $\kappa(G) \geq k$.*

Remarks. A cut point (or articulation point) of a graph G is a point $v \in V$ for which $G[V \setminus \{v\}]$ has more connected components than G.[3] A connected graph without cut points is called biconnected; a connected graph with cut points is called separable; the maximum inseparable induced subgraphs of a graph G are called biconnected components. The separation of a graph G into its biconnected components is unique (283). This fact, together with the fact that each biconnected component is analyzed on its own, we can assume without loss of generality that the considered graphs are biconnected.

The subsequent definition extends the cut point construct, it is derived from Hopcroft and Tarjan [1973].

Definition C.8 (Separation Pair) *Let $\{a, b\}$ be a pair of vertices in a biconnected multigraph G, and let the edges of G be divided into equivalence classes $E_1, \ldots, E_n$ such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an endpoint are in the same class.*

The classes E_i are called separation classes of G with respect to $\{a, b\}$. If there are at least two separation classes, then $\{a, b\}$ is a separation pair of G unless (1) there are exactly two separation classes, and one class consists of a single edge, or (2) there are exactly three classes, each consisting of a single edge.

If G is a biconnected multigraph such that no pair $\{a, b\}$ is a separation pair of G, then G is triconnected.

While the triconnectivity of a graph G follows canonically from Definition C.7 or C.8, the characterization of a graph's triconnected components is more involved. The reason for this difficulty is that triconnected components possess no property that permits their detection by a divide-and-conquer approach. Instead, it is necessary to investigate the relation of H with respect to G if a subgraph H of G forms a suspect triconnected component. Moreover, Hopcroft and Tarjan introduce different types of triconnected components, and hence the relation between H and G must be investigated relating different properties (111). Their definitions are given now.

Definition C.9 (Split Graph, Splitting, Split Component) *Let G be a multigraph with separation pair $\{a, b\}$ and related separation classes $E_1, \ldots, E_n$. Moreover, let $E' = \bigcup_{i=1}^{k} E_i$ and $E'' = \bigcup_{i=k+1}^{n} E_i$ be such that $|E'| \geq 2$, $|E''| \geq 2$, and let $G_1 = \langle V(E'), E' \cup \{(a, b)\}\rangle$, and $G_2 = \langle V(E''), E'' \cup \{(a, b)\}\rangle$. Then the graphs G_1 and G_2 are called split graphs of G with respect to $\{a, b\}$. Replacing G by two split graphs is called splitting G.*

If the split graphs are further split, in a recursive manner, until no more splits are possible, the remaining graphs are called split components of G.

Remarks. (1) The new edges $\{a, b\}$ added to G_1 and G_2 are called virtual edges; they can be labeled to identify the split. (2) If G is biconnected then any split graph of

[3] $G[V]$ denotes the subgraph of G that is induced by V.

G is also biconnected. (3) The split components of a multigraph are not necessarily unique.

The split components of a multigraph are of three types: triangles of the form $\langle\{a,b,c\},\ \{\{a,b\},\ \{a,c\},\ \{b,c\}\}\rangle$, triple bonds of the form $\langle\{a,b\},\ \{\{a,b\},\ \{a,b\},\ \{a,b\}\}\rangle$, and triconnected graphs. To obtain unique triconnected components, the split components must be partially reassembled.

Reassembling is accomplished by merging. Suppose that $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$ are two split components containing an equally labeled virtual edge $\{a,b\}$. Then the result of a merging operation is a graph G_{1+2} with node set $V_{1+2} = V_1 \cup V_2$ and edge set $E_{1+2} = E_1 \setminus \{\{a,b\}\} \cup E_2 \setminus \{\{a,b\}\}$.

Definition C.10 (Triconnected Component) *Let G be a multigraph whose split components are a set of triangles $\mathcal{S}_3$, a set of triple bonds $\mathcal{P}_3$, and a set of triconnected graphs $\mathcal{C}$. If the triangles are merged as much as possible to give a set of polygons $\mathcal{S}$, and if the triple bonds are merged as much as possible to give a set of bonds $\mathcal{P}$, then the set of graphs $\mathcal{S} \cup \mathcal{P} \cup \mathcal{C}$ forms the set of triconnected components of G.*

Remarks. (1) The triconnected components of a graph G are unique (see (284)).(2) The triconnected components in $\mathcal{S}$ are not triconnected. They establish generic series connections: Virtual edges designate the connection of a subgraph; the other edges designate single elements in S. From the viewpoint of a Kirchhoff interconnecting network the non-virtual incident edges can be replaced with a single edge of appropriate impedance. This process is called series reduction. (3) The triconnected components in $\mathcal{P}$ are defined on two points only. They establish generic parallel connections: Virtual edges designate the connection of a subgraph; the other edges designate single elements in S—from the viewpoint of a Kirchhoff interconnecting network they can be replaced with a single edge of appropriate admittance. This process is called parallel reduction. (4) The triconnected components in $\mathcal{C}$ establish minimum independent subnetworks (see (257)).

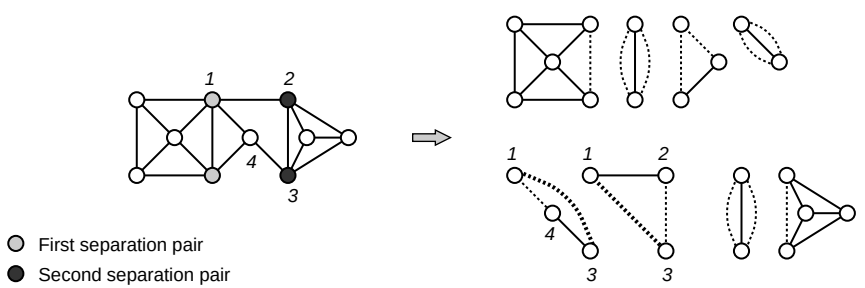


Figure C.10. A graph and its split components. When the triangles $(1,3,4)$ and $(1,2,3)$ are merged, the right hand side shows all triconnected components of the graph.

Figure C.10 shows a graph and its split components. Except the triangles $(1,3,4)$

and $(1, 2, 3)$, the split components establish triconnected components; the set of triconnected components is complete if the triangles are merged.

The algorithm presented in (111) delivers the triconnected components as defined above and runs in $\mathcal{O}(|E|)$. We will rely on it in the next subsection. The algorithm originates from Auslander and Parter's idea for an efficient planarity test (111, 14). Root of its efficiency is the statement of necessary conditions for separation pairs along with a clever computation of these conditions within several depth-first search runs.

Hopcroft and Tarjan's algorithm does not consider the semantics of independent subnetworks. As a consequence, independent subnetworks can be torn, resulting in inadmissible segmentations. Figure C.11 shows two isomorphic graphs with a different s, t-labeling. A decomposition of this graph into its split components tears the independent subnetwork with source s_2 and sink t_2.

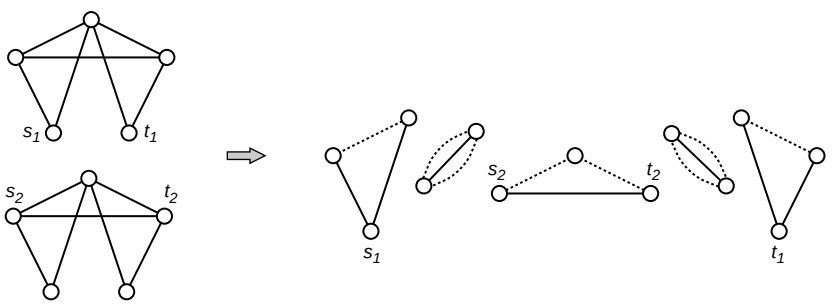


Figure C.11. Two isomorphic graphs with a different s, t-labeling (left) and the related split components (right). The independent subnetwork with the labeling s_1, t_1 is torn.

Obviously all triconnected components of a two-terminal labeled graph $\langle G, s, t\rangle$ establish independent subnetworks if s and t are nodes of the same triconnected component. The following definition and Lemma C.1 formalize this assertion.

Definition C.11 (Elementary Contraction, s-t-Contractible) *Let $\langle G, s, t\rangle$, $G = \langle V, E\rangle$, be a connected two-terminal labeled graph (not necessarily series-parallel), let $v, w \in V$, $v \neq w$, and let $V_w \subset V$ comprise the nodes adjacent to w. Then the graph $G' = \langle V', E'\rangle$ is called an elementary contraction of G respecting v, if $V' := V \setminus \{w\}$, and $E' := E \setminus \{\{w, x\} \mid x \in V_w\} \cup \{\{v, x\} \mid x \in V_w, x \neq v\}$.*

G is called s-t-contractible towards a graph $G' = \langle V', E'\rangle$, if G' is the result of a sequence of elementary contractions, and if $\{s, t\} \in E'$.

The s-t-contractibility states that the flow conservation between s and t remains intact for a two-terminal labeled graph $\langle G, s, t\rangle$. It can be ensured by simply adding the edge $\{s, t\}$ to G if s and t are not adjacent. This modification of G does not restrict its segmentation into independent subnetworks.

Lemma C.1 (s-t-Contractibility) *Let $\langle G, s, t\rangle$, $G = \langle V, E\rangle$, be a connected two-terminal labeled graph (not necessarily biconnected) with source s and sink t, and let $\{s,t\} \not\in E$. Moreover let G' be $\langle V, E \cup \{\{s,t\}\}\rangle$, and let $G'_1, \ldots, G'_m$ be the triconnected components of G'. Then the following holds:*

(1) $\exists G'_i$ which is s-t-contractible,

(2) G' can be decomposed into the same independent subnetworks like G.

Proof. Point (1). Follows immediately from the fact that there must be some graph G'_i that contains the edge $\{s,t\}$. Point (2). Observe that for an independent subnetwork $\langle G_i, a, b\rangle$ that has s (or t) amongst its nodes one of the following equations must hold: $a = s$ or $b = s$. This follows from the independent subnetwork definition. If G' cannot be decomposed into the same independent subnetworks like G then this must be on account of the edge $\{s,t\}$. It prohibits a segmentation of some G'_i into the independent subnetworks $\langle G_i, s, a\rangle$ and $\langle G_j, t, b\rangle$, which could be formed when segmenting the original graph G. Since $\langle G_i, s, a\rangle$ and $\langle G_j, t, b\rangle$ form independent subnetworks, a and b must be articulation points of G, which in turn means that $\{s,a\}$ and $\{s,b\}$ establish separation pairs in G'. Hence, an independent subnetwork G'_k can be formed that contains $\{s,t\}$ as its only non-virtual edge. Conversely, the edge $\{s,t\}$ does not prohibit the formation of independent subnetworks that can be formed in G. ◇

As outlined in the remarks on Page 171, the three types of triconnected components form the backbone for the segmentation of a circuit S: Based on the the sets $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{C}$, a connector structure, or as the case may be, an adaptor structure is easily constructed. In this connection it is useful and quite natural to extend the concept of sp-trees (Definition C.12) towards spc-trees.

Definition C.12 (spc-Tree) *An spc-tree $T_{\langle G,s,t\rangle}$ of a two-terminal (multi)graph $\langle G, s, t\rangle$ is a rooted tree whose nodes are either of type s-node, p-node, c-node, or leaf-node. A c-node is labeled by the graph $\langle V_H, E_H, u, v\rangle$ it stands for; the other nodes are labeled by a pair (u, v). The children of an s-node are ordered; the leafs of $T_{\langle G,s,t\rangle}$ are of type leaf-node and correspond one-to-one to the edges of G.*

Every node of an spc-tree corresponds to a unique two-terminal graph $\langle H, u, v\rangle$; the root of $T_{\langle G,s,t\rangle}$ corresponds to the graph $\langle G, s, t\rangle$. The two-terminal graph defined by an s-node is the result of the series composition applied to its children in their given order, and the two-terminal graph defined by a p-node is the result of the parallel composition applied to its children. The two-terminal graph defined by a c-node is triconnected, has more than three nodes, and follows no construction rule.

The spc-tree $T_{\langle G,s,t\rangle}$, $T_{\langle G,s,t\rangle} = \langle V_T, E_T\rangle$ is easily constructed. $V_T = \{1, \ldots, n + |E_G|\}$ where n denotes the number of triconnected components and $|E_G|$ denotes the number of edges in G; the nodes in $\{1, \ldots, n\}$ correspond one-to-one to the triconnected components and are labeled respecting the triconnected component's type as

S-node, P-node, and C-node respectively. E_T contains an edge $\{v, w\}$ if and only if one of the following conditions is fulfilled: (1) v and w correspond to triconnected components and have a common virtual edge, (2) v corresponds to a triconnected component and w is an edge in v.

Remarks. Since both the series adaptor and the parallel adaptor are realized as three-port adaptors, the nodes of the spc-tree that are labeled as P-node or S-node may be expanded again to account for their restricted number of ports. Moreover, observe that the height of the decomposition tree defines the longest propagation path of the adaptor structure. Consequently the root of the decomposition tree should be defined as some node leading to a minimum tree height. The subsequent definition picks up both aspects and introduces a normalized spc-tree.

Definition C.13 (Normalized spc-Tree) *Let $T_{\langle G,s,t\rangle} = \langle V_T, E_T\rangle$ be an spc-tree. $T_{\langle G,s,t\rangle}$ is called normalized spc-tree if each node $v \in V_T$ labeled S or P has at most two successors, and if the root v of $T_{\langle G,s,t\rangle}$ represents a center of $T_{\langle G,s,t\rangle}$ and has a degree larger than* 1.

Remarks. (1) $T_{\langle G,s,t\rangle}$ is normalized by replacing each node $v \in V_T$ labeled S or P that has more than two successors with the root of a balanced binary tree, T_v, whose leafs are the successors of v; the inner nodes of T_v get the same label as v. (2) The center of a tree $T = \langle V, E\rangle$ can be computed in $\mathcal{O}(|V|)$ (186).

The Algorithm ADAPTORS The previous paragraph provides the theoretical underpinning for the following adaptor synthesis algorithm.

ADAPTORS
Input. An electrical circuit model $B\langle F, \mathcal{M}\rangle$ of a system S.
Output. A normalized spc-tree defining the optimum adaptor scheme and adaptor types, the port resistances, and the adaptor coefficients.

(1) Generate the corresponding graph G of S.
(2) Partition G respecting its biconnected components $\mathcal{G} = \{G_1, \ldots, G_m\}$.
(3) $\forall G' \in \mathcal{G}$ **do**
(4) Check G' for inadmissible segmentation.
(5) Detect triconnected components in G'.
(6) Construct an spc-tree for G'.
(7) **end**
(8) Construct an spc-tree $T_{\langle G,s,t\rangle}$ for the entire graph G.
(9) Normalize $T_{\langle G,s,t\rangle}$.
(10) Compute the port resistances and adaptor coefficients.

Theorem C.1 *Given an electrical circuit model $B\langle F, \mathcal{M}\rangle$ containing $|\mathcal{M}| = n$ elements. Then* ADAPTORS *computes a normalized spc-tree defining the optimum adaptor scheme and types, the port resistances, and the adaptor coefficients in $\mathcal{O}(n)$.*

Proof. The runtime bounds for the steps in line 1–5 follow from the considerations and algorithms pointed out in the previous subsection. The connection of the forest of the m spc-trees, $m < n$, Line 8, is linear. Finally, the adaptor computations, Line 10, involve only a constant number of operations for each of the n elements (see the previous subsection and especially (72, 73)).

In the sequel, steps of ADAPTORS are illustrated at the sample graph of Figure C.12, which establishes the corresponding graph G of some electrical circuit S.

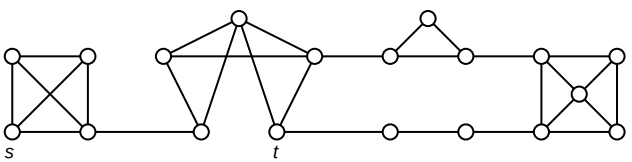


Figure C.12. Corresponding graph G of some electrical circuit S.

Line 2. Partition G respecting its biconnected components, $G_1, \ldots, G_m$. Label the articulation points of the G_i by s_i or t_i, such that each biconnected component contains a source s_i and a sink t_i (see Figure C.13).

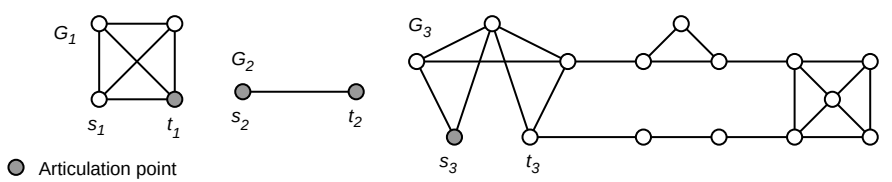


Figure C.13. Decomposition of G respecting its biconnected components and relabeling of the biconnected as two-terminal graphs.

Line 4. Check for inadmissible segmentations. In the sample graph an edge $\{s_2, t_2\}$ is introduced. However, this step is superfluous for electrical circuit models if s and t are incident to the signal source.

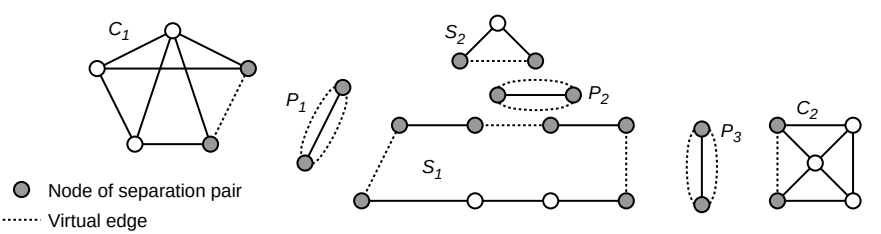


Figure C.14. Detection of the triconnected components in G_3.

Line 5. Detect in G_3 the three sets of different triconnected components, $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{C}$. Figure C.14 shows the result.

Line 6. Construct an spc-tree $T_{\langle G_3, s_3, t_3\rangle}$ for G_3; the Figure C.15 shows the result. At this place the previously mentioned series reductions and parallel reductions are ideally performed: Engineering knowledge on useful reductions can be formulated by means of simple contraction rules, which may even investigate the context of an element.

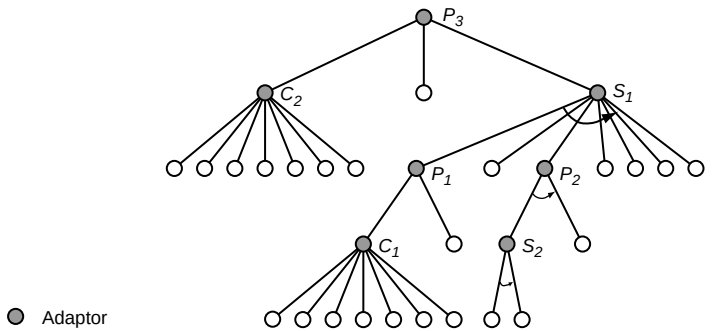


Figure C.15. An unnormalized spc-tree of G_3; the leafs of the tree correspond to the edges in G_3.

Line 8. Construct an spc-tree $T_{\langle G,s,t\rangle}$ for the entire graph G. This accomplished by connecting the roots of the trees $T_{\langle G_i, s_i, t_i\rangle}$ with a new node that is labeled as an S-node.

Line 9. Normalize the decomposition tree $T_{\langle G,s,t\rangle}$; the Figure C.16 shows the result. Obviously, the ideal adaptor for having no reflection-free port is associated with the root of the normalized the decomposition tree.

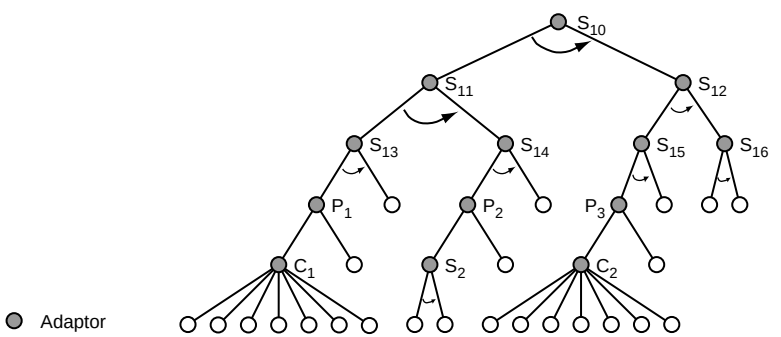


Figure C.16. The normalized counterpart of the spc-tree from Figure C.15.

Line 10. Based on the component parameters in F, $F \in B\langle F, \mathcal{M}\rangle$, compute the port resistances for the adaptors.

Remarks. The algorithm ADAPTORS can be extended with respect to multiports. To this end each subgraph that is induced by a multiport is completed such that it forms a clique.

Design Generation by Graph Grammars The transformation of an spc-tree into a corresponding adaptor structure can be specified by a design graph grammar that operationalizes the following tasks: (1) Normalization of the spc-tree by splitting s-nodes and p-nodes with more than three edges, (2) replacement of c-nodes with a special two-port adaptor structure, (3) replacement or comprisal of particular element combinations, and (4) generation of the layout and the domain-typic appearance. Figure C.17 hints some of the transformation tasks.

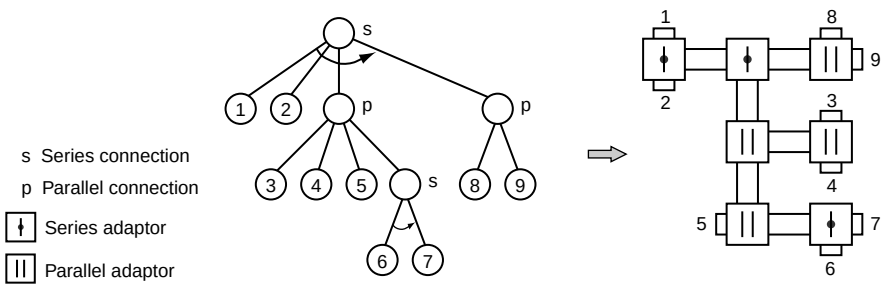
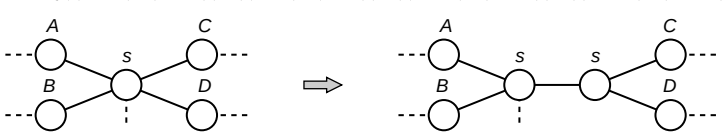


Figure C.17. Transformation of an spc-tree into an adaptor structure.

The grammar $\mathcal{G} = \langle \Sigma, P \rangle$ performs the transformation task (1) where $\Sigma = \{p, s, A, \ldots, H\}$ and P contains two splitting rules (for s-nodes shown below) of the form $T \rightarrow \langle R, I \rangle$:

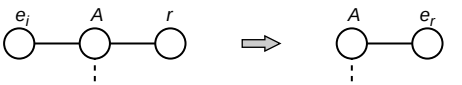
$$
\begin{aligned}
(1)\quad T &= \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3,4,5\}, \{\{1,5\},\{2,5\},\{3,5\},\{4,5\}\}, \\
&\qquad \{(1,A),(2,B),(3,C),(4,D),(5,s)\}\rangle \\
R &= \langle V_R, E_R, \sigma_R \rangle = \langle \{6,7,8,9,10,11\}, \{\{6,10\},\{7,10\},\{8,11\},\{9,11\}, \\
&\qquad \{10,11\}\}, \{(6,A),(7,B),(8,C),(9,D),(10,s),(11,s)\}\rangle \\
I &= \{((E,A),(E,A)),((F,B),(F,B)),((G,C),(G,C)),((H,D),(H,D))\}
\end{aligned}
$$



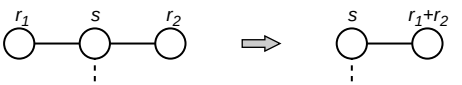
With respect to transformation task (2) we refer to a paper of Meerkötter and Fränken (178). They present a generic construction rule for adaptor structures if the graph $G(S)$ of an electrical system S contains closely connected subgraphs. Due to the complexity of this transformation we abstain from a presentation of the respective design graph grammar in this place. For the same reason the graph grammar rules in the sequel, relating the transformation tasks (3) and (4) are only specified graphically.

Transformation task (3) contains design optimization potential. According to engineering know-how and design experience, new elements can be substituted for particular element combinations, or elements of the same type can be comprised—examples:

(3a) Introduction of resistive sources.



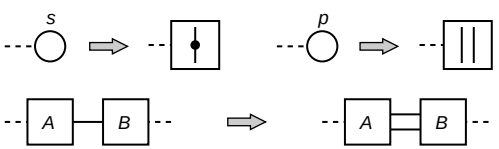
(3b) Comprisal of series resistances.



(3c) Comprisal of parallel resistances.

The figures below show some display rules that change the appearance of an spc-tree into an adaptor structure. The interesting point in this connection is that such rules fit seamlessly in Brandenburg's layout graph grammar approach.

(4a) Adaptor display rules.



(4b) Port display rules.

Because of the regular structure of wave digital filters, layout graph grammars are an adequate means for the drawing task in hand. They perform a syntax-directed translation of textual representations of graphs into graph drawings (30). A design graph grammar becomes a layout graph grammar by attribution, where the attributes describe geometric relations. Details and successful applications of this approach can also be found in (98, 29, 311).

Synopsis

Problemclass Analysis; specifically simulation of passive, electrical continuous time systems.

Problem Solving Method Integration of differential-algebraic equations.

Source Model Dynamic, continuous time model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$.

- *Fidelity Level F.* Electrical quantities.
- *Granularity Level $\mathcal{M}$.* Electrical elements that form one-port elements.
- *Input F_U.* Voltage and current signals.
- *State Prescription Function Δ.* Implicit state space description in the form of local, but non-causal differential-algebraic equations.
- *Output Function Λ.* Courses of selected quantities in F.
- *Behavior Model Processing.* Implicit integration method.

Reformulated Model Dynamic, continuous time model $\langle F_U, F_Z, F_Y, \mathcal{V}, \Delta, \Lambda \rangle$.

- *Fidelity Level F.* Voltage-wave quantities.
- *Granularity Level $\mathcal{M}$.* a/b-equivalents of the electrical elements and adaptor building blocks.
- *Input F_U.* Wave signals.
- *State Prescription Function Δ.* Causal difference equations.
- *Output Function Λ.* Courses of selected quantities in F.
- *Behavior Model Processing.* Local propagation.

Knowledge Source Topological analysis of the source model's Kirchhoff interconnecting network. Comprisal and replacement rules for elements in $\mathcal{M}$.

C.2 Learning Similarity Measures from Object Classifications

The section addresses a key aspect within various knowledge-based analysis and synthesis tasks: The construction of a measure that adequately models the similarity between two problem instances. This may be the similarity between two documents within a document retrieval task, the similarity between two cases within a case-based reasoning task, or a similarity assessment between two points in a graph when working on a visualization task.

Given two problem instances, a domain expert is in a position to assess the similarity between these instances with respect to a problem solving task in hand. It is a question of high importance how this part of an expert's problem-solving expertise can be elicited and made explicit.

In its general form, a set of objects (the problem instances), O, is given, where each object $x \in O$ is described by a vector of features or demands, $d(x) = (f_1, \ldots, f_p)$. The similarity between two objects x and y, $x, y \in O$, is taken to assess the usability of a solution of instance x as a solution for instance y—an idea that became popular under the name of case-based reasoning, CBR (1, 156). Usability can be stated a-posteriori only while the similarity between two objects can be stated immediately (196). The quantification of the concept "usability" by means of the similarity between two feature vectors shows the crucial importance that comes up to the computation of the features.

In the following, the similarity between two objects x and y is designated by a relation "*sim*" where $sim(x, y)$ determines a value from the interval $[0; 1]$. The larger is the value of *sim* the more similar are x and y to each other.

Developing "*sim*" is a Knowledge Acquisition Problem A recurring situation in, for instance, case-based problem solving is that a set of cases is given, but no similarity measure can ad-hoc be stated. A similarity measure establishes a particular form of knowledge, which—using AI terminology—can be acquainted from some source. An often applied concept to acquire similarity knowledge is the interview of domain experts: "Are these two problem instances, x and y, similar?" "What are the significant features that make x and y similar?" "To which extent are x and y similar?"

The sample questions make problems of the concept "knowledge acquisition by questioning" obvious. On the one hand, it is hard for the domain expert to give quantifiable answers while, on the other hand, it is hard for the knowledge engineer to access the quality of these answers. This is not surprising since knowledge about object similarities establishes problem-solving expertise, which—in itself—is hard to become operationalized, as already pointed out by Hayes-Roth et al.:

> *"The transfer and the transformation of problem-solving expertise from a knowledge-source to a program is the heart of the expert-system development process."*
>
> Hayes-Roth, Waterman, and Lenat, 1983, pg. 23

In this section we exploit the fact that a similarity measure can also be constructed from other knowledge sources, for instance from the knowledge that is encoded within an existing object classification (266). Obviously, each classification implies knowledge about feature relevance and feature similarity with respect to the classified objects. Given such a knowledge source, methods from the field of machine learning can be used to transform implicit knowledge on object similarities into an explicit similarity measure (see Figure C.18).

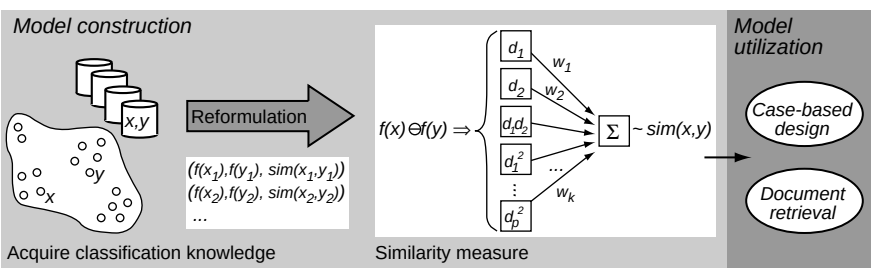


Figure C.18. Instead of interviewing domain experts, a similarity measure is constructed from an existing classification of the objects.

We view this process as a kind of model reformulation (see Page 52) since there is a paradigm shift in model processing: The features of a model are no longer used for some (absolute) classification process; instead, two model instances are taken and their relative distance in the model space is assessed. This reformulation happens, ideally, without altering the model's accuracy or its level of granularity. Also note that the processing efficieny is not affected.

Remarks. The development of a similarity measure is highly domain-dependent, and we fall back on the domain of fluidic engineering from which realistic models and tasks are derived. In fact, the automatic generation of a similarity measure for fluidic engineering tasks should be seen as a completion of our approach to the automation of fluidic circuit design in Section A.1. There, the similarity measure was developed in close collaboration with human designers.

Underlying Model and Tasks

Similarity measures can be used for those tasks in fluidic engineering that are not treated at a deep, physical level of behavior but at the much more abstract level of function. At this level, the complex physics of a fluidic circuit is reduced to a set of features which characterizes the circuit's usability to fulfill a desired function. The following list outlines tasks that are solved at an abstract functional level.

- *Functional Analysis.* Check whether two fluidic systems are similar with respect to their intended operation (288).
- *Fluidic System Design.* Construct a new fluidic system by coupling together already designed units (fluidic axes) from different systems (258, 110).
- *Document Retrieval.* Query a database for diagrams of fluidic systems that are similar with respect to a given set of demands.

The model of fluidic function as specified in Definition C.14 establishes the knowledge level at which the mentioned tasks are solved. The next subsection shows in which way this functional model is encoded as a feature vector for fluidic circuit objects.

Taken an engineer's point of view, the gist of a model of fluidic function consists of a set of state variables, F_X, along with the discrete state prescription function, Δ. Each state variable in F_X represents the function of a fluidic axis; Δ characterizes the behavior of the fluidic axes by means of the working phases of the output units, which are cylinders in most cases.

Definition C.14 (Model of Fluidic Function) *Let S be a fluidic system and let $\langle F, \mathcal{M} \rangle$ be a model of S. A model of fluidic function over $\langle F, \mathcal{M} \rangle$ is a discrete event model, $B\langle F, \mathcal{M} \rangle = \langle F_U, F_Z, \mathcal{V}, \Delta \rangle$ whose elements are defined as follows.*

(1) $F = F_U \cup F_Z$ is a set of functionalities, described below. The elements $M \in \mathcal{M}$ are called fluidic axes.

(2) F_U is the set of input variables, defining the extrinsic forces and events. F_Z is the set of constraint variables, defining the working phases of the axes and the velocities of the cylinder pistons.

(3) The sets U_f, U_f^T, and Z_f designate the domains of the variables f in F_U and F_Z. Likewise, $\mathcal{U}$, $\mathcal{U}^T$, and $\mathcal{Z}$ designate the Cartesian products of the input variable domains and the constraint variable domains. The time base T is a subset of $\mathbf{R}^+$. $\mathcal{V}$ comprises the domains of all functionalities.

(4) Δ declares a set of state variables, $F_X \subseteq F_Z$, and a state space, $\mathcal{X}$, which is the projection of $\mathcal{Z}$ with respect to F_X. The state variables correspond one-to-one to the fluidic axes in $\mathcal{M}$; the number of states in $\mathcal{X}$ is finite. Δ specifies the

discrete phase transitions of a model of fluidic function and comprises both an external and an internal state prescription function, say, $\Delta = \Delta_e \cup \Delta_i$.

Given a vector of external events (triggered by an operator or a numerical control), Δ_e prescribes the next state, depending on the current state and the vector of input forces at the current point in time. In contrast, Δ_i schedules the next state change that is triggered by proximity switches or piston stops, so-called internal events. Δ_i depends on the current state, the vector of input force functions, and the time elapsed since the last state change.

Remarks. For the purposes of this section it is not necessary to engage into state prescription functions for discrete event models and their simulation. The interested reader may refer to (310, 206), for instance.

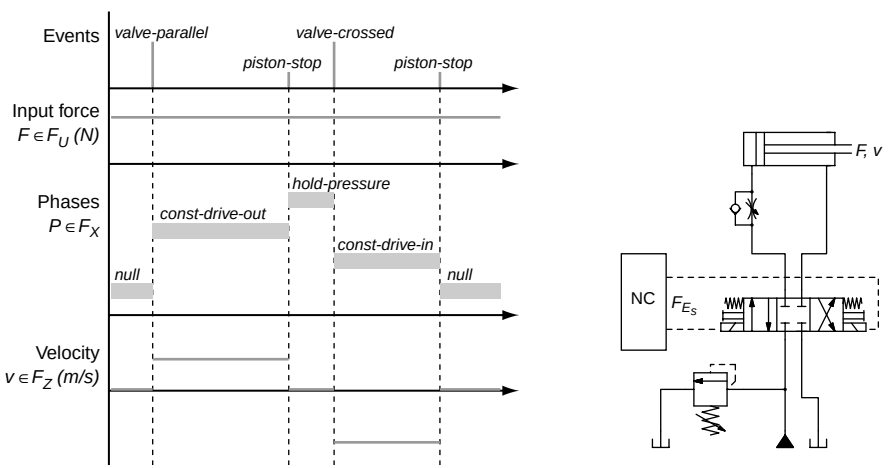


Figure C.19. The diagrams on the left-hand side describes a possible functional model of the circuit on the right.

Example. The example specifies the functional model of a fluidic system with a single axis whose cylinder drives out, performs a press job, and drives in. Figure C.19 shows for a constant extrinsic force, F, and the switch events triggered by a numerical control, E_s, both the respective diagrams and the circuit. The state space $\mathcal{X}$, constructed over the single state variable P, consists of four states which are called (working) phases.

(1,2) *Model.*
$F_U = \{F, E_s\},\ F_Z = \{P, v\},\ F_X \subseteq F_Z = \{P\},$
$F = F_U \cup F_Z,\ \mathcal{M} = \{\{F, E_s, P, v\}\}$

(3) *Domains.*
$U_F = \mathbf{R}^+,\ U_F^T = U_F$ since $F(t)$ is constant.

$U_{E_s} = \{$*valve-parallel*, *valve-crossed*$\}$,
$X_P = \{$*const-drive-in*, *const-drive-out*, *hold-pressure*, *null*$\}$, $Z_v = \mathbf{R}$

(4) *State Prescription.*
$\Delta_e : X_P \times U_F \times U_{E_s} \rightarrow \mathcal{Z}$,
$\Delta_i : X_P \times U_F \times T \rightarrow \mathcal{Z}$

The function Δ_e prescribes the next phase, which is started immediately on an external event. The function Δ_i *schedules* the next phase; i. e., it prescribes the next phase, which will be started not until the internal event *piston-stop* is occurred. Table C.1 shows a simplified definition of Δ where the elapsed time til the *piston-stop* events has been omitted.

Current phase	Force	Event	Next phase	Velocity
null	$500N$	*valve-parallel*	*const-drive-out*	$0.5m/s$
const-drive-out	$500N$	*piston-stop*	*hold-pressure*	$0m/s$
const-drive-out	$500N$	*valve-crossed*	*const-drive-in*	$-0.8m/s$
hold-pressure	$500N$	*valve-crossed*	*const-drive-in*	$-0.8m/s$
const-drive-in	$500N$	*piston-stop*	*null*	$0m/s$
const-drive-in	$500N$	*valve-parallel*	*const-drive-out*	$0.5m/s$

Table C.1. Definition of Δ for the example. The specification is simplified insofar that Δ does not schedule the internal *piston-stop* events at definite points in time.

Remarks. The abstraction from a physical behavior model in continuous time towards an event-based functional model can be done automatically. For this, the state prescription function of the continuous time model is simulated and investigated with respect to intervals of stationary flows or velocities (115). However, the development of such an abstraction is not discussed in this place.

Similarity Measures

Our objective is the development of a similarity measure for models of fluidic function to solve the engineering tasks mentioned at the outset. Our approach is the application of learning methods; they shall bridge the gap between implicit expert knowledge about circuit similarities and explicit similarity functions. This subsection develops the necessary concepts.

On Similarity Measures in General Much work has been done in the last decades on similarity measures; good overviews can be found in (224, 69, 123, 298, 219). A common similarity measure is the simple weighted linear similarity function. Let $(f_1^{(x)}, \ldots, f_p^{(x)})$, $(f_1^{(y)}, \ldots, f_p^{(y)})$, $p \in \mathbf{N}$, be two feature vectors of two objects x, y. Then the simple weighted linear similarity measure between x and y is defined as follows.

$$sim(x, y) = w_0 + \sum_{1 \leq i \leq p} w_i \cdot (f_i^{(x)} \ominus f_i^{(y)}), \quad w_0, w_i \in \mathbf{R}$$

It is often claimed that *sim* is a symmetric function which maps onto $[0; 1]$ and that it provides the reflexivity property, say, $sim(x, y) = 1 \Leftrightarrow x = y$ (see Definition A.3, Page 99). The definition of the operator "$\ominus$" depends on the features' types:

- *Cardinal.* A feature is called cardinal if and only if all values of the feature are real numbers. Values of cardinal features can be added, subtracted, multiplied, and divided and the result is still a reasonable value for the feature. For cardinal features, $f^{(x)}$, $f^{(y)}$, the following definition is often used:

$$f^{(x)} \ominus f^{(y)} = -|f^{(x)} - f^{(y)}|$$

- *Nominal.* Nominal values can only be compared with respect to equality. If a nominal feature has only two possible values it is called a binary feature. For nominal features, $f^{(x)}$, $f^{(y)}$, the following definition is often used (301):

$$f^{(x)} \ominus f^{(y)} = \begin{cases} 1, & \text{if } f^{(x)} = f^{(y)} \\ 0, & \text{otherwise} \end{cases}$$

Remarks. Note that learning such functions means to find adequate values for the parameters w_i. More complex distance functions have been examined by the authors in (266).

A Similarity Measure for Fluidic Systems Following Definition C.14, each hydraulic system S can be reduced to a functional model $B\langle F, \mathcal{M}\rangle$. In accordance with (269, 110, 288), the similarity between two hydraulic systems, S_x, S_y, is defined using the similarities between individual axes in the respective functional models $B\langle F_x, \mathcal{M}_x\rangle$ and $B\langle F_y, \mathcal{M}_y\rangle$:

$$sim(S_x, S_y) = \sum_{M_x \in \mathcal{M}_x} \max\{sim_{axes}(M_x, M_y) \mid M_y \in \mathcal{M}_y\} \tag{C.5}$$

where $\mathcal{M}_x$ and $\mathcal{M}_y$ denote the axes of S_x and S_y, sim_{axes} denotes a function measuring the similarity between two axes, and $|\mathcal{M}_x| \leq |\mathcal{M}_y|$ holds.

Let an axis M_x be described by a p-dimensional vector of cardinal features $d(M_x)$, say, $d(M_x) = (f_1^{(x)}, \ldots, f_p^{(x)}) \in \mathbf{R}^p$. Then the similarity measure $sim_{axes}(M_x, M_y)$ shall be defined as a simple weighted linear similarity:

$$sim_{axes}(M_x, M_y) = w_0 + \sum_{1 \leq i \leq p} w_i \cdot |f_i^{(x)} - f_i^{(y)}|, \quad w_0, w_i \in \mathbf{R} \tag{C.6}$$

The feature vector of a fluidic axes, $d(M)$, which is necessary to compute the similarity between two fluidic circuits, S_x, S_y, is directly extracted from the functional models $B\langle F_x, \mathcal{M}_x \rangle$ and $B\langle F_y, \mathcal{M}_y \rangle$. Each axis is described by two types of features, phase descriptions and phase orders, both of which are defined below.

(1) *Phase Description.* Phases are divided into the categories $K = \{$*constant-drive, position-drive, hold-position, accelerate, fast-drive, hold-pressure, press*$\}$, $|K| = 7$. Each category $\kappa \in K$ is characterized by 5 features, which answer the following questions.

   a) Which maximum force (in Newton) is applied to the working element?

   b) How many phases of the specific category exist in the respective axis?

   c) How long (in seconds) is the duration of the phase ?

   d) Which distance (in mm) is covered by the working element?

   e) How precisely must the axis work? This is a value from $[0; 1]$ that defines the acceptable deviations from the duration, the distance, and the force.

(2) *Phase Sequence.* For each combination $(\kappa_1, \kappa_2), \kappa_1, \kappa_2 \in K, \kappa_1 \neq \kappa_2$, a feature is deployed that specifies the number of times a phase of category κ_1 is immediately followed by a phase of category κ_2.

Together, the feature vector—so to speak, demand vector $d(M)$ for an axis M is organized as follows.

$d(M) = \big($(phase description *"constant-drive"*), (phase description *"position-drive"*),
(phase description *"hold-position"*), (phase description *"accelerate"*),
(phase description *"fast-drive"*), (phase description *"hold-pressure"*),
(phase description *"press"*), (phase sequence)$\big)$

Methods for Constructing Similarity Measures Existing methods for constructing similarity measures can be divided into two main classes: (1) Methods that employ reinforcement-learning and (2) algorithms that rely on statistical analysis for the main part.

Reinforcement-learning methods predict a similarity value and ask the user or a different system to rate the prediction. Based on this rating the weights w_i are adjusted. Statistical methods analyze given examples and deduce appropriate weights.

Name	Type	Remarks	Literature
EACH	reinforcement learning	extra parameters needed	(233)
RELIEF	reinforcement learning	binary weights	(136)
CCF	statistical	only binary features	(45)
GM-CDW	statistical		(113)

Table C.2. Selected existing methods for the construction of similarity measures.

Table C.2 lists representatives of well known methods; more examples can be found in (25, 8, 255).

These methods have in common that the knowledge acquisition step (how to obtain the necessary knowledge from an expert) and the learning step (finding appropriate values for the weights w_i) are not treated separately. Our approach, which is described in the next section, differentiates between these two steps.

Combining the knowledge acquisition step and the learning step entails several problems:

- Since the expert is integrated into such methods in a predefined manner, no flexibility is granted in the way the knowledge is obtained. Hence additional knowledge sources cannot be tapped.
- Although the methods mentioned in Table C.2 rely on standard learning paradigms such as reinforcement learning, they do not apply standard learning algorithms such as regression or neural networks but employ proprietary algorithms. While for standard learning algorithms advantages and disadvantages have been examined, almost nothing is known about the proprietary algorithms.
- Verifying a combined method is difficult since learning problems cannot be distinguished from knowledge acquisition problems.

A Universal Strategy Take again a look at Figure C.18 on Page 181. It hints the two steps of the universal strategy that is employed here to construct a similarity measure: A knowledge acquisition step and a reformulation (learning) step. This separation allows for the usage of both tailored acquisition methods and tailored learning methods to obtain the necessary information from an expert. The first step always results in a database, $\mathcal{C}$, of feature-vector pairs whose similarity is already known:

$$\begin{aligned}\mathcal{C} = \{&(d(x_1), d(y_1), sim(x_1, y_1)),\\ &(d(x_2), d(y_2), sim(x_2, y_2)),\\ &(d(x_3), d(y_3), sim(x_3, y_3)), \ldots\}\end{aligned} \tag{C.7}$$

The reformulation step uses the rated vector pairs and applies a supervised learning strategy to find values for the weights w_i. For our applications both regression

and neural networks have been applied. Note that only a small but typical set of objects, the learning set, is used for learning purposes.

Reformulation Strikes a New Path to Acquisition

The similarity between two hydraulic systems, S_x, S_y, has been reduced to a maximization of the cumulated similarities between fluidic axes, sim_{axes}, used in Equation (C.5) on Page 185. The specification of values for this function poses a knowledge acquisition problem, because domain experts have the necessary knowledge only implicitly. To express their understanding about axes similarities explicitly, e. g. as a mathematical function, means asking to much for most experts. The reformulation of similarity knowledge that is encoded implicitly in a classification shows a way out.[4]

Knowledge Source 1: Partitioning the Set of Objects For this method the expert has to partition a given set of axes. Two axes are similar if and only if they belong to the same class. Let $\mathcal{M} = \{M_1, \ldots, M_n\}$ be the set of axes and let $c : \mathcal{M} \rightarrow C$ be the classification function, where C comprises the set of possible classes. The classification function c is specified by the domain expert. Then the similarity sim_{axes} is defined as follows.

$$\forall\, M_x, M_y, \in \mathcal{M} :\ sim_{axes}(M_x, M_y) = \begin{cases} 1, & \text{if } c(M_x) = c(M_y) \\ 0, & \text{otherwise} \end{cases} \tag{C.8}$$

Reasonable classes, for example, are $C = \{manipulation, press, hold\}$ or $C = \{high\text{-}pressure, low\text{-}pressure, fast\text{-}drive\}$.

The main disadvantage bound up with this knowledge source is that a partitioning, say, a disjunctive classification, is sometimes difficult to be stated. The advantages of this knowledge source are:

- n classifications define $\frac{n^2}{2}$ similarities.
- Domain experts have few problems in classifying fluidic axes.

Although in the learning set only the similarity values 0 and 1 are given, learning algorithms like regression result in similarity measures that can yield any similarity value from the interval $[0; 1]$. This is because the given data is abstracted by the learning algorithms.

[4] This is in accordance with Richter's argumentation: Knowledge (on similarity) is distributed over the containers vocabulary, similarity measure, case base, and solution transformation, where each container is able to contain all available knowledge (223).

Knowledge Source 2: Graphical Similarity Specification This method demands a minimum of explicit knowledge. The expert is asked for an exemplary visualization of his understanding of the similarity between objects. By means of a computer, this visualization is abstracted towards a graph, from which a similarity measure is computed. No additional knowledge is demanded from the expert.

Again let $\mathcal{M} = \{M_1, \ldots, M_n\}$ be the set of axes. The expert manually defines a layout by specifying a function $\rho : \mathcal{M} \to \mathbf{N} \times \mathbf{N}$, which defines a two-dimensional position for each axis. The similarity of two axes M_x, M_y is defined by:

$$sim_{axes}(M_x, M_y) = -||M_x, M_y||_2 \tag{C.9}$$

where $||x, y||_2$ denotes the Euclidean distance between the positions of x and y.
The following points distinguish this knowledge source from the previous one:

- The graphical definition of similarities is closer to the mental model of the user than is the definition of a classification function c.
- By placing n objects, $\frac{n^2}{2}$ similarities are defined.

A difficulty of the graphical similarity specification is that by placing one axis, the distances to $n - 1$ other objects must be taken into account. To simplify this layout problem, only object distances up to certain maximum distance are considered. For this, the layout is clustered in a first step, say, groups of closely related objects are identified.

Remarks. Observe the duality to clustering by multi-dimensional scaling, where the similarity assessments between the objects form the starting point (16, 103). The approach of this section goes the other way round: Having a clustering in the form of classes or a two-dimensional plot, one is able to construct a similarity measure.

Learning and Results Input for the learning step was a database of the form (C.7) with rated axes pairs $(d(M_x), d(M_y), sim_{axes}(M_x, M_y))$, which was used to find values for the weights w_i in the similarity function (C.6). Learning was done by applying least-square regression and by means of neural networks. Details can be found in (21, 234, 302, 112).

The described acquisition methods have been implemented and applied to the learning of similarity measures for fluidic axes. As the subsequent results show, usable similarity measures could be constructed from both knowledge sources.

- *Knowledge Source 1.* 67 fluidic axes were classified into 9 classes by a domain expert; the similarity measure was constructed according to Equation (C.8). The error rate on the learning set was 12%, while the error rate on a test set was 16%.[5] The error rate was defined as the percentage of axes that were misclassified when using the learned similarity measure as classificator.

[5] The learning set comprised the axes used for the learning process; the axes in the test set have not been used for learning.

- *Knowledge Source 2.* A graphical arrangement of circuit documents, which has been proposed by the domain expert, was analyzed and similarity measures were constructed. To evaluate the quality of the learned measures the Mean Square Error (MSE) was used:

$$\sqrt{\sum_{M_x, M_y} (sim^t_{axes}(M_x, M_y) - sim^e_{axes}(M_x, M_y))^2}$$

where sim^t_{axes} denotes the similarity as predicted by the learned similarity measure, while sim^e_{axes} denotes the empirical similarity measure (C.9) defined by the manual layout. On a test set, an MSE of 0.22 has been achieved. Since the similarity values are from the interval $[0, 1]$ the MSE defines an average variation from the empirical similarity.

More on Learning A closer look to the regression method revealed that about 20 features played a predominant role; half of them represented phase orders. Important phase descriptions were "*precision*", "*distance*", and "*number of phases*"; relevant phase orders were "*constant-drive* after *constant-drive*" and "*position-drive* after *hold-position*". An *F*-Test stated a significance of 0.001 respecting the above regression, thus precluding random influences.

The R^2-value of the regression, which measures the amount of explained variation, was 0.2. Say, the regression explained the observation only partially. To improve this result a more complex similarity function than Equation (C.6) can be employed. This makes sense since domain experts allege dependencies between the axes' features. The following similarity measure considers also a possible interactivity between axis features:

$$sim_{axes}(M_x, M_y) = w_0 + \sum_{i=1}^{p} w_i \cdot |f_i^{(x)} - f_i^{(y)}| + \sum_{i=1}^{p} \sum_{j=1}^{p} w_{ij} \cdot |f_i^{(x)} - f_i^{(y)}| \cdot |f_j^{(x)} - f_j^{(y)}|$$

where $w_0, w_i, w_{ij} \in \mathbf{R}$, $f_i^{(x)} \in d(M_x)$, and $f_i^{(y)} \in d(M_y)$. The regression based on this function improved the R^2 value to 0.49, stating that dependencies between features have been captured.

Synopsis

Problemclass Acquisition of similarity knowledge for demanding analysis and synthesis tasks in fluidic engineering.

Problem Solving Method Clustering; regression of weight vectors.

Source Model Collection $\mathcal{B}$ of discrete event models of the form $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, \mathcal{V}, \Delta\rangle$ defined over $\langle F, \mathcal{M}\rangle$ (models of fluidic function). The elements in $\mathcal{M}$ are called fluidic axes; the set $\boldsymbol{\mathcal{M}}$ comprises all fluidic axes in $\mathcal{B}$; i. e., it is the set union of the $\mathcal{M}$ in $\langle F, \mathcal{M}\rangle$, $\langle F, \mathcal{M}\rangle \in \mathcal{B}$. A model of fluidic function in $\mathcal{B}$ is characterized as follows.

- *Fidelity Level F.* Selected fluidic quantities.
- *Granularity Level* $\mathcal{M}$. Fluidic axes; $|\mathcal{M}|$ is the number of axes.
- *Input* F_U. The input variables $f \in F_U$ prescribe extrinsic forces and events at the fluidic axes.
- *State Prescription Function* Δ. The state variables $f \in F_X$, $|F_X| = |\mathcal{M}|$, define a vector of phases; a phase is a symbolic description of a simple fluidic function such as *fast-drive* or *hold-pressure*. The number of states in $\mathcal{X}$ is finite. Δ consists of an external and an internal state prescription function and specifies the discrete phase transitions. The dimension of Δ's domain is bound by $3 \cdot |\mathcal{M}| + 1$, resulting from $|\mathcal{M}|$ state variables, $2 \cdot |\mathcal{M}|$ inputs, and the time base.
- *Behavior Model Processing.* Discrete event simulation.

Reformulated Model $\mathcal{B}$ is unaltered.

- *Similarity Measure.* A function $sim_{axes} : \boldsymbol{\mathcal{M}} \times \boldsymbol{\mathcal{M}} \rightarrow [0; 1]$, which assigns two fluidic axes a similarity value.

Knowledge Source Classification of fluidic axes by a domain expert; the classification can be given as a partitioning of $\boldsymbol{\mathcal{M}}$ or as two-dimensional arrangement of its elements.

D

Model Envisioning

Model envisioning means information visualization or visual data mining respecting models of technical systems; it can be considered as a new kind of problem solving method. In particular, we summarize methods under this term that prepare structure models in a graphical way in order to provide insights or to simplify the access when dealing with large systems. See Page 53, Section 2.4, for a comparison to other model construction approaches.

Model envisioning happens in several steps: structure model preparation, clustering or identification, and graph drawing. Depending on the application, the importance of these steps may vary. Typically, the result of an envisioning process yields a platform for subsequent analysis, modification, maintenance, or acquisition tasks.

This chapter presents three applications where model envisioning has successfully been employed. Section D.1 shows how envisioning is used within the model formulation of computer networks. In Section D.2, envisioning is employed to simplify the understanding and maintenance of configuration knowledge bases, and within Section D.3, envisioning provides a strategy for the functional analysis of fluidic systems.

From the standpoint of the methods employed, the three sections emphasize three different aspects. Section D.1 concentrates on graph clustering; in particular, it introduces both a new measure for the evaluation of clusters and a new algorithm for cluster detection. Section D.2 uses clustering as well, but focuses on graph drawing and classification. Section D.3, however, can be regarded as a domain-specific graph matching problem, which is attacked by the combined application of design graph grammars and shortest-path algorithms.

D.1 Supporting Model Formulation for LANs

The design and configuration of computer networks, such as local area networks (LANs), needs profound knowledge of the volume and the distribution of the network's traffic. While many tools support the recording and the statistical analysis of inter-computer communication, e. g., the traffic can be measured on each data line by so-called RMON-devices, there is no generic concept to form a global model from this data. As a consequence, it is hard to formulate generic design or configuration rules for such networks—although a lot of design decisions have to be met: Which computers shall be bundled in a single virtual LAN? Where to place routers? Which topology is preferable? Must the switching technology be upgraded?

In (265) Stein and Niggemann suggest to tackle the mentioned model formulation and analysis problems by purposefully rendering a network's traffic graphs. The idea is to use clustering techniques to envision the coherence between the measured communication load and the underlying network structure (see Figure D.1).

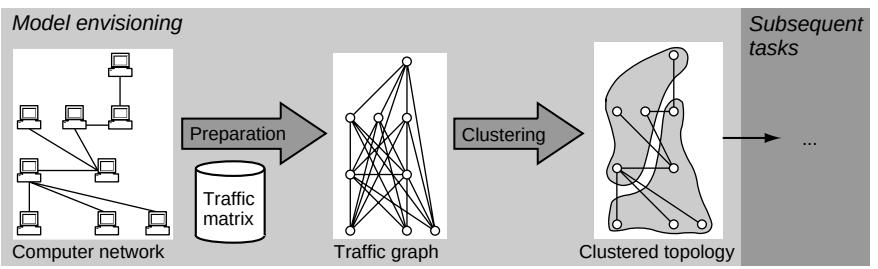


Figure D.1. Model envisioning of a computer network is attained by creating and clustering the network's traffic graph.

The presented approach shall help to identify components that are bottlenecks for the traffic, to detect critical network situations, or to identify shiftings in the communication structure that may result from new users, new technologies, and new tasks.

Note that the coherence between communication clusters and network clusters can be expressed graphically, by using graph drawing methods to envision traffic bottlenecks and critical network situations, as well as numerically. The latter is achieved by comparing the routing effort within the original network structure and the structure proposed by the clustering.

Underlying Models

Definition D.1 (Structured Cabling Traffic Model) *Let S be a network with structured cabling topology and let $\langle F, \mathcal{M}\rangle$ be a model of S. A structured cabling traffic model over $\langle F, \mathcal{M}\rangle$ is a memoryless behavior model, $B\langle F, \mathcal{M}\rangle = \langle F_U, F_Z, \mathcal{V}, \Delta\rangle$, whose elements are defined as follows.*

(1) F is the set of aggregated communication loads; $\mathcal{M}$ defines the nodes of the network and corresponds to the network elements at the primary, secondary, and tertiary level; $|M_i \cap M_j| \leq 1$ with $M_i, M_j \in \mathcal{M}$, $i \neq j$. In particular, let $\mathcal{M}_t \subset \mathcal{M}$ correspond to the network elements at the tertiary level.

(2) For each $M \in \mathcal{M}_t$ there is an input variable in F_U, $|F_U| = |\mathcal{M}_t|$, that defines the communication traffic generated by the user behavior at M; the set of constraint variables, F_Z, defines the communication load between each pair of adjacent nodes. No output variables are considered in this model, and $F = F_U \cup F_Z$.

(3) The sets U_f and Z_f designate the domains of the variables f in F. The domain of each input variable is $\mathbf{N}^{|\mathcal{M}_t|}$; given a vector for the input variable of M_i, $M_i \in \mathcal{M}_t$, its j-th component defines the communication traffic in KByte/s from M_i to M_j, $M_j \in \mathcal{M}_t$. The domain of each constraint variable is $\mathbf{N}$; if $M_i \cap M_j \neq \emptyset$, $M_i, M_j \in \mathcal{M}$, the constraint variable in $M_i \cap M_j$ defines the communication load in KByte/s between M_i and M_j.

$\mathcal{U}$ and $\mathcal{Z}$ designate the Cartesian products of the input variable domains and the constraint variable domains. $\mathcal{V}$ comprises the domains of all functionalities.

(4) The set of state variables, F_X, is empty. The state prescription function Δ maps an input vector $\mathbf{u} \in \mathcal{U}$ onto the global vector of communication loads, say, $\Delta : \mathcal{U} \to \mathcal{Z}$.

(5) Based on F_U, a so-called traffic matrix $\mathbf{A}$ of dimension $|\mathcal{M}_t| \times |\mathcal{M}_t|$ can be defined; $\mathbf{A}$ becomes an upper triangle matrix if communication directions are neglected. Each element $a_{ij,\, i<j}$ in $\mathbf{A}$ defines the communication load between two—not necessarily adjacent—nodes M_i and M_j in $\mathcal{M}_t$.

Example. Table D.1 specifies a model $\langle F, \mathcal{M}\rangle$ of some network and the graph of $\langle F, \mathcal{M}\rangle$, which corresponds to the network topology. Here, the functionalities F are $\{u_1, u_2, u_3, z_1, \ldots, z_5\}$ and the network elements $\mathcal{M}$ are $\{M_1, \ldots, M_6\}$.

The behavior model $B\langle F, \mathcal{M}\rangle$ is defined as follows. The network elements M_1, M_2, and M_3 form the tertiary level $\mathcal{M}_t$; the set of input functionalities, F_U, is $\{u_1, u_2, u_3\}$; the set of constraint functionalities, F_Z, is $\{z_1, \ldots, z_5\}$. The domain U_i for a functionality $u_i \in F_U$ is $\mathbf{N}^3$; the domain Z_i for a functionality $z_i \in F_Z$ is $\mathbf{N}$; $\mathcal{U} := U_1 \times U_2 \times U_3$, and $\mathcal{Z} := Z_1 \times \ldots \times Z_5$.

Objects in $\mathcal{M}$
$M_1 = \{u_1, z_1\}$
$M_2 = \{u_2, z_2\}$
$M_3 = \{u_3, z_3\}$
$M_4 = \{z_1, z_2, z_4\}$
$M_5 = \{z_3, z_5\}$
$M_6 = \{z_4, z_5\}$

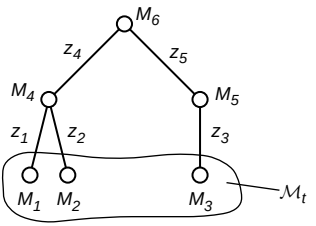


Table D.1. Model $\langle F, \mathcal{M} \rangle$ of a network with structured cabling topology (left-hand side) along with the graph of $\langle F, \mathcal{M} \rangle$ (right-hand side).

In this example, $\Delta(\mathbf{u})$ is defined as the cumulative load on $(z_1, \ldots, z_5)$ when routing the communication traffic of an input vector $\mathbf{u} \in \mathcal{U}$ along shortest paths, assuming uniform edge lengths.

For instance, when given the input $\mathbf{u} = \left(\begin{pmatrix} 0 \\ 15 \\ 12 \end{pmatrix}, \begin{pmatrix} 10 \\ 0 \\ 12 \end{pmatrix}, \begin{pmatrix} 10 \\ 20 \\ 0 \end{pmatrix} \right)$, the value of $\Delta(\mathbf{u}) \in \mathcal{Z}$ is $(47, 57, 54, 54, 54)$. Observe that $\Delta(\mathbf{u})$ defines the communication load on the edges in the above graph of $\langle F, \mathcal{M} \rangle$, generated at M_1, M_2, and M_3.

The directed and the undirected traffic matrix representations of F_U are given in the tables below.

	From:	M_1	M_2	M_3
To:	M_1	0	10	10
	M_2	15	0	20
	M_3	12	12	0

	M_1	M_2	M_3
M_1	0	25	22
M_2	0	0	32
M_3	0	0	0

Definition D.2 (Peer-to-Peer Traffic Graph) *Let $B\langle F, \mathcal{M} \rangle = \langle F_U, F_Z, \mathcal{V}, \Delta \rangle$ be a structured cabling traffic model. The related peer-to-peer traffic graph of $B\langle F, \mathcal{M} \rangle$ is an undirected, weighted graph $G = \langle V, E, w \rangle$, $w : E \to \mathbf{N}$. $V = \{v_1, \ldots, v_{|\mathcal{M}_t|}\}$, E is defined by means of the traffic matrix $\mathbf{A}$:*

$$\{v_i, v_j\} \in E \;\Rightarrow\; w(\{v_i, v_j\}) = a_{ij} + a_{ji}, \quad \wedge \quad a_{ij} + a_{ji} > 0 \;\Rightarrow\; \{v_i, v_j\} \in E$$

Example. With respect to the above example, the peer-to-peer traffic graph G is defined as follows. $V = \{v_1, v_2, v_3\}$; $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}\}$; $w(\{v_1, v_2\}) = 25$, $w(\{v_1, v_3\}) = 22$, $w(\{v_2, v_3\}) = 32$.

Envisioning the Model-Traffic-Coherence

Two new ideas are outlined in this subsection: (1) The clustering of traffic graphs as a means to envision weak points in the design of computer networks, and (2) the Λ-measure and its computation as a means for clustering weighted graphs.[1]

The rationale behind the former point is evident: Large edge weights (= high communication loads) in a traffic graph $G = \langle V, E, w \rangle$ indicate a high degree of node interaction. Clearly, the distance between such nodes in the network must be minimum in order to minimize the routing effort, which in turn can be quantified by the routing's congestion and dilatation values (157).

A network topology should consider the communication behavior by placing nodes with a high communication load close to each other.

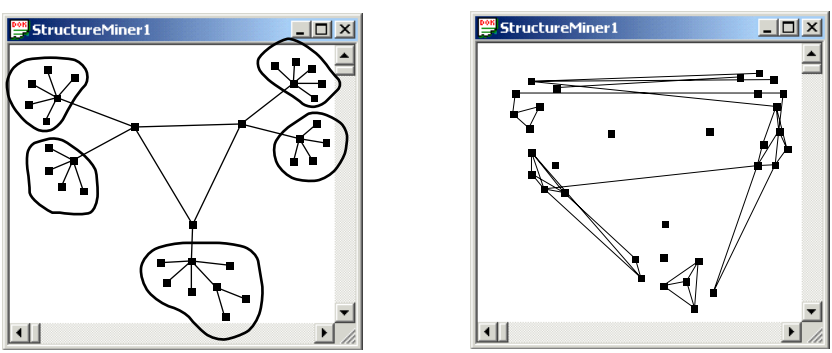


Figure D.2. The left side shows the topology of a local area network; nodes at the tertiary level that belong to the same switch are encircled. The right side shows a traffic graph of this network, which results from a traffic recording; edges with a zero weight have been omitted.

Look at the local area network on the left-hand side of Figure D.2. The encircled nodes are sets of network elements at the tertiary level, and within each set the node distances are minimum. In other words, the network topology defines equivalence classes—say: clusters—within $\mathcal{M}_t$, whereas the objective is to keep the inter-cluster communication load as small as possible.

The right-hand side of Figure D.2 shows a traffic graph that has been constructed from a traffic recording of the left network. The traffic graph contains the same nodes like the network, but all edges belonging to the topology have been removed, while edges indicating the communication loads between the nodes in $\mathcal{M}_t$ have been introduced.

Let us assume that the traffic graph could be clustered with respect to the measured traffic in such a way that the intra-cluster communication is maximized while

[1] The section presents some results of the cooperative and pleasurable research with Oliver Niggemann in the field of clustering and visualization.

the inter-cluster communication is minimized. Then the similarity between the clustering of the network topology and the clustering of the traffic graph envisions the adequacy of the chosen network topology. This similarity is called model-traffic-coherence here.

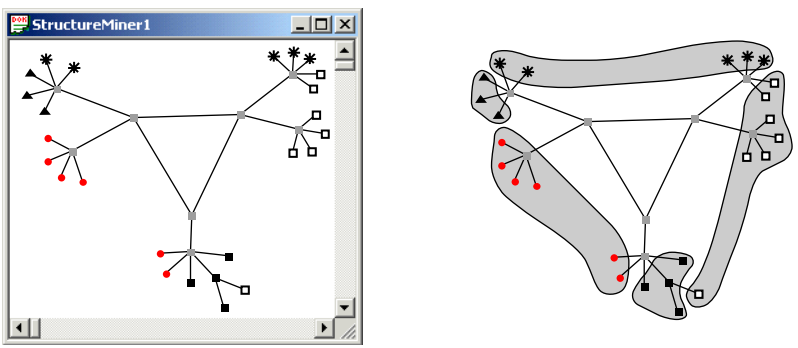


Figure D.3. Clustering of the network topology from Figure D.2 (indicated by the node symbols and by shaded areas on the left and right side respectively). Basis for the clustering is the weighted traffic graph.

Figure D.3 shows a traffic graph clustering according to the outlined philosophy; for envisioning purposes the clustering is drawn within the network topology. Obviously, there is a considerable incoherence between the traffic graph clustering and the topological clustering; it gives rise to a reconfiguration of the network topology.

More information and other applications grounded on traffic visualization can be found in (201). Observe that the most delicate question within the envisioning procedure was kept unanswered up to now: What are suited concepts and algorithms to cluster weighted graphs in the desired way? This and related questions are addressed now.

Clustering: The Λ-Measure Here as well as in the subsequent section, clustering is used as a technique to identify structures within graphs. But what is structure? Our common sense understanding of this term is reflected quite well by the following definition.

> *"Structure defines the organization of parts as dominated by the general character of the whole."*
>
> Merriam-Webster Publishing Company, 1998

To quantify this descriptive definition for graphs, we have developed a new measure, called "weighted partial connectivity", Λ, which has been presented in (265). The weighted partial connectivity is defined for a decomposition—say, a clustering of a graph G, and it is based on the graph-theoretical concept of edge connectivity.

Definition D.3 (Decomposition/Clustering, Cut, Edge Connectivity) *Let $G = \langle V, E \rangle$ be a graph with nodes V and edges E.*[2]

(1) $\mathcal{C}(G) = (C_1, \ldots, C_n)$ is a decomposition (clustering) of G into n subgraphs induced by the C_i, if $\bigcup_{C_i \in \mathcal{C}} = V$ and $C_i \cap C_{j,j \neq i} = \emptyset$. The induced subgraphs $G(C_i)$ are called clusters.

(2) $cut(\mathcal{C}) \subseteq E$ comprises the edges between the clusters.

$$cut(\mathcal{C}) = \bigcup_{C_i, C_j \in \mathcal{C}, i<j} \{(v_i, v_j) \mid (v_i, v_j) \in E, v_i \in C_i, v_j \in C_j\}$$

(3) The edge connectivity $\lambda(G)$ of a graph G denotes the minimum number of edges that must be removed to make G an unconnected graph: $\lambda(G) = \min\{|E'| : E' \subset E \text{ and } G' = \langle V, E \setminus E' \rangle \text{ is not connected}\}$.

Definition D.4 (Λ) *Let G be a graph, let $\mathcal{C} = (C_1, \ldots, C_n)$ be a decomposition of G, and let $\lambda(C_i) \equiv \lambda_i$ designate the edge connectivity of $G(C_i)$. The weighted partial connectivity of $\mathcal{C}$, $\Lambda(\mathcal{C})$, is defined as*

$$\Lambda(\mathcal{C}) := \sum_{i=1}^{n} |C_i| \cdot \lambda_i$$

Figure D.4 shows a graph and three decompositions each of which is evaluated with its weighted partial connectivity Λ. Obviously, a maximization of $\Lambda(G)$ leads to a clustering that resembles G's structure—a fact which suggests the following definition.

Definition D.5 (Λ-Structure) *Let G be a graph, and let $\mathcal{C}^*$ be a decomposition of G that maximizes Λ:*

$$\Lambda(\mathcal{C}^*) \equiv \Lambda^* := \max\{\Lambda(\mathcal{C}) \mid \mathcal{C} \text{ is a decomposition of } G\}$$

Then the contraction $H = \langle \mathcal{C}^*(G), E_{\mathcal{C}^*} \rangle$ is called Λ-Structure, or simply: structure of G.

Remarks. (1) A key feature of the Λ-structure as a clustering criterion is its implicit definition of a structure's number of clusters. (2) The weighted partial connectivity, Λ, can be made independent of the graph size by dividing it by the graph's node number $|V|$. The resulting normalized Λ value is designated by $\bar{\Lambda} \equiv \frac{1}{|V|} \cdot \Lambda$.

[2] Concepts and definitions of graph theory are used in their standard way; they are adopted from (158, 126).

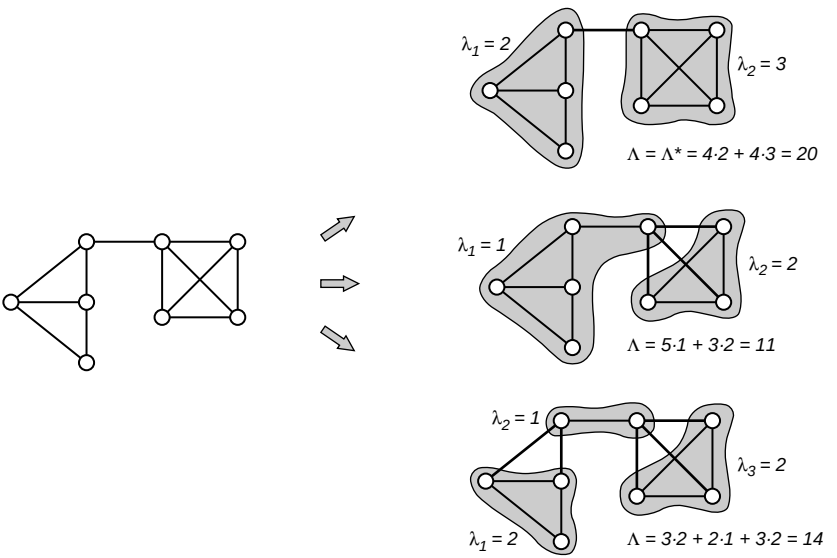


Figure D.4. Three graph decompositions and their related Λ-values.

Operationalizing Λ-Maximization This paragraph presents a fast clustering algorithm that optimizes the weighted partial connectivity Λ. The algorithm implements a local heuristic and is suboptimal.

Initially, each node of a graph gets assigned its own cluster. Within the following re-clustering steps, a node adopts the same cluster as the majority of its neighbors belong to. If there exist several such clusters, one of them is chosen randomly. If re-clustering comes to an end, the algorithm terminates.

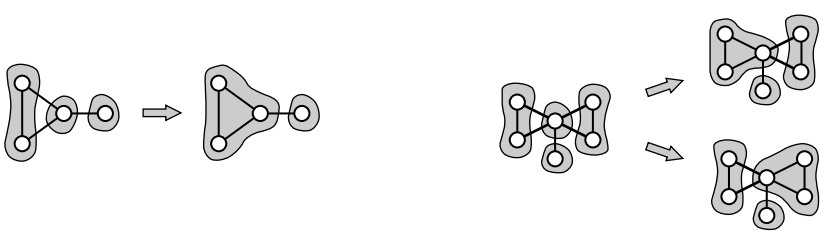


Figure D.5. A definite majority clustering situation (left) and an undecided majority clustering situation (right).

The left-hand side of Figure D.5 shows the definite case: most of the neighbors of the central node belong to the left cluster, and the central node becomes a member of that cluster. In the situation depicted on the right hand side, the central node has

the choice between the left and the right cluster. We now write down this algorithm formally.

MAJORCLUST
Input. A graph $G = \langle V, E \rangle$.
Output. A function $c : V \to \mathbf{N}$, which assigns a cluster number to each node.

(1) $n = 0, t = \mathit{false}$
(2) $\forall v \in V$ **do** $n = n + 1, c(v) = n$ **end**
(3) **while** $t = \mathit{false}$ **do**
(4) $t = \mathit{true}$
(5) $\forall v \in V$ **do**
(6) $c^* = i$ **if** $\left|\{u \mid \{u, v\} \in E \wedge c(u) = i\}\right|$ is maximum
(7) **if** $c(v) \neq c^*$ **then** $c(v) = c^*, t = \mathit{false}$
(8) **end**
(9) **end**

Remarks. (1) To avoid "chaining effects", the choice point situations of the algorithm must be treated randomly. This applies to Step 5, when selecting a node $v \in V$, as well as to Step 6, when several maximum clusters stand to reason to define a node's new cluster membership c^*. (2) A re-clustering in Step 6 is called a definite majority decision, if and only if the maximum of $\left|\{u \mid \{u, v\} \in E \wedge c(u) = i\}\right|$ is unique (cf. the left-hand side of Figure D.5).

Extension for Weighted Graphs. It is both useful and obvious to extend our structure identification approach by introducing edge weights. The amount of the weight $w(e)$ models the importance of an edge $e \in E$. The necessary prerequisite is a generalization of $\Lambda(\mathcal{C})$ by introducing the *weighted* edge connectivity $\tilde{\lambda}$ of a graph $G = \langle V, E \rangle$ as follows.

$$\tilde{\lambda}(G) = \min\{\sum_{e \in E'} w(e) \mid E' \subset E \text{ and } G' = \langle V, E \setminus E' \rangle \text{ is not connected}\}$$

In the same way the algorithm MAJORCLUST is altered: Every node v now adapts the same cluster as the *weighted* majority of its neighbors, i. e., every neighbor counts according to the weight—so to speak: the importance of the edge connecting it to v. Graphs without edge weights can be understood as special cases of weighted graphs with a constant weight function.

Theorem D.1 (Definite Decision Runtime of MAJORCLUST) *The algorithm* MAJORCLUST *terminates after* $\mathcal{O}(|E|)$ *definite majority decisions.*

Proof of Theorem. Let $G = \langle V, E \rangle$ be a graph, and let $\mathcal{C}(G)$ be a decomposition of G. Moreover, $k_{\mathcal{C}} : V \to \mathbf{N}$ defines the number of nodes adjacent to v that have a different cluster membership, say, $k_{\mathcal{C}}(v) = \left|\{\{v, w\} \mid \{v, w\} \in E, w \in V, c(v) \neq c(w)\}\right|$, and $K_{\mathcal{C}}(G) = \sum_{v \in V} k_{\mathcal{C}}(v)$ defines the sum of all $k_{\mathcal{C}}$-values for G. $K_{\mathcal{C}}(G)$ is connected to

the set of inter-cluster edges by the identity $K_{\mathcal{C}}(G) = 2 \cdot |cut(C(G))|$, and obviously holds the inequation $K_{\mathcal{C}'}(G) \leq 2 \cdot |E|$ for every decomposition $\mathcal{C}'(G)$.
Let $v \in V$ be the node that will change its cluster membership in consequence of a definite majority decision, and let $\{C_1, \ldots, C_{k_\mathcal{C}}\} \subseteq \mathcal{C}(G)$ designate the clusters adjacent to v before the cluster change. Without loss of generality we assume the following ordering of the adjacent clusters according to their sizes: $|C_1| \geq |C_2| \geq \ldots \geq |C_{k_\mathcal{C}}|$. If v is currently a member of cluster C_j it follows that $|C_1| > |C_j|, j \geq 2$, since v is subject to a definite majority decision.
After v's move from cluster C_j to C_1, $K_{\mathcal{C}}(G)$ is decreased by $2 \cdot |C_1|$ and increased by $2 \cdot |C_j|$, where $2 \cdot (|C_1| - |C_j|) > 0$. Since initially $K_{\mathcal{C}}(G) \leq 2 \cdot |E|$ holds, MAJORCLUST must terminate after $\mathcal{O}(|E|)$ definite majority decisions. ◇

Recall that the above MAJORCLUST-algorithm permits "pathological" cases, where the algorithm oscillates between two or more indefinite clustering situations. Although such a situation is unlikely to happen, it does prohibit the specification of a worst case runtime complexity. Corollary D.1 circumvents this indeterminacy by imposing an additional restriction.

Corollary D.1 (Runtime of MAJORCLUST) *Assuming that* MAJORCLUST *terminates if it comes to no definite majority decision for* $|V|$ *steps,* MAJORCLUST *terminates after* $\mathcal{O}(|V| \cdot |E|)$ *steps.*

Proof of Corollary. The proposition follows directly from Theorem D.1.

Remarks. The restriction of the corollary poses no severe limitation for the transferability of the runtime considerations: Indefinite clustering situations can be averted completely by adding a randomly distributed weight surcharge $\delta(e), e \in E$, to every edge weight $w(e)$. By constructing δ sufficiently small with regard to the minimum of the weight function w, the overall characteristic of the edge weight distribution is not affected. This advisement and experimental results show the usability of the algorithm for large graphs with several thousand nodes.

The algorithm's greatest strength, its restriction to local decisions, is bound up with its sub-optimality. In every step only a node's neighbors are considered, resulting in an excellent runtime behavior. On the other hand, by disregarding global criteria like the connectivity, MAJORCLUST cannot always find the optimum solution.

Figure D.6 exemplifies the behavior: The optimum solution for graph (a) is one cluster, which is also the solution as found by MAJORCLUST. For graph (b), a splitting into the two clusters $\{v_1\}$ and $V \setminus \{v_1\}$ is optimum. MAJORCLUST cannot find this decomposition—working strictly locally, it behaves exactly as on graph (a) and creates only one cluster.

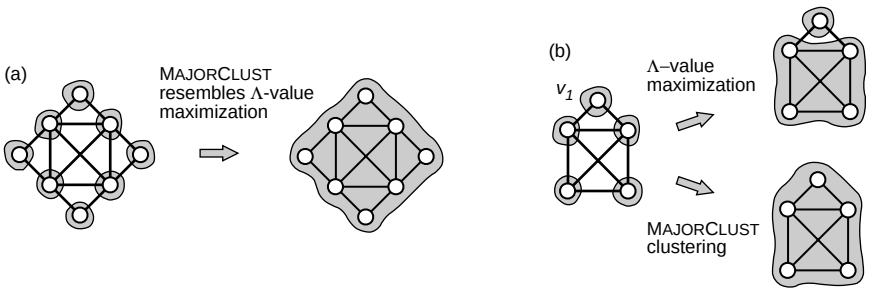


Figure D.6. The local behavior of MAJORCLUST may lead to sub-optimum Λ-values.

Clustering: Comparison and Theoretical Considerations

This subsection outlines well-known clustering approaches (the interested reader may delve in (122, 69), among others) and gives a performance comparison between two representatives and MAJORCLUST. As well as that it presents an interesting theoretical contribution: By formulating the "strong splitting condition" a close relation between Min-cut clustering and Λ-maximization is pointed out.

Existing clustering approaches can be qualified as working hierarchically or non-hierarchically. Hierarchical algorithms create a tree of node subsets by successively subdividing or merging the graph's node sets. In order to obtain a unique clustering, a second step is necessary that prunes this tree at adequate places.

Hierarchical algorithms can be further classified into agglomerative and divisive approaches. The former start with each vertex being its own cluster and union clusters iteratively. The latter start with the entire graph as a single cluster, which is successively subdivided. Examples for divisive algorithms are Min-cut clustering or dissimilarity-based algorithms; typical agglomerative algorithms are k-nearest neighbor or linkage methods. Non-Hierarchical algorithms subdivide the graph into clusters within one step; examples are clustering techniques based on minimum spanning trees (307), self-organizing Kohonen networks, or approaches which optimize a given goal criterion (18, 227, 231).

Λ-maximization and MAJORCLUST can be qualified as non-hierarchical and exclusive. MAJORCLUST quickly finds a—usually suboptimal—solution for the problem of Λ-maximization. Based on a set of randomly created graphs, Figure D.7 contrasts MAJORCLUST's runtime characteristic related to Min-cut clustering and Kohonen clustering. More comprehensive analyses of MAJORCLUST are given in (198, 182).

Clustering Based on Nearest-Neighbor Strategies Nearest-neighbor clustering operates by merging the two closest clusters; its widespread use results in several variations (80, 252, 125, 69). The following qualitative comparison to MAJORCLUST cannot take all existing variations into consideration.

(1) Nearest-Neighbor clustering, like all hierarchical algorithms, does not define the (optimum) number of clusters. Λ-maximization implicitly defines both cluster number and size.

(2) The greedy nature of nearest-neighbor methods (unlike as in MAJORCLUST, nodes are never reassigned to another cluster) leads to so-called chaining effects (69).

(3) The transformation of the partitioning tree into a unique clustering results in difficulties if clusters have strongly varying point densities or inter-cluster distances.

(4) Nearest-neighbor methods rely on distance information only and disregard connectivity information. For weighted graphs this may lead to clusterings which lack the human sense of esthetics, for unweighted graphs this may result in a failure to find any clusters.

Clustering Based on the Minimum Cut Min-cut clustering subdivides a weighted graph recursively at its minimum cut (158, 304), which is the smallest set of edges whose removal disaggregates the graph. Min-cut clustering can produce natural clusterings but is bound up with several problems.

(1) It has to be predefined when a cluster should not be subdivided anymore.

(2) The computation of a graph's minimum cut is expensive. Moreover, the minimum cut is often not definite, resulting in a choice point situation when recursively dividing the graph.

(3) Min-cut clustering tends to form clusters with a single point, thus lacking the human sense of esthetics.

Kohonen Clustering This method defines a clustering of a weighted graph implicitly by centroid nodes (147, 21, 184): Each node belongs to its closest centroid node. Initially, a number of centroid nodes is chosen randomly from the graph's node set. Then, by iteratively investigating all nodes, each centroid node moves into the center of its cluster. The algorithm terminates when all centroid nodes possess stable positions.

Aside from the difficulty of determining a useful number of centroid nodes, its main restriction results from the behavior to create always centric cluster regions.

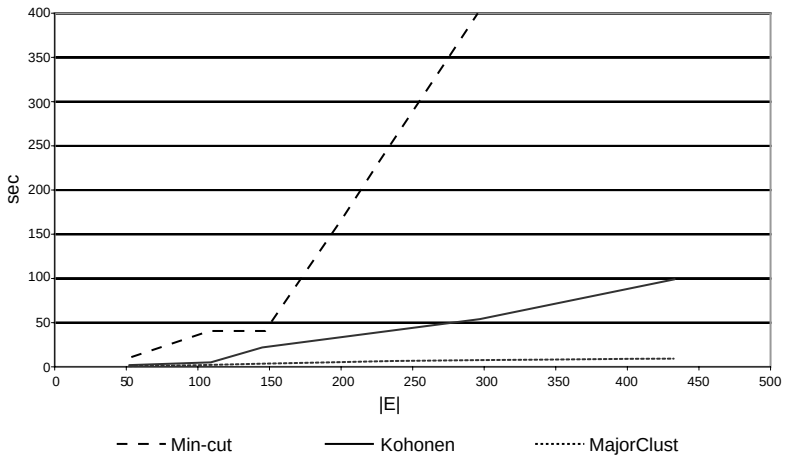


Figure D.7. Experimentally determined runtime behavior between Min-cut clustering, Kohonen clustering, and MAJORCLUST. The underlying graph data base contains about 1000 graphs whose average degrees lie between 4 and 20.

Theoretical Considerations Different rules of decomposition, which are implied by the Λ-structure clustering, are worth to be noted. They come into play if the decomposition of $\mathcal{C}(G)$ of a (sub)graph G fulfills the property stated in the next definition.

Definition D.6 (Strong Splitting Condition (265)) *Let $G = \langle V, E\rangle$ be a graph, and let $\mathcal{C}(G) = (C_1, \ldots, C_n)$ be a decomposition of G. $\mathcal{C}(G)$ fulfills the strong splitting condition, if the following strict inequation holds:*

$$\lambda(G) < \min\{\lambda_1, \ldots, \lambda_n\}$$

Remarks. (1) Let Λ_1 designate the Λ-value of the trivial decomposition $\mathcal{C}_1(G)$ of a graph G, where $\mathcal{C}_1(G) \equiv (V)$. If for some $\mathcal{C}(G)$ the strong splitting condition is satisfied, the application of the proposed decomposition enlarges the Λ-value for G, i. e., $\Lambda(\mathcal{C}(G)) > \Lambda_1$. In this sense, the strong splitting condition can be designated as a commensurate decomposition rule.
(2) Let G be a subgraph of some graph H, and let some decomposition $\mathcal{C}(G)$ of G fulfill the strong splitting condition. Then the application of the $\mathcal{C}(G)$ raises the mean of the clusters' connectivity values λ_i in H.
(3) If for no decomposition $\mathcal{C}$ the strong splitting condition holds, G will be decomposed only, if for some $\mathcal{C}$ the condition $|V| \cdot \lambda(G) < \Lambda(\mathcal{C})$ is fulfilled. This inequality establishes a necessary condition for decomposition—it is equivalent to the following special case of the structure definition: $\max\{\Lambda(\{V\}), \Lambda(\mathcal{C})\} = \Lambda(\mathcal{C})$, since $\Lambda(\{V\}) \equiv |V| \cdot \lambda(G)$.

Obviously does a graph decomposition according to the strong splitting condition follow the human sense when identifying clusters or structures in a graph, and there is a relation to the Min-cut splitting approach, which is pointed out now.

Theorem D.2 (Strong Splitting Condition) *Let $G = \langle V, E \rangle$ be a graph, and let $\mathcal{C}(G) = (C_1, C_2)$ be a decomposition of G into two clusters. If $\mathcal{C}(G)$ fulfills the strong splitting condition $\lambda(G) < \min\{\lambda_1, \lambda_2\}$, the application of $\mathcal{C}(G)$ results in a decomposition at a minimum cut.*

Proof of Theorem. Let $\mathcal{C}'(G)$ be a decomposition of G with $|\mathcal{C}'(G)| = 2$ and $\mathcal{C}'(G) \neq \mathcal{C}(G)$, and let $cut(\mathcal{C}'(G))$ be minimum. $\mathcal{C}'(G) \neq \mathcal{C}(G)$ entails that the decomposition prescribed by $\mathcal{C}'(G)$ splits either C_1, C_2, or both. Since the respective edge connectivities λ_1 and λ_2 are strictly larger than λ, every split of C_1 or C_2 must contain more than λ edges. Hence, $cut(\mathcal{C}'(G))$ cannot be minimum. ◇

Remarks. If the strong splitting condition does not apply, an optimum decomposition according to the structuring value need not be the same decomposition as found by a minimum cut strategy. This is a consequence of the latter's disregard for cluster sizes. Figure D.8 demonstrates such an example. Here, C_x refers to a clique with $x \geq 3$ nodes. An optimum solution according to the weighted partial connectivity Λ, which is also closer to the human sense of esthetics, consists of one cluster $\{v_1, v_2, v_3, v_4\}$ and a second cluster C_x. An algorithm that employs Min-cut splitting does only separate v_1.

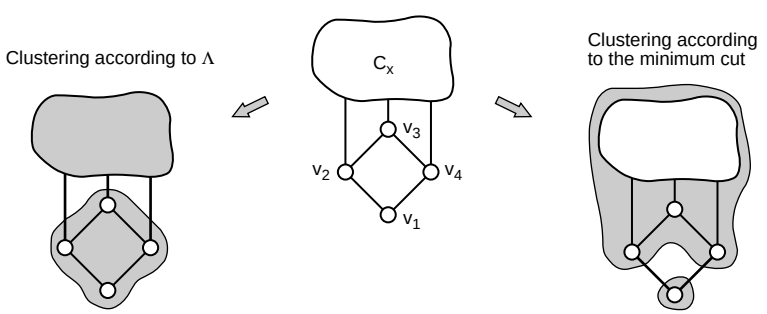


Figure D.8. Weighted partial connectivity (Λ-) maximization versus Min-cut clustering..

Observe that, as already pointed out, maximizing the weighted partial connectivity determines the optimum cluster number, while the minimum cut approach lacks any criterion for the number of necessary division steps.

Synopsis

Problemclass Model formulation of computer networks.

Problem Solving Method No generic computer-automated method available.

Source Model Memoryless behavior model $\langle F_U, F_Z, \mathcal{V}, \Delta, \Lambda \rangle$, defined over $\langle F, \mathcal{M} \rangle$.

- *Fidelity Level F.* Aggregated communication loads, specified in KByte/s.
- *Granularity Level $\mathcal{M}$.* Network elements, such as servers, clients, switches, bridges, and hubs. $\mathcal{M}_t \subset \mathcal{M}$ corresponds to the servers and clients at the tertiary level.
- *Input $F_U \subset F$.* Communication load caused by the user behavior at the network elements in $\mathcal{M}_t$; $|F_U| = |\mathcal{M}_t|$.
- *State Prescription Function Δ.* $F_X = \emptyset$ (memoryless). Δ maps an input vector onto the global vector of communication loads.
- *Order of Magnitude of the Application.* Experiments have been performed with $|\mathcal{M}_t|$ ranging from 100 to 2000.

Envisioned Model Clustering of a structure model $S\langle F, \mathcal{M} \rangle$.

- *Clustering.* Definition of equivalence classes on the elements in $\mathcal{M}_t$.
- *Model-Traffic-Coherence.* Coherence between network topology and traffic clustering. The coherence can be evaluated either graphically or numerically. The latter is achieved by relating the congestion and dilatation values of the original topology with those of the improved topology.
- *Subsequent Tasks Enabled.* Traffic analysis, identification of bottlenecks, network administration, network reconfiguration.

Knowledge Source Based on Δ, construction of the weighted peer-to-peer traffic graph G. Clustering of G by MAJORCLUST and Λ-maximization.

D.2 Maintaining Knowledge Bases

The maintenance and analysis of knowledge bases is of central importance when tackling knowledge-intensive tasks. This section introduces envisioning concepts that have been developed to simplify the access and the handling of knowledge bases as they are used within resource-based configuration.

The resource-based configuration paradigm is a successful approach in the field of automatic configuration of technical systems. It is based on the resource-based component model, wherein the components of the interesting technical system are described by simple property constraints (105). Section B.1 of this work looks at resource-based models from the compilation point of view and gives a detailed introduction to this configuration approach.

Resource-based configuration obtained its popularity since it allows for the local modeling of large systems: If a component description is modified or if a new component is added to the knowledge base, it is not necessary to review the correctness of all possible configurations. The constraint modeling paradigm guarantees that a configuration that fulfills the resource constraints is technically sound.

However, with increasing complexity of the system to be configured, knowledge acquisition and knowledge management, such as comparative analyses between several knowledge bases, becomes a challenge. In this connection, the visualization of configuration knowledge bases in the form of component graphs is an interesting approach: By automatically identifying and rendering the modules and the assembly of the system to be configured, the knowledge engineer can be supported within reengineering and maintenance tasks. See Figure D.9 for an overview of the envisioning process.

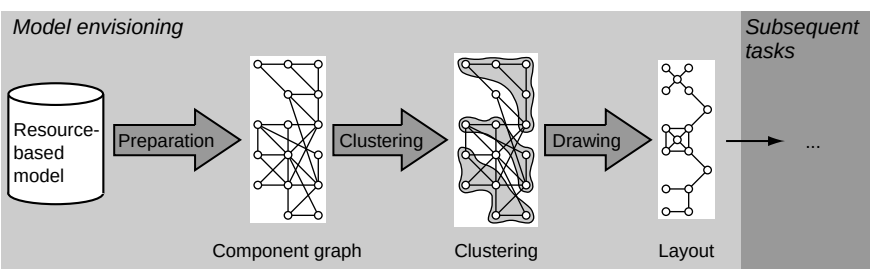


Figure D.9. Model envisioning of a resource-based model is attained by creating, clustering, and drawing the model's component graph.

Underlying Models

Definition D.7 (Component-Resource Graph) *Let $\langle F, \mathcal{M} \rangle$ be a model. A component resource graph over $\langle F, \mathcal{M} \rangle$ is a directed, bipartite graph $\langle V, E \rangle$ whose elements are defined as follows.*

(1) $V = V_{\mathcal{O}} \cup V_{\mathcal{R}}$ where $V_{\mathcal{O}} \cap V_{\mathcal{R}} = \emptyset$ and $|V| = |\mathcal{M}|$. The nodes in $V_{\mathcal{O}}$ and $V_{\mathcal{R}}$ correspond to the objects and resources in $\mathcal{M}$ respectively (see Section B.1, Page 124, for an introduction to the underlying ideas).

(2) The semantics of an edge (v_o, v_r) or (v_r, v_o), $v_o \in V_{\mathcal{O}}$, $v_r \in V_{\mathcal{R}}$ is as follows. Object O, corresponding to v_o, supplies (demands) resource R corresponding to v_R.

Remarks. This definition is similar to Definition B.3, Page 130, where component-resource graphs were already introduced. In contrast to the former, this definition does not rely on a behavior model.

Example. The component-resource graph $\langle V, E \rangle$ in Figure D.10 belongs to a model with four components and four resources. $V = V_{\mathcal{O}} \cup V_{\mathcal{R}}$, with $V_{\mathcal{O}} = \{v_{O_1}, v_{O_2}, v_{O_3}, v_{O_4}\}$ and $V_{\mathcal{R}} = \{v_{R_1}, v_{R_2}, v_{R_3}, v_{R_4}\}$; the set of edges, E, may be read of the figure.

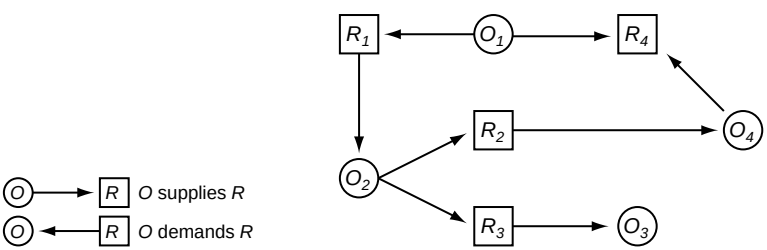


Figure D.10. Component-resource graph of some model $\langle F, \mathcal{M} \rangle$. $O_1 \in V_{\mathcal{O}}$, for instance, supplies R_1 and R_4 and demands nothing.

Definition D.8 (Component Graph) *Let $\langle F, \mathcal{M} \rangle$ be a model with the component-resource graph $G = \langle V_{\mathcal{O}} \cup V_{\mathcal{R}}, E \rangle$. G becomes a component graph $G' = \langle V', E', \sigma \rangle$ by deleting from G the node set $V_{\mathcal{R}}$ and by introducing for each $r \in V_{\mathcal{R}}$ short cut edges between all nodes incident to r that are connected by a directed path in G:*

$$\begin{aligned} V' &= V \setminus V_{\mathcal{R}}, \\ E' &= \{(v, w) \mid (v, r), (r, w) \in E, r \in V_{\mathcal{R}}\} \end{aligned}$$

The labeling function $\sigma : E' \to \Sigma$ labels the edges in G' according to the resources $R \in \mathcal{R}$, $\mathcal{R} \subset \mathcal{M}$.

Remarks. Note that a component graph is closer to the structural set-up of a technical system and thus is better suited for envisioning purposes than is the component-resource graph. Also note that the directional information of the edges is preserved. Figure D.11 shows the component graph of the previous example.

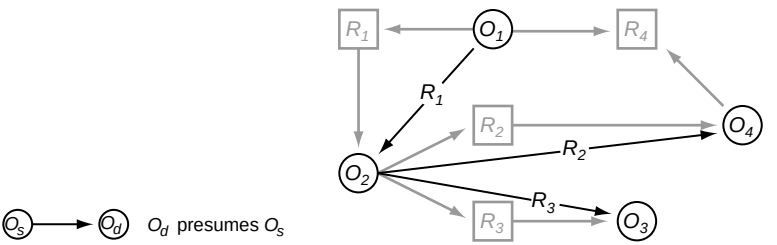


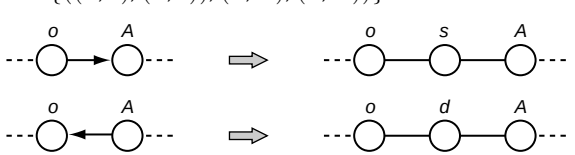
Figure D.11. The figure shows the component graph to the component-resource graph from Figure D.10; the latter is shown gray in the background.

Envisioning Resource-Based Models

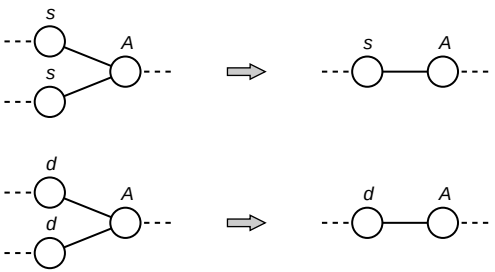
The envisioning of resource-based models is done by generating, clustering, and drawing of the respective component graph.

Generating the Component Graph The design graph grammar $\mathcal{G} = \langle \Sigma, P \rangle$ below transforms a component-resource graph into a component graph. Note that this transformation must consider the multiple supply and demand of resources, which can lead to the introduction of $\mathcal{O}(|V_\mathcal{O}|^2)$ new edges. $\Sigma = \{o, s, d, A, B, C, D\}$, P contains six rules, $T \rightarrow \langle R, I \rangle$: two resource encoding rules, two gathering rules, and two object combination rules; the first of each is specified formally while the graphics shows both of them.

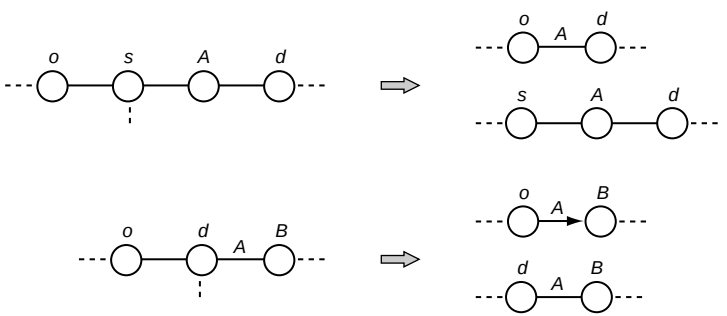
(1) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2\}, \{(1,2)\}, \{(1,o),(2,A)\} \rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{3,4,5\}, \{\{3,4\},\{4,5\}\}, \{(3,o),(4,s),(5,A)\} \rangle$
$I = \{((B,o),(B,o)),(B,A),(B,A))\}$



(2) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3\}, \{\{1,3\}, \{2,3\}\}, \{(1,s), (2,s), (3,A)\} \rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5\}, \{\{4,5\}\}, \{(4,s), (5,A)\} \rangle$
$I = \{((B,s),(B,s)), ((B,A),(B,A))\}$



(3) $T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2,3,4\}, \{\{1,2\}, \{2,3\}, \{3,4\}\},$
$\{(1,o), (2,s), (3,A), (4,d)\} \rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{5,6,7,8,9\}, \{\{5,6\}, \{7,8\}, \{8,9\}\},$
$\{(5,o), (6,d), (7,s), (8,A), (9,d), (\{5,6\}, A)\} \rangle$
$I = \{((B,o),(B,o)), ((C,s),(C,s)), ((D,d),(D,d))\}$



The edge labels represent the resources that are shared between adjacent nodes, say, components. It is plausible that different resources establish different connection strengths between components, and that domain knowledge can be used to define an order amongst the resources in $\mathcal{R} \subset \mathcal{M}$.

Within the resource-based component models we divide the resources into the four classes listed in Table D.2. The connection strength (1 = weakly connected, 4 = strongly connected), given in the third column, is used to define a weight function on the edges of the component graph.

Connection	Technical interpretation	Strength
$presume(o_1, o_2)$	o_1 needs o_2 without explicitly using a resource	1
$simple(o_1, o_2)$	o_1 uses a resource of o_2	2
$plug\text{-}in(o_1, o_2)$	o_1 uses a resource of o_2 and is connected to o_2	3
$slot(o_1, o_2)$	o_1 uses both a resource and place within o_2	4

Table D.2. Connection types, their technical interpretation in the domain, and a number indicating the characteristic connection strength.

Clustering To come straight to the point, the clustering of the component graph is accomplished by Λ-maximization, which is approximated with MAJORCLUST (see Section D.1, Page 199). The remainder of this paragraph explains the rationale behind the clustering of structure models and underpins the adequacy of Λ-maximization.

The use of clustering methods to render the structure model of a technical system relies on the following paradigms (cf. 265, pg. 124).

(1) *Modular Character.* The system can be decomposed into several modules or functions such that each element of the system (say, each node of its component graph) belongs to exactly one module.

(2) *Connectivity.* Modules are defined implicitly, merely exploiting the graph-theoretical concept of connectivity: The connectivity between nodes assigned to the same module is assumed to be higher than the connectivity between any two nodes from two different modules.

Remarks. Point 1 reflects hierarchical or decentralization aspects of a system or an organization. Point 2 is based on the observation that the elements within a module are closely related; the modules themselves, however, are coupled by narrow interfaces only. A similar observation can be made respecting organizational or biological structures. These structuring paradigms may not apply to all kinds of systems—but, for a broad class of technical systems they form a useful set of assumptions.

The narrow-interface-property of Point 2 suggests the following definition for the quality of a clustering $\mathcal{C}$ (see Page 199 for the definition of a clustering and its cut).

Definition D.9 (Quality of a Clustering) *Let $G = \langle V, E \rangle$ be a graph. The quality of a clustering, $q(\mathcal{C})$, is defined as the ratio of the cluster density, $d(\mathcal{C})$, and the cluster connectivity, $c(\mathcal{C})$.*

$$q(\mathcal{C}) = \frac{d(\mathcal{C})}{c(\mathcal{C})}, \quad \text{where } d(\mathcal{C}) = \frac{|E| - |\,cut(\mathcal{C})|}{|V|} \quad \text{and} \quad c(\mathcal{C}) = \frac{|\,cut(\mathcal{C})|}{|\mathcal{C}|}.$$

Experiments have shown that Λ-maximization resembles a maximization of $q(\mathcal{C})$. Because of the involved graph properties and numerical effects that cancel each other,

this behavior cannot be proved in its generality for arbitrary graphs. Even if a comparison of Λ-maximization and q-maximization is restricted to clusterings with a constant number of clusters, no strict dominance can be observed.

Example. Figure D.12 shows for three similar graphs two clusterings $\mathcal{C}_1$ (left) and $\mathcal{C}_2$ (right) with $|\mathcal{C}_1| = |\mathcal{C}_2| = 2$. While $\Lambda(\mathcal{C}_1) < \Lambda(\mathcal{C}_2)$ holds within all cases, the q-values develop irregularly: $q(\mathcal{C}_1) < q(\mathcal{C}_2)$ in Case 1, $q(\mathcal{C}_1) = q(\mathcal{C}_2)$ in Case 2, and $q(\mathcal{C}_1) > q(\mathcal{C}_2)$ in Case 3.

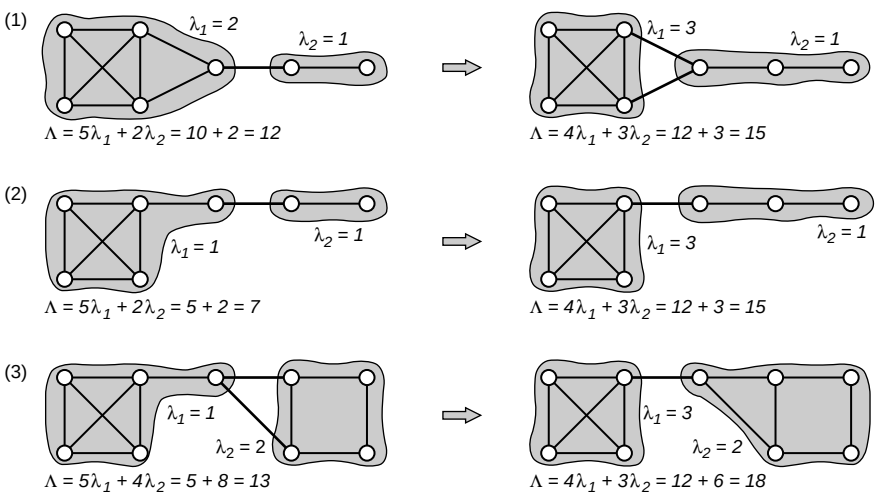


Figure D.12. Top down: Each line shows a graph with two clustering alternatives (left and right) and the respective Λ-values.

The reason for the divergent behavior is that the Λ-value considers a density *distribution* by reckoning up individual cluster densities, while the q-value relies on a single density mean. This shows the supremacy of Λ-maximization over q-maximization. Note that clustering based on a graph's minimum cut (see Page 204) does also consider individual densities, but treats only two clusters at the same time. Λ-maximization, on the other hand, is not restricted to a fix cluster number but determines the cluster number implicitly—a fact that makes Λ-maximization superior to Min-cut clustering as well.

Component Graph Drawing Drawing a clustered component graph happens within two steps: Cluster arrangement and drawing subgraphs in the clusters. The former step is accomplished by a simulated annealing optimization whose evaluation method aims at a minimization of the entire layout area but penalizes the intersection of cluster boundings. Details can be found in (200).

For the graph drawing step a circular layout algorithm proved to be first choice.

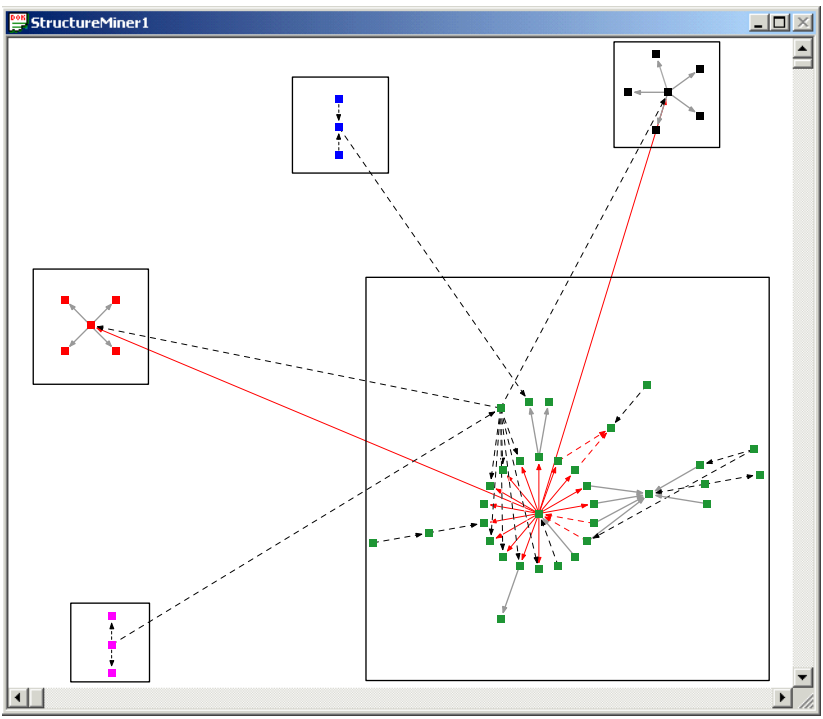


Figure D.13. The component graph of the knowledge base for the configuration of telecommunication systems.

This algorithm first constructs a minimum spanning tree and defines the center of this tree as the new root (186). The nodes are placed in concentric circles around the center node while each layer of the tree forms its own circle (55). The runtime complexity $\mathcal{O}(|E| \cdot \log |E|)$ is bound by the minimum spanning tree computation.

The Figures D.13 and D.14 depict knowledge base envisioning examples: They show component graphs of an original knowledge base that is used for the resource-based configuration of telecommunication systems. The presented layouts have been computed according to the mentioned principles. The snapshots were created with the STRUCTUREMINER system, which has been developed by Niggemann (198) and which contains the mentioned as well as several other layout algorithms. Note that due to the lack of color, fading features, and interactive zooming the printed layouts must stay behind their screen counterparts in the STRUCTUREMINER program.

Component Graph Classification Envisioning knowledge bases does not only relate to lucid graph layouts but is a collective term for various knowledge base man-

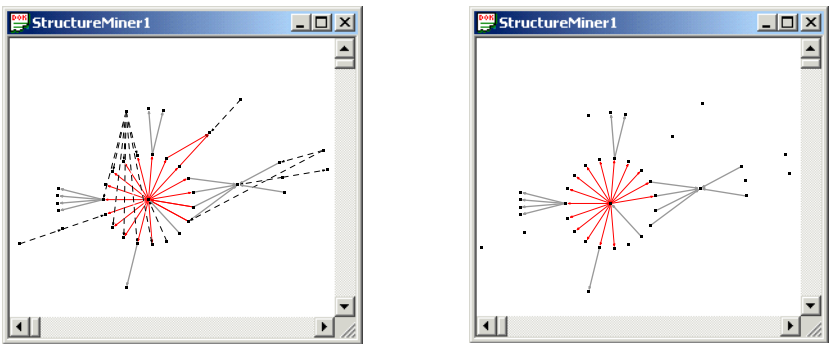


Figure D.14. The left snapshot shows the largest cluster of the above knowledge base. The right snapshot renders the same cluster but shows only connections of the types "plug-in" and "slot".

agement methods. One such method is called "component graph classification" here. The method is intended to support the comparative analysis when several knowledge bases are given, which stem, for instance, from the same domain, from modeling variants of the same technical system, or from different points of time in a knowledge base's life cycle. Among others, comparative analyses play a role within configuration problems.

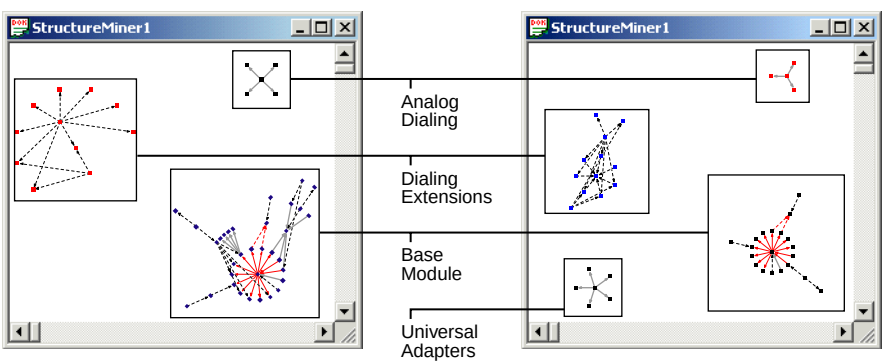


Figure D.15. The figure illustrates component graph classification: Between two configuration knowledge-bases, which describe telecommunication system variants, the same technical modules are identified and rendered in a similar way.

Figure D.15 shows two component graphs, which originate from the configuration knowledge bases of two similar technical systems. By component graph classification it becomes possible to contrast those parts of the system that represent the same module, or that resemble each other with respect to their function.

The classification of a component graph G is a matter of similarity assessment

for the clusters found in G—each of which is represented by a subgraph H_x of G, $x \in \{1,\ldots,k\}$. The characterization of a subgraph H in turn is reduced to a p-dimensional vector of graph features $d(H)$ listed in Table D.3, say, $d(H) = (f_1,\ldots,f_p) \in \mathbf{R}^p$. Observe that the features are independent of the domain.

Feature
number of nodes
number of edges
minimum/maximum/average node indegree
minimum/maximum/average node outdegree
minimum/maximum/average distance between nodes
diameter
number of biconnected components
edge connectivity
number of clusters found by MAJORCLUST

Table D.3. Graph features chosen for the cluster similarity assessment.

We used regression to construct for each cluster $x \in \{1,\ldots,k\}$ a classification function $c_x : \mathbf{R}^p \to \{1,\ldots,k\}$, which maps a feature vector $d(H)$ of an arbitrary graph H onto $[0;1]$. A value of $c_x(d(H)) = 0$ indicates that cluster x and the cluster represented by H are not similar, while $c_x(d(H)) = 1$ stands for a very high cluster similarity. The function c_y of a maximum similar cluster respecting a feature vector $d(H)$ fulfills the inequation $c_y(d(H)) \geq c_x(d(H)), x = 1,\ldots,k$. Figure D.16 illustrates the classificator.

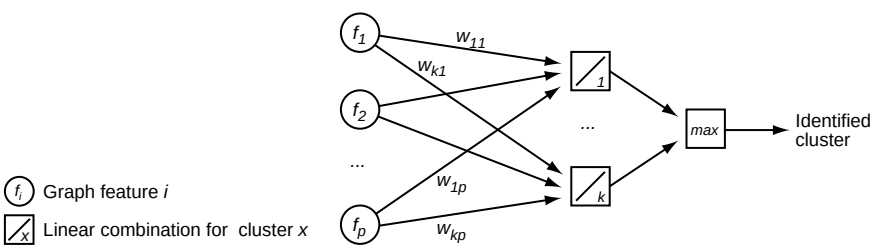


Figure D.16. For each of the k clusters of a component graph a classification function c is constructed, which maps a graph feature vector $(f_1,\ldots,f_p)$ onto $[0;1]$.

For this kind of direct classification a linear model for the functions c_x proved to be sufficient (10):

$$c_x(f_1,\ldots,f_p) = w_0 + \sum_{1\leq j\leq p} w_{ij}\cdot f_j, \quad x = 1,\ldots,k$$

The weights w_{ij} are computed by means of k multiple linear regressions but could be learned using a neural network approach as well (302, 21, 226). Input for the

regression of a function c_x are vectors $(d(H), v)$, where $v = 1$ if H represents the cluster x and $v = 0$ otherwise.

In our knowledge base example (see Figure D.13 and Figure D.15), regression was performed on a training set with 42 clusters. The discrimination quality of the learned functions was verified with 5 test sets containing between 16 and 22 clusters: The classification error was less than 16%. In other words, in most of the given knowledge bases (each describing a different system) it was possible to identify those modules that play a similar technical role.

Graph Drawing: Other Methods

A variety of methods has been developed to draw graphs nicely; most of them pursue a categorical approach: They try to arrange a graph according to a particular scheme, paradigm, or philosophy, such as hierarchical leveling, attracting forces, or recurring resemblances (280, 220, 60, 86, 128, 248, 281, 303). A categorical approach can produce excellent layouts if a graph is biased towards some layout paradigm, and if this property of the graph is actually detected.

Without implying a graph structure, the layout of a graph can be defined by a quality measure, q, that captures a variety of esthetics criteria. Nevertheless, depending on q and the graph size, the maximization of q establishes an extremely difficult optimization problem, which can rarely be solved efficiently.

Recent graph drawing developments arm up layout algorithms by the exploitation of cluster information: A graph is divided into subgraphs, the so-called clusters, which can be laid out rather independently from each other. Our graph drawing approach of the previous subsection can be seen as a precursor of these developments (199).

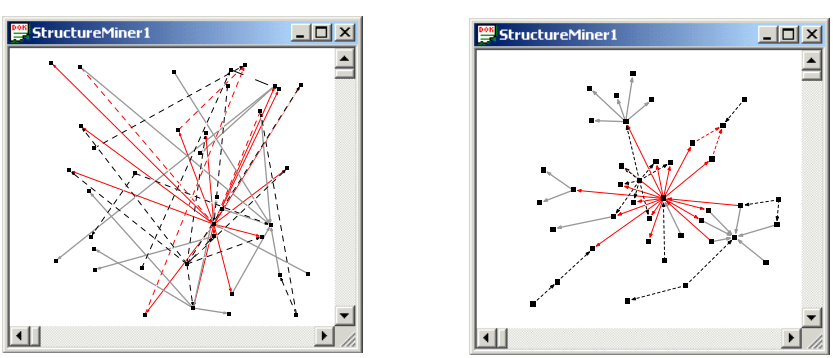


Figure D.17. The left snapshot shows a random layout of the "base module"-cluster of Figure D.14; the right snapshot shows the same cluster layouted by means of a spring-embedder algorithm.

Good overviews on graph drawing algorithms can be found in (282, 55). Their pros and cons with respect to visual structure mining have been investigated by us and are discussed amongst others in (198). A hierarchical leveling algorithm (280), for instance, is suited well when component-resource graphs instead of component graphs shall be envisioned (200).

Finally, we present in Figure D.17 (right-hand side) a layout of a part of the knowledge base example, generated with the well-known spring embedder algorithm (128). This algorithm belongs to the class of "force-directed" layout algorithms that produce acceptable layouts for a wide range of graphs. Force-directed algorithms model a graph as a set of rings (= nodes) connected by springs (= edges). The nodes are placed in some initial state and the springs move the nodes toward a minimum energy state. This energy is reduced by solving a partial differential equation for each vertex. Each vertex is repositioned in turn until the energy goes below some threshold.

Synopsis

Problemclass Maintenance of Knowledge Bases.

Problem Solving Method No generic computer-automated method available.

Source Model Structure model $S\langle F, \mathcal{M}\rangle$ of a resource-based model, defined over $\langle F, \mathcal{M}\rangle$ (cf. Section B.1, Page 136).

- *Fidelity Level F.* Aggregated physical properties.
- *Granularity Level $\mathcal{M}$.* Definition of components, $\mathcal{O} \subset \mathcal{M}$, and resources, $\mathcal{R} \subset \mathcal{M}$. $\mathcal{O} \cap \mathcal{R} = \emptyset$, $\mathcal{O} \cup \mathcal{R} = \mathcal{M}$.
- *Order of Magnitude of the Application.* $|\mathcal{O}| \approx 100$ and $|\mathcal{R}| \approx 200$; the size of a particular resource $R \in \mathcal{R}$ varies from 1 (specialty) to $|\mathcal{O}|$ (ubiquity). On average, resource size and component size are of the same order of magnitude and smaller than 20.

Envisioned Model Graph layout of the clustered component graph of $S\langle F, \mathcal{M}\rangle$.

- *Clustering.* Definition of equivalence classes on the elements in $\mathcal{O}$, which correspond to subsystems or modules.
- *Layout.* Clear arrangement of clusters and drawing of all components and connections. Graphical emphasis of different component connections, which may be faded in or out.
- *Subsequent Tasks Enabled.* Creation of new knowledge bases, comparative analysis of knowledge bases, maintenance of components, technical analysis, identification of delivery bottlenecks.

Knowledge Source Classification of resources respecting connection strengths. Clustering of component graph by Λ-maximization due to MAJORCLUST; drawing by a circular layout algorithm. Identification of equal clusters (modules) by a graph-theoretical similarity assessment.

D.3 Analyzing the Structure of Fluidic Systems

The analysis and design of fluidic control systems is a complex and time-consuming task that, at the moment, cannot be automated completely. Nevertheless, important subtasks like simulation or control concept selection can be efficiently supported by a computer. Prerequisite for a successful support is a well-founded analysis of a fluidic system's functional structure. The functional structure shows the fundamental modes of action of a circuit by isolating the different tasks (functions) the plant has to fulfill. It represents some kind of qualitative system description. Key elements within the functional structure are (1) the so-called "fluidic axes", defined below and exemplified in Figure D.18, and (2) the couplings between several axes, which are introduced later on.

Definition D.10 (Fluidic Axis) *A fluidic axis both represents and fulfills a function D of an entire fluidic plant. The axis defines the connections and the interplay among the working, control, and supply elements that realize D.*[3]

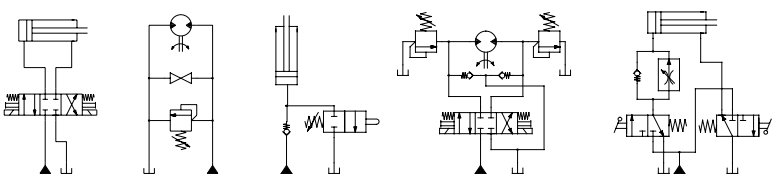


Figure D.18. The figure depicts drawings of five fluidic (hydraulic) axes for different functions and of different complexity.

This section shows how the functional structure of a fluidic system can be rendered automatically, that is to say, envisioned. The envisioning process relates to graph theory in first place and is sketched out in Figure D.19.

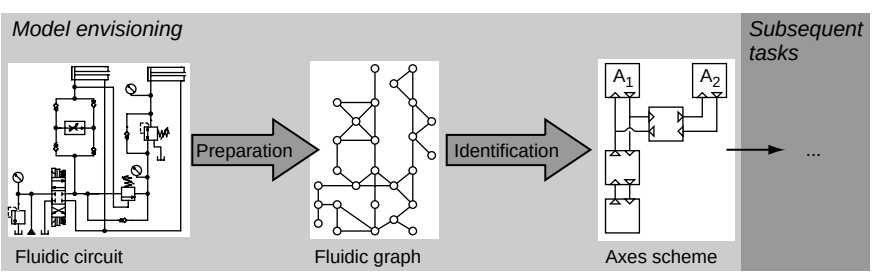


Figure D.19. Model envisioning of a fluidic circuit means to render the functional structure. It is attained by identifying the circuit's axes along with their couplings in the fluidic graph.

[3] The same definition is also given in Section A.1, which deals with the automated design of fluidic systems.

In this place we do not engage into the benefits of a structural analysis; related information can be found in (271, 288). Also note that the examples used here are taken from the hydraulic domain, but the envisioning methodology can be applied to pneumatic systems as well.

Underlying Models and Coupling Types

Definition D.11 (Corresponding Graph of a Fluidic System) *Let S be a fluidic system and let $\langle F, \mathcal{M} \rangle$ be a model of S. The corresponding (fluidic) graph $G(S)$ of S is a structure model $\langle V, E, \sigma \rangle$ over $\langle F, \mathcal{M} \rangle$; it is defined as follows.*

(1) F is the set of variables decribing physical properties of S, such as pressures, flows, velocities, forces, etc.

(2) $\mathcal{M}$ is the set of fluidic components used within S and can be partitioned into four sets, $\mathcal{W}$, $\mathcal{C}$, $\mathcal{S}$ and $\mathcal{A}$. $\mathcal{W}$ comprises the working elements such as cylinders and motors, $\mathcal{C}$ comprises the control elements, i. e., the directional valves, $\mathcal{S}$ comprises supply elements like pumps and tanks, and $\mathcal{A}$ comprises auxiliary elements such as filters.

(3) $\sigma : V \to \{w, c, s, a, i\}$ is a labeling function. It assigns each node $v \in V$ a label from $\{w, c, s, a\}$, indicating the membership of its related component $M_v \in \mathcal{M}$. If two components share a control functionality, the associated edge in E is labeled i (information exchange). Note, since $G(S)$ is a graph of the model $\langle F, \mathcal{M} \rangle$ there is a bijective mapping between $\mathcal{M}$ and V (cf. Definition 2.1, Page 26).

Figure D.20 contrasts a hydraulic circuit with its corresponding graph.

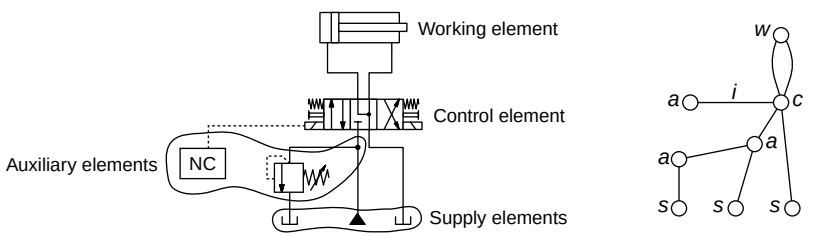


Figure D.20. Hydraulic circuit (left) and its fluidic, say, hydraulic graph (right). Control lines are shown dashed.

A Hierarchy of Coupling Types The recognition of the fluidic axes does not suffice to fully analyze a system. Within the diagnosis context, for instance, knowledge about the relationships between individual axes is essential for a precise statement concerning a faulty component, since a defect within one axis may spread its faulty behavior throughout the entire plant. Similar considerations apply to simulation tasks, design problems, or the development of a control strategy for a crucial working element. Thus, it is necessary to investigate the interdependences between the axes within a functional circuit analysis. The following coupling types have been worked out by Stein and Vier (271).

- *Type 0 (No Coupling).* Fluidic axes possess no coupling if there is neither a power nor an informational connection between them.
- *Type 1 (Informational Coupling).* Fluidic axes which are connected only by control connections are called informationally coupled. Notice that control lines can be realized by means of electrical, hydraulic, or pneumatic lines.
- *Type 2 (Parallel Coupling).* Fluidic axes which possess their own access to a common power supply are coupled in parallel.
- *Type 3 (Series Coupling).* A series coupling connects fluidic axes whose power supply or disposal is realized via the preceding or following axis.
- *Type 4 (Sequential Coupling).* A sequential coupling is given, if the performance of a following axis depends on the state variables, e. g. the pressure or the position of the preceding one in order to work in a sequence.

In order to facilitate an automatic classification of the above coupling types, a precise mathematical formulation must be developed. It is given in the following definition and relies on the concept of fluidic graphs.

Definition D.12 (Coupling Types) *Given is a model $\langle F, \mathcal{M} \rangle$ of a fluidic system S containing two subcircuits, A, B, that realize two different fluidic axes. Let $G(S) := \langle V, E, \sigma \rangle$ be a fluidic graph of S, and let $G(A) := \langle V_A, E_A \rangle$ and $G(B) := \langle V_B, E_B \rangle$ be subgraphs of $G(S)$ corresponding to the axes A and B.*

- *Type 0 (No Coupling). If $G(S)$ is not connected, and if $G(A)$ and $G(B)$ are subgraphs of different connected components in G, then A and B are not coupled.*
- *Type 1 (Informational Coupling). Let $E' = \{e \mid \sigma(e) = i\}$ be a set of edges associated with control lines in S. If $G' := \langle V, E \setminus E' \rangle$ is not connected, and if $G(A)$ and $G(B)$ are subgraphs of different connected components in G', then A and B are informationally coupled.*
- *Type 2 (Parallel Coupling). Let $v_a \in V_A$ and $v_b \in V_B$, $v_a \neq v_b$, $M_{v_a}, M_{v_b} \in \mathcal{C}$, i. e., v_a and v_b are associated with control elements. Moreover, let $v_s \in V$ be associated with a supply element $M_{v_s} \in \mathcal{S}$. Then A and B are coupled in parallel if for each node v_w, associated with a working element $M_{v_w} \in \mathcal{W}$, a path $P := \{v_s, \ldots, v_w\}$ exists such that the following conditions hold.*

(1) $P \cap \{v_a, v_b\} = \{v_a\} \Rightarrow v_w \in V_a$

(2) $P \cap \{v_a, v_b\} = \{v_b\} \Rightarrow v_w \in V_b$

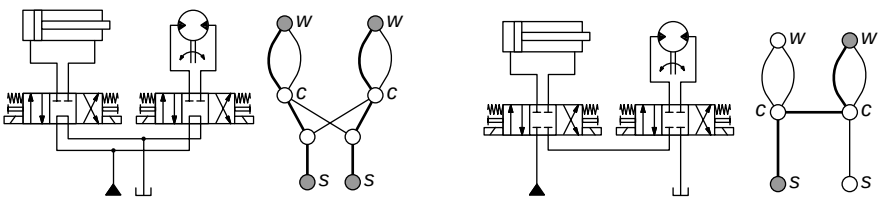


Figure D.21. Circuit with two axes coupled in parallel (left) and circuit containing hydraulic axes coupled in series (right).

- *Type 3 (Series Coupling). Let* $v_a \in V_A$ *and* $v_b \in V_B$, $v_a \neq v_b$, $M_{v_a}, M_{v_b} \in \mathcal{C}$, *i. e.,* v_a *and* v_b *are associated with control elements. Moreover, let* $v_s \in V$ *be associated with a supply element* $M_{v_s} \in \mathcal{S}$. *Then* A *and* B *are coupled in series if there exists a node* v_w, *associated with a working element* $M_{v_w} \in \mathcal{W}$, *such that for each path* $P := \{v_s, \ldots, v_w\}$ *the following condition holds.*

  (1) $v_a \in P \Rightarrow v_b \in P$.

- *Type 4 (Sequential Coupling). Let be* $V_A \neq V_B$. A *and* B *are sequentially coupled if* A *and* B *have no coupling of type* $0, \ldots, 3$.

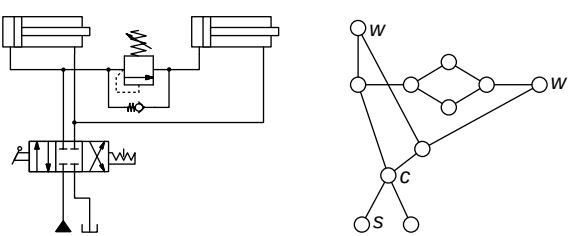


Figure D.22. Circuit containing sequentially coupled hydraulic axes.

Remarks. The engineering point of view: Axes of type 0 don't have any physical connection and can be investigated independently. Axes coupled in parallel are controlled by their own control elements; if axes are coupled in series, at least one axis controls the flow of all other axes (cf. Figure D.21).

Envisioning the Functional Structure

Due to their inherent structure, the recognition of fluidic axes and their coupling types within a system S can be solved with path search algorithms on $G(S)$: Each axis needs a pump, representing a pressure source, some valves along with additional auxiliary components for control purposes, and cylinders and motors which represent the working devices responsible for the output. However, fluidic axes typically contain substructures that hinder a straightforward detection, and a three-step envisioning procedure has been developed, consisting of graph preprocessing, axes identification, and coupling type determination. Details may be found in (267).

Graph Preprocessing To reduce the complexity of $G(S)$—but, in first place, to make axes identification possible at all, $G(S)$ is simplified by means of merging, deletion, and contraction rules. Figure D.23 illustrates the application of such rules.

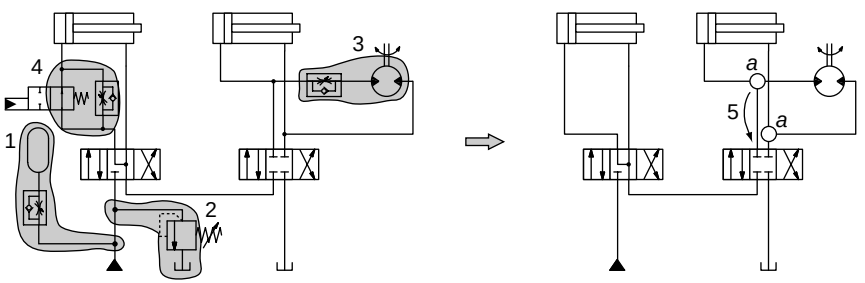


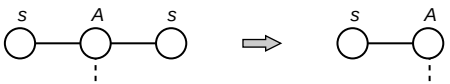
Figure D.23. Simplification of a circuit by expanding T-junctions, stripping off branches, and deleting loops. The left-hand side shows places where contraction rules are applied, the right-hand side shows the use of merging rules.

The following design graph grammar provides a means to accomplish the preprocessing of $G(S)$. Let $\mathcal{G} = \langle \Sigma, P \rangle$ be a design graph grammar for the structural simplification of fluidic circuits where $\Sigma = \{a, c, s, A, B, C\}$, $P := P_{\text{contraction}} \cup P_{\text{merging}}$ is the set of graph transformation rules, $T \rightarrow \langle R, I \rangle$, and is specified in the following.

(1,2) *Contraction of Sequences.* Contractible sequences divide into dead branches and into auxiliary components connected in series with working or control elements (cf. Figure D.23, Case 1 – 3 respectively).

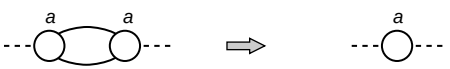
$$T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1, 2^\star, 3\}, \{\{1,2\}, \{2,3\}\}, \{(1,A), (2,a), (3,B)\} \rangle$$
$$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5\}, \{\{4,5\}\}, \{(4,A), (5,B)\} \rangle$$
$$I = \{((C,A),(C,A)), ((C,B),(C,B))\}$$

$T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1^\star, 2, 3^\star\}, \{\{1,2\},\{2,3\}\}, \{(1,s),(2,A),(3,s)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{4,5\}, \{\{4,5\}\}, \{(4,s),(5,A)\}\rangle$
$I = \{((B,A),(B,A))\}$



(3) *Contraction of Loops.* A circuit may contain cyclic structures or components connected in parallel. These structures are not necessary for detection purposes if they neither contain nor control a working element (cf. Figure D.23, Case 4).

$T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2\}, \{\{1,2\},\{1,2\}\}, \{(1,a),(2,a)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{3\}, \{\}, \{(3,a)\}\rangle$
$I = \{((A,a),(A,a))\}$



(4) *Merging with Control Elements.* The merging of auxiliary elements with control elements is applied to auxiliary nodes of degree 3 and shown as Case 5 on the right-hand side of Figure D.23.

$T = \langle V_T, E_T, \sigma_T \rangle = \langle \{1,2\}, \{\{1,2\}\}, \{(1,c),(2,a)\}\rangle$
$R = \langle V_R, E_R, \sigma_R \rangle = \langle \{3\}, \{\}, \{(3,c)\}\rangle$
$I = \{((A,c),(A,c)),((A,a),(A,c))\}$

Figure D.24 shows a part of a hydraulic circuit, the corresponding graph, and the application of the two sequence contraction rules.

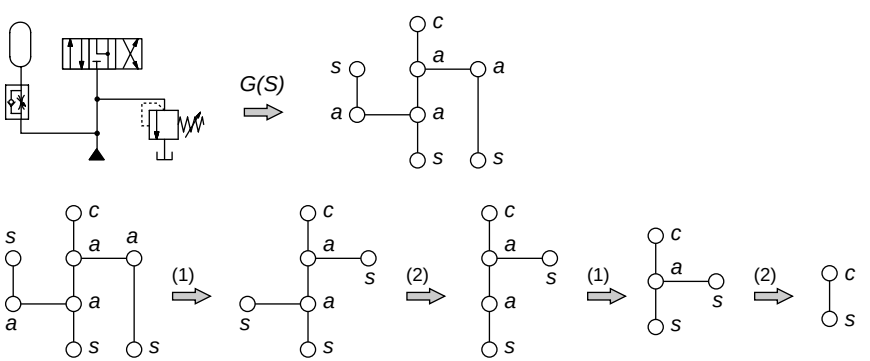


Figure D.24. Application of the sequence contraction rules for the circuit of Figure D.23.

Remarks. The runtime complexity in the preprocessing is dominated by the algorithm for loop detection, which can be assessed with $\mathcal{O}(|E|)$ (177).

Axes Identification Identifying a fluidic axis means to search for a set of nodes in the fluidic graph $G(S)$ whose counterpart in the circuit realizes a particular function. Each such set must contain a node labeled s and one or more nodes labeled w. Moreover, all components that also belong to the axis must lie on some path between the working and the supply element. This observation suggests to employ Dijkstra's and Floyd's algorithms (177, 68) to investigate in the preprocessed fluidic graph all shortest paths between nodes corresponding to supply elements and nodes corresponding to working elements.

Each shortest-path run labels the edges in the form of a directed tree, which encodes a successor relationship between the nodes. Nodes that lie on the same path in the directed tree define the components that belong to the same axis. The time complexity for this axes identification procedure is $\mathcal{O}(|V|^2 \cdot |E|)$ and is discussed in (271).

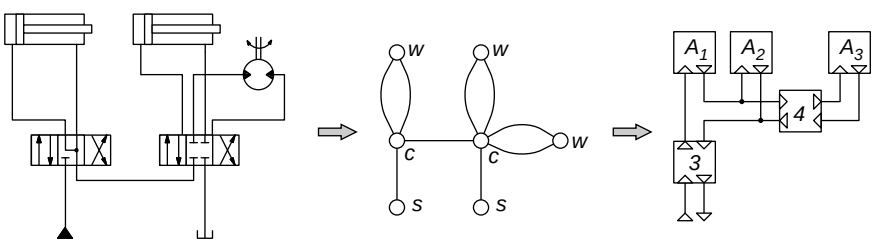


Figure D.25. The preprocessed circuit (left), its corresponding graph (middle), which is used to indentify axes and couplings in order to construct the axes coupling scheme (right).

Coupling Type Analysis The analysis of coupling types requires the comparison of supply paths between the working elements of the axes. If a circuit contains exactly two axes, the coupling type can be classified with a search effort of $\mathcal{O}(|E|)$. Given a circuit with n axes, possible couplings between all axes pairs must be analyzed. Using a naive approach, the above search effort is carried out $\binom{n}{2} \in \mathcal{O}(n^2)$ times. If, on the other hand, a circuit contains a lot of axes of only one coupling type, a linear number of comparisons is sufficient.

In this connection the transitivity property of coupling types can be exploited. E. g., given three axes, A_1, A_2, and A_3 and information on the couplings $\kappa(A_1, A_2)$ and $\kappa(A_1, A_3)$, $\kappa \in \{0, \ldots, 4\}$. Then for the third coupling holds that $\kappa(A_1, A_2) \geq \min\{\kappa(A_1, A_2), \kappa(A_1, A_3)\}$. Stated another way, a weaker coupling is not possible since the axes A_1 and A_3 are coupled indirectly via A_2 (cf. Figure D.26).

Remarks. The above determination procedure is simplified within some respects. It neglects, for instance, that identical axes must be identified as such and comprised to

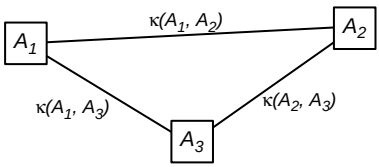


Figure D.26. The couplings between axes are not independent from each other.

one single axis. Identical axes are composed from the same components, they have an equivalent structure, and they are controlled by a single control element.

Case Study

When working with hydraulic engineering experts it became clear that a definition of the term "fluidic (hydraulic) axis" must stay imprecise up to a certain degree: The informal description "A fluidic axis realizes a subfunction of a fluidic system" leaves a scope of interpretation—e. g., regarding the components which actually must be count to an axis and which not. Thus a precise definition of the hydraulic axes analysis problem cannot be stated.

The consequences are: (1) A human expert has the final say whether or not the result of an analysis algorithm is correct. (2) The result of an analysis algorithm must not be absolutely correct or wrong but correct up to a certain degree.

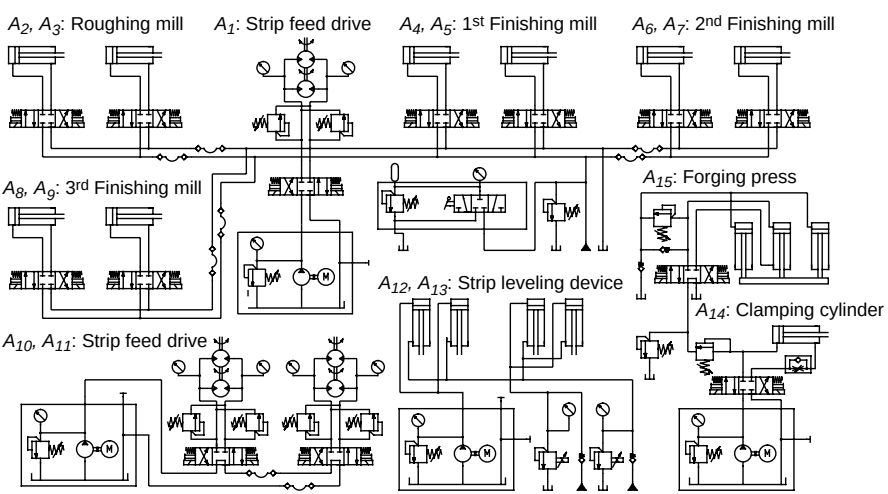


Figure D.27. Hydraulic circuit diagram of a cold-rolling plant.

We overcame the problems that result from this fuzziness by acquiring and en-

coding analysis knowledge direct from domain experts. Moreover, a library was built up to prove the quality of our algorithms, which contains 160 hydraulic circuits of different size and complexity from various engineering applications. In the circuits of this library more than 95% of the hydraulic axes are identified correctly by the algorithms. The solutions of the remaining cases is not entirely off the track but contain a small number of incorrectly assigned components.

Finally, as a representative example with respect to complexity and dimension, Figure D.27 shows the circuit diagram of a cold-rolling plant (299, 61). Here, more than 20 actuators work the coiled steel strips. Figure D.28 depicts the functional view in the form of the global coupling scheme.

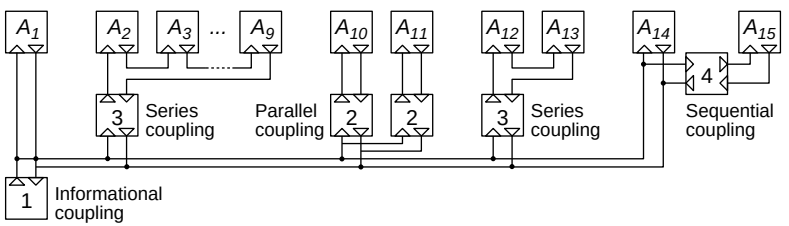


Figure D.28. The global coupling scheme of the above cold-rolling plant.

Synopsis

Problemclass Functional analysis of fluidic systems.

Problem Solving Method No generic computer-automated method available.

Source Model Structure model over a model $\langle F, \mathcal{M} \rangle$ of a fluidic system S.

- *Fidelity Level F.* Variables decribing physical properties of S, such as pressures, flows, velocities, or forces.
- *Granularity Level $\mathcal{M}$.* Fluidic components used within S.
- *Order of Magnitude of the Application.* Medium-sized up to large circuits, $|\mathcal{M}| \in [20, \ldots, 500]$.

Envisioned Model Isolated fluidic functions within the structure model.

- *Fluidic Axes.* A set of numbers $\{1,\ldots,k\}$, denoting the k axes found, and a mapping from the components in $\mathcal{M}$ onto $\mathcal{P}(\{1,\ldots,k\})$, the power set of $\{1,\ldots,k\}$.[4]
- *Coupling Scheme.* A rooted tree with k leafs, defining the coupling hierarchy between the k axes. The inner nodes are labeled with a coupling type number from $\{0,\ldots,4\}$.
- *Subsequent Tasks Enabled.* Demand formulation and interpretation, smart simulation, optimization, control concept selection and evaluation diagnosis, didactics.

Knowledge Source Definition of fluidic axes and coupling types by domain experts. Representative circuit library for test purposes. Graph-theoretical analysis of structure models of fluidic systems with respect to both the definitons and heuristic classification knowledge.

[4]Recall that a component can belong to several axes.

References

[1] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AICOM*, pages 39–59, 1994.

[2] M. Abderrahim and A. Whittaker. Mechatronics '98. In J. Adolfsson and J. Karlsen, editors, *Interfacing Autolev to Matlab and Simulink*, pages 897–901. Pergamon, September 1998.

[3] Simaan M. Abourizk, Jingsheng Shi, Brenda McCabe, and Dany Hajjar. Automating the Process of Building Simulation Models. In *Proceedings of the 1995 Winter Simulation Conference (WSC 95)*, Proceedings in Artificial Intelligence, pages 1032–1038, Arlington, VA, December 1995.

[4] A. Abu-Hanna and Y. Gold. An integrated, deep-shallow expert system for multi-level diagnosis of dynamic systems. In John S. Gero, editor, *Artificial Intelligence in Engineering: Diagnosis and Learning*, pages 75–94. Elsevier, Amsterdam, 1988.

[5] Sanjaya Addanki, Roberto Cremonini, and J. Scott Penberthy. Reasoning about Assumptions in Graphs of Models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1324–1330, Detroit, Michigan, August 1989.

[6] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington D. C., May 1993. ACM Press.

[7] David Aha. Case-Based Learning Algorithms. In Ray Bareiss, editor, *Proceedings of the Case-Based Reasoning Workshop 1991*, pages 147–158. Morgan Kaufmann, 1991.

[8] David Aha. Tolerating Noisy, Irrelevant, and Novel Attributes in Instance-Based Learning Algorithms. *International Journal of Man-Machine Studies*, 1992.

[9] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Massachusetts, 1983.

[10] Leona S. Aiken and Stephen G. West. *Multiple Regression*. Sage, New York, 1991.

[11] Martin Anantharaman, Bodo Fink, Martin Hiller, and Stefan Vogel. Integrated Development Environment for Mechatronic Systems. In *Proceedings of the Third Conference on Mechatronics and Robotics*, Paderborn, Germany, 1995.

[12] D. Angluin and C. H. Smith. Inductive Inference: Theory and Methods. *Computational Surveys*, 15(3):237–269, September 1983.

[13] V. Arvind, R. Beigel, and A. Lozano. The Complexity of Modular Graph Automorphism. In *Proceedings of the Fifteenth Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science, LNCS*, pages 172–182. Springer, 1998.

[14] L. Auslander and S. V. Parter. On Imbedding Graphs in the Plane. *Journal of Mathematics and Mechanics*, 10(3):517–523, May 1961.

[15] W. Backé and H. Murrenhoff. *Grundlagen der Ölhydraulik, Vorlesung*. IHP, RWTH Aachen, 1994.

[16] Klaus Backhaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber. *Multivariate Analysemethoden*. Springer, Berlin Heidelberg New York, 1999.

[17] Daniela Bailer-Jones. How can Mental Models, as a Model of Cognitive Reasoning, be Applied to Scientific Models? In *Proceedings of the Seventh Annual Meeting of the European Society for Philosophy and Psychology*, Lisbon, Portugal, September 1998.

[18] Thomas Bailey and John Cowles. Cluster Definition by the Optimization of Simple Measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, September 1983.

[19] R. Barletta and D. Hennessy. Case Adaptation in Autoclave Layout Design. In K. J. Hammond, editor, *Proceedings of the Case-Based Reasoning Workshop*, pages 203–207. Morgan Kaufmann, 1989.

[20] Martin Bauer, Benno Stein, and Jürgen Weiner. Problemklassen in Expertensystemen. *KI – Künstliche Intelligenz: Forschung, Entwicklung, Erfahrungen*, 3:13–18, September 1991.

[21] R. Beale and T. Jackson. *Neural Computing*. Institute of Physics, Bristol, Philadelphia, 1994.

[22] Peter Beater. *Entwurf hydraulischer Maschinen*. Springer, Berlin Heidelberg New York, 1999.

[23] Daniel G. Bobrow, editor. *Qualitative Reasoning about Physical Systems*. North-Hollamd, Amsterdam New York Oxford, 1984.

[24] Gert Böhme. *Fuzzy-Logik*. Springer, Berlin Heidelberg New York, 1993.

[25] A. Bonzano, P. Cunningham, and B. Smyth. Using Introspective Learning to Improve Retrieval in CBR: A case Study in Air Traffic Control. In *Proceedings of the Second ICCBR Conference*, Berlin Heidelberg New York, 1997. Springer.

[26] Heather Booth and Robert E. Tarjan. Finding the Minimum-Cost Maximum Flow in a Series–Parallel Network. *Journal of Algorithms*, 15:416–446, 1993.

[27] Elisabeth Bradley and Reinhard Stolle. Automatic Construction of Accurate Models of Physical Systems. *Annals of Mathematics and Artificial Intelligence*, 17 (1-2):1–28, 1996.

[28] Franz J. Brandenburg. On the Complexity of the Membership Problem of Graph Grammars. In Manfred Nagl and Jürgen Perl, editors, *Graphtheoretic Concepts in Computer Science*, pages 40–49, Linz, 1983. Trauner.

[29] Franz J. Brandenburg. Layout Graph Grammars: The Placement Approach. In Hartmut Ehrig, editor, *Graph Grammars and Their Application to Computer Science*, number 532 in Lecture Notes in Computer Science, LNCS, pages 144–156, Berlin Heidelberg New York, 1991. Springer.

[30] Franz J. Brandenburg. Designing Graph Drawings by Layout Graph Grammars. In Roberto Tamassia and Ioannis G. Tollis, editors, *Proceedings of the DIMACS International Workshop of Graph Drawing (GD 94), Princton*, number 894 in Lecture Notes in Computer Science, LNCS, pages 416–427, Berlin Heidelberg New York, 1994. Springer.

[31] Andreas Brandstaedt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes—A Survey*. Society for Industrial and Applied Mathematics, New York, 1999.

[32] Joost Breuker. Model-Driven Knowledge Acquisition: Interpretation Models. Technical Report Exprit Project 1098, Memo 87, Department of Social Science Informatics, University of Amsterdam, 1987.

[33] Axel Brinkop, Norbert Laudwein, and Rüdiger Maassen. Routine Design for Mechanical Engineering. In *Proceedings of the Sixth Annual Conference on Innovative Applications of AI (IAAI 94)*, Seattle, August 1994.

[34] David C. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann, 1989.

[35] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The* MYCIN *Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Massachusetts, 1984.

[36] J. G. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: an Artificial Intelligence Approach*, volume 2, pages 371–392. Morgan Kaufmann, 1986.

[37] François E. Cellier. *Continuous System Simulation*. Springer, Berlin Heidelberg New York, 1991.

[38] François E. Cellier, Hilding Elmqvist, and Martin Otter. Modeling from Physical Principles. In W.S. Levine, editor, *The Control Handbook*, pages 99–108. CRC Press, Boca Raton, FL, 1995.

[39] B. Chandrasekaran and Rob Milne. Reasoning About Structure, Behavior, and Function. *SIGART Newsletter*, Juli 85(93):4–59, 1985.

[40] B. Chandrasekaran, Michael C. Tanner, and John R. Josephson. Explanation: The Role of Control Strategies and Deep Models. In James A. Hendler, editor, *Expert Systems: The User Interface*, pages 219–247. Ablex Publishing Corporation, 1987.

[41] William J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[42] Daniel J. Clancy and Benjamin Kuipers. Model Decomposition and Simulation. In Toyoaki Nishida, editor, *Proceedings of the Eighth International Workshop on Qualitative Reasoning about Physical Systems (QR 94)*, pages 45–54, Nara, Japan, June 1994.

[43] Anthony G. Cohn. Qualitative Reasoning. In R. T. Nossum, editor, *Proceedings of the ACAI'87*, pages 61–95, Nara, Japan, 1987.

[44] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-Rewriting Hypergraph Grammars. *Journal of Computer and System Sciences*, 46:218–270, 1993.

[45] R. H. Creecy, B. M. Masand, S. J. Smith, and D. I. Waltz. Trading Mips and Memory for Knowledge Engineering. *Communications of the ACM*, 35, 1992.

[46] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—An Approach to Domain-Independent Construction. Technical Report 21, Department of Computer Science, University of Hamburg, Hamburg, Germany, March 1989.

[47] Daniel Curatolo. *Wissensbasierte Methoden zur effizienten Simulation fluidtechnischer Systeme*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1996.

[48] Jonny Carlos da Silva and David Dawson. The Development of an Expert System for Hydraulic Systems Design Focusing Concurrent Engineering Aspects. In *Proceedings of the International Conference on Enginieering Design (ICED 97)*, 1997.

[49] R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Revohot, 1989.

[50] B. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Dissertation, University of Utrecht, Netherlands, 1997.

[51] Johan de Kleer and Brian C. Williams. Diagnosing Multiple Faults. *Automated Reasoning*, pages 372–388, 1987.

[52] Rina Dechter and Judea Pearl. The Anatomy of Easy Problems: A Constraint Satisfaction Formulation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI 85)*, Los Angeles, California, 1985.

[53] Rina Dechter and Judea Pearl. The Cycle-Cutset Method for Improving Search Performance in AI Applications. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, Orlando, Florida, 1987.

[54] Johan deKleer and Brian C. Williams. Diagnoses with Behavioral Models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1324–1330, Detroit, Michigan, August 1989.

[55] G. Di Battista, P. Eades, Roberto Tamassia, and I. G. Tollis. *Graph Drawing—Algorithms for the Visualization of Graphs*. Prentice-Hall, New York London Tokyo, 1999.

[56] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202, Menlo Park, CA, July 1995. Morgan Kaufmann.

[57] Oskar Dressler and Peter Struss. The Consistency-based Approach to Automated Diagnosis of Technical Devices. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 267–311. CSLI, Stanford, 1996.

[58] Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge Replacement Graph Grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 95–162. World Scientific, Singapore, 1997.

[59] John Durkin. *Expert Systems: Design and Development*. MacMillan, 1998.

[60] P. Eades. A Heuristic for Graph-Drawing. *Congressus Numerantium*, 42:149–160, 1984.

[61] H. Ebertshäuser. Grundlagen der hydraulischen Schaltungstechnik. *O+P Ölhydraulik und Pneumatik*, 38(10):604–607, 1994.

[62] Marko van Eekelen, Sjaak Smetsers, and Rinus Plasmeijer. Graph Rewriting Systems for Functional Programming Languages. Technical report, Computing Science Institute, University of Nijmegen, 1998.

[63] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2 Applications, Languages, and Tools. World Scientific, 1999.

[64] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3 Concurrency, Parallelism, and Distribution. World Scientific, 1999.

[65] Hilding Elmquist. Object-Oriented Modeling and Automated Formula Manipulation in Dymola. In *SIMS'93*, Kongsberg, Norway, June 1993. Scandinavian Simulation Society.

[66] Joost Engelfriet and Grzegorz Rozenberg. Node Replacement Graph Grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 1–94. World Scientific, Singapore, 1997.

[67] Larry Eshelman. MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic, Norwell, Massachusetts, 1988.

[68] Shimon Even. *Graph Algorithms*. Pitman, London, 1979.

[69] B. S. Everitt. *Cluster Analysis*. Eward Arnolds, New York, Toronto, 1993.

[70] H. Faatz. *Der Hydrauliktrainer*. Mannesmann Rexroth GmbH, Lohr a. Main, Lohr am Main, 1988.

[71] Brain Falkenhainer and Ken Forbus. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, 51:95–143, 1991.

[72] Alfred Fettweis. Digital Filter Structures Related to Classical Filter Networks. *Archiv für Elektronik und Übertragungstechnik*, 25(2):79–89, 1971.

[73] Alfred Fettweis. Wave Digital Filters: Theory and Practice. *Proceedings of the IEEE*, 74(2):270–327, February 1986.

[74] Alfred Fettweis and Gerald Hemetsberger. *Grundlagen der Theorie elektrischer Schaltungen*. Brockmeyer, 1995.

[75] Richard Fikes, T. Gruber, Yumi Iwasaki, A. Levy, and P. Nayak. How Things Work—Project Overview. Technical Report KSL-91-70, Knowledge Systems Laboratory, Computer Science Department, Stanford University, November 1991.

[76] Donald P. Finn and Padraig Cunningham. Physical Model Generation in Thermal Engineering Problems described by Partial Differential Equations. In Toyoaki Nishida, editor, *Proceedings of the Eighth International Workshop on Qualitative Reasoning about Physical Systems (QR 94)*, pages 90–97, Nara, Japan, June 1994.

[77] Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, and Gerry Stahl. Embedding Critics in Design Environments. *The Knowledge Engineering Review*, 8 (4):285–307, December 1993.

[78] Paul A. Fishwick. The Role of Process Abstraction in Simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 18:18–39, February 1988.

[79] Paul A. Fishwick. Extending Object-Oriented Design for Physical Modeling. *ACM Transaction on Modeling and Computer Simulation*, Special Issue on Model Specification and Representation, 1996.

[80] K. Florek, J. Lukaszewiez, J. Perkal, H. Steinhaus, and S. Zubrzchi. Sur la Liason et la Division des Points d'un Ensemble Fini. *Colloquium Methematicum*, 2, 1951.

[81] Kenneth D. Forbus and Johan de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, Massachusetts, 1993.

[82] Kenneth D. Forbus, Dedre Genter, and Keith Law. MAC/FAC: A Model of Similarity-Based Retrieval. *Cognitive Science*, 19:141–205, 1994.

[83] Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.

[84] Frederick K. Frantz. A Taxonomy of Model Abstraction Techniques. In *Proceedings of the 1995 Winter Simulation Conference (WSC 95)*, Proceedings in Artificial Intelligence, pages 1413–1420, Arlington, VA, December 1995.

[85] Eugene C. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the Association for Computing Machinery*, 29(11):24–32, November 1982.

[86] T. Fruchterman and E. Reingold. Graph-Drawing by Force-Directed Placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.

[87] Brian Gaines. General Systems Research: Quo Vadis. In *General Systems Yearbook*, volume 24, pages 1–9, 1994.

[88] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.

[89] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1997.

[90] Patrice O. Gautier and Thomas R. Gruber. Generating Explanations of Device Behavior Using Compositional Modeling and Causal Ordering. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI 93)*. AAAI Press, 1993.

[91] Derdre Gentner and Albert L. Stevens. *Mental Models*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[92] John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.

[93] John P. van Gigch. *System Design Modeling and Metamodeling*. Plenum Press, New York, 1991.

[94] A. K. Goel, S. Bhatta, and E. Stroulia. KRITIK: An Early Case-Based Design System. In M. L. Maher and P. Pu, editors, *Issues and Applications of Case-Based Reasoning in Design*, pages 87–132, Hillsdale, New Jersey, 1997. Lawrence Erlbaum Associates.

[95] A. K. Goel and B. Chandrasekaran. Use of Device Models in Adaption of Design Cases. In K. J. Hammond, editor, *Proceedings of the Case-Based Reasoning Workshop*, pages 203–207. Morgan Kaufmann, 1989.

[96] A. K. Goel, J. L. Kolodner, M. Pearce, R. Billington, and C. Zimring. Towards a Case-Based Tool for Aiding Conceptual Design Problem Solving. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, California, 1991. Morgan Kaufmann.

[97] Stefan Goldschmidt. Modellbildung am Beispiel von Starrkörpersystemen. Systematische Darstellung und Untersuchung von Software-Unterstützung. Study work, Department of Mathematics and Computer Science, University of Paderborn, Germany, November 1996.

[98] E. J. Golin and S. P. Reiss. The Specification of Visual Language Syntax. *Visual Languages*, 1:141–157, 1990.

[99] Martin Charles Golumbic. Algorithmic Aspects of Intersection Graphs and Representation Hypergraphs. *Graphs and Combinatorics*, 4:307–321, 1985.

[100] Martin Charles Golumbic. Interval Graphs and Related Topics. *Discrete Mathematics*, 55:113–243, 1985.

[101] Thomas R. Gruber. Model Formulation as a Problem Solving Task: Computer-assisted Engineering Modeling. *International Journal of Intelligent Systems*, 8(1): 105–127, 1992.

[102] Hans-Werner Güsgen. CONSAT—*A System for Constraint Satisfaction*. Dissertation, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, November 1987.

[103] Joachim Hartung. *Statistik*. Oldenbourg, München, 1999.

[104] F. Hayes-Roth, D. Waterman, and D. Lenat. *Building Expert Systems*. Addison Wesley Publishing Company, London, 1983.

[105] M. Heinrich and E. W. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems for Modular Components. In *Proceedings of the CAIA'91*, pages 257–264, 1991.

[106] D. Hennessy and D. Hinkle. Initial Results from Clavier: A Case-Based Autoclave Loading Assistent. In R. Bareiss, editor, *Proceedings of the Case-Based Reasoning Workshop*, pages 225–232. Morgan Kaufmann, 1991.

[107] Thorsten Hesse and Benno Stein. Hybrid Diagnosis in the Fluidic Domain. In E. Alpaydin and C. Fyfe, editors, *International ICSC Symposium on Engineering of Intelligent Systems (EIS 98)*, pages 893–899. ICSI Academic Press, February 1998. ISBN 3-906454-11-8.

[108] Sture Hägglund. Introducing Expert Critiquing Systems. *The Knowledge Engineering Review*, 8(4):281–284, December 1993.

[109] Thomas R. Hinrichs and Janet L. Kolodner. The Roles of Adaptation in Case-Based Design. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI 91)*, Cambridge, Massachusetts, 1991. MIT Press.

[110] Marcus Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1999.

[111] J. E. Hopcroft and R. E. Tarjan. Dividing a Graph into Triconnected Components. *SIAM Journal of Computing*, 2(3):135–158, September 1973.

[112] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, New York, 1989.

[113] N. Howe and C. Cardie. Examining Locally Varying Weights for Nearest Neighbor Algorithms. In *Proceedings of the Eleventh ICML*. Morgan Kaufmann, 1997.

[114] Eyke Hüllermeier. Approximating Cost Functions in Resource-Based Configuration. Notes in Computer Science tr-rsfb-98-060, Department of Mathematics and Computer Science, University of Paderborn, Germany, September 1998.

[115] Uwe Husemeyer. *Heuristische Diagnose mit Assoziationsregeln*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 2001.

[116] Edward Hyvönen. Constraint reasoning based on interval arithmetic: the tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.

[117] IEEE, March 1997.

[118] Yumi Iwasaki and B. Chandrasekaran. Design Verification through Function and Behavior-Oriented Representations: Bridging the Gap between Function and Behavior. In *Proceedings of the Second International Conference on Artificial Intelligence in Design*. Kluwer Academic, 1992.

[119] Yumi Iwasaki and Alon Y. Levy. Automated Model Selection for Simulation. Knowledge Systems Laboratory (KSL), QS93, 1993.

[120] Yumi Iwasaki and Chee Meng Low. Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes. Technical Report KSL 91-69, Knowledge Systems Laboratory (KSL), Computer Science Department, Stanford University, November 1991.

[121] Samuel L.S. Jacoby and Janusz S. Kowalik. *Mathematical Modeling with Computers*. Prentice-Hall, Englewood Cliffs, N.J., 1980.

[122] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, New York London Tokyo, 1988.

[123] M. Jambu. *Explorative Datenanalyse*. Gustav Fischer, 1992.

[124] J. N. R. Jeffers. *Modelling*. Chapman and Hall, London New York, 1982.

[125] S. C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32, 1967.

[126] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen*. BI Wissenschaftsverlag, Wien, 1990.

[127] Gary Kahn. MORE: From Observing Knowledge Engineers to Automating Knowledge Acquisition. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 7–35. Kluwer Academic, Norwell, Massachusetts, 1988.

[128] T. Kamada and S. Kawai. An algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31:7–15, 1989.

[129] Thomas A. Kane and David A. Levinson. *Dynamics: Theory and Application*. McGraw-Hill, New York, 1985.

[130] Werner Karbach, X. Tong, and Angi 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 the EKAW'88*, number 43 in GMD-Studie, pages 31/1–17, Sankt Augustin, 1988. GMD.

[131] Walter J. Karplus. The Spectrum of Mathematical Modeling and Systems Simulation. *Mathematics and Computers in Simulation*, XIX:3–10, 1977.

[132] Manfred Kaul. *Syntaxanalyse von Graphen bei Präzedenz-Graph-Grammatiken*. Dissertation, Department of Mathematics and Computer Science, University of Passau, Passau, 1986.

[133] Manfred Kaul. Practical Applications of Precedence Graph Grammars. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph Grammars and Their Application to Computer Science*, number 291 in Lecture Notes in Computer Science, LNCS, pages 326–342, Berlin Heidelberg New York, 1987. Springer.

[134] B.W. Kernighan and S. Lin. Partitioning Graphs. *Bell Laboratories Record*, January 1970.

[135] Changwook Kim and Tae Eui Jeong. HRNCE Grammars—A Hypergraph Generating System with an eNCE Way of Rewriting. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, number 1073 in Lecture Notes in Computer Science, LNCS, pages 383–396, Berlin Heidelberg New York, 1996. Springer.

[136] K. Kira and L. A. Rendell. A Practical Approach to Feature Selection. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scottland, 1992. Morgan Kaufmann.

[137] Yoshinobu Kitamura and Mitsuru Ikeda. Domain Ontology Design for Model-based Reasoning and its Evaluation. Technical Report AI-TR-96-1, Institute of Scientific and Indsutrial Research, Osaka University, Osaka, Japan, January 1996.

[138] Johan de Kleer. Problem Solving with the ATMS. *Artificial Intelligence*, 28: 197–224, 1986.

[139] Johan de Kleer and John Seely Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24:7–83, 1984.

[140] Hans Kleine Büning, Daniel Curatolo, and Benno Stein. Configuration Based on Simplified Functional Models. Notes in Computer Science tr-ri-94-155, Department of Mathematics and Computer Science, University of Paderborn, Germany, November 1994.

[141] Hans Kleine Büning, Daniel Curatolo, and Benno Stein. Knowledge-Based Support within Configuration and Design Tasks. In *2nd Biennial European Joint Conference on Engineering Systems Design and Analysis (ESDA 94)*, Lecture Notes in Artificial Intelligence, LNAI, pages 435–441, Berlin Heidelberg New York, July 1994. Springer.

[142] Renate Klempien-Hinrichs. Node Replacement in Hypergraphs: Simulation of Hyperedge Replacement and Decidability of Confluence. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, number 1073 in Lecture Notes in Computer Science, LNCS, pages 397–411, Berlin Heidelberg New York, 1996. Springer.

[143] G. J. Klir. *Architecture of Systems Complexity*. Sauders, New York, 1985.

[144] Achim Knoch and Michael Bottlinger. Expertensysteme in der Verfahrenstechnik – Konfiguration von Rührapparaten. *Chem.-Ing.-Tech.*, 65(7):802–809, 1993.

[145] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[146] Ina Koch. Enumerating All Connected Maximal Common Subgraphs in Two Graphs. *Theoretical Computer Science*, 250(1-2):1–30, 2001.

[147] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin Heidelberg New York, 1990.

[148] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California, 1993.

[149] Martin Korff. Application of Graph Grammars to Rule-Based Systems. In Hartmut Ehrig, editor, *Graph Grammars and Their Application to Computer Science*, number 532 in Lecture Notes in Computer Science, LNCS, pages 505–519, Berlin Heidelberg New York, 1991. Springer.

[150] Granino A. Korn and John V. Wait. *Digital Continuous-System Simulation*. Prentice-Hall, Englewood Cliffs, N.J., 1978.

[151] W. Kortuem. Trends in der Rechnersimulation im Hinblick auf die Systemdynamik Fahrzeug–Fahrweg. *VDI Berichte, Nr. 1219*, pages 169–203, 1995.

[152] Wolfgang Kowalk. *System, Modell, Programm*. Spektrum Akademischer Verlag, Heidelberg Berlin, 1996.

[153] Benjamin Kuipers. De Kleer and Brown's "Mental Models"—A Critique. Technical report, Department of Mathematics, Tufts University, Medford, Massachusetts, 1981.

[154] H. S. Kumar and C. S. Krishnamoorthy. A Framework for Case-Based Reasoning in Engineering Design. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 9(3):161–182, 1995.

[155] T. Laußermair and K. Starkmann. Konfigurierung basierend auf einem Bilanzverfahren. In *Sixth Workshop "Planen und Konfigurieren" (PUK 92)*, number FR-1992-001 in FORWISS, München, Germany, 1992.

[156] David B. Leake. Case-Based Reasoning: Issues, Methods, and Technology. In *Proceedings of the First International Conference on Case-Based Reasoning*, Sesimbra, Portugal, 1995.

[157] Frank Thomas Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Morgan Kaufmann, San Mateo, 1992.

[158] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, New York, 1990.

[159] Theodor Lettman and Ulf Dunker. AQUA *Benutzerhandbuch*, March 1998.

[160] Klaus Ulrich Leweling and Benno Stein. Hybrid Constraints in Automated Model Synthesis and Model Processing. In Susanne Heipcke and Mark Wallace, editors, *5th International Conference on Principles and Practice of Constraint Programming (CP 99), Workshop on Large Scale Combinatorial Optimisation and Constraints*, pages 45–56, Leamington Spa England, October 1999. Dash Associates.

[161] Ulrike Lichtblau. *Flußgrapggrammatiken*. PhD thesis, Universität Oldenburg, 1990.

[162] Ulrike Lichtblau. Recognizing Rooted Context-Free Flowgraph Languages in Polynomial Time. In Hartmut Ehrig, editor, *Graph Grammars and Their Application to Computer Science*, number 532 in Lecture Notes in Computer Science, LNCS, pages 538–548, Berlin Heidelberg New York, 1991. Springer.

[163] Zheng-Yang Liu and Arthur M. Farley. Shifting Ontological Perspectives in Reasoing about Physical Systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence(AAAI 91)*, pages 395–400, Anaheim, California, 1990. AAAI Press.

[164] R. S. H. Mah. *Chemical Process Structures and Information Flows*. Butterworths, 1990.

[165] Mary Lou Maher. Process Models for Design Synthesis. *AI Magazine*, pages 49–58, 1990.

[166] Mary Lou Maher, B. Balachandran, and D. M. Zhang. *Case-Based Reasoning in Design*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1995.

[167] Mary Lou Maher and Andres Gomez de Silva Garza. The Adaptation of Structural System Designs Using Genetic Algorithms. In *Proceedings of the International Conference on Information Technology in Civil and Structural Engineering Design: Taking Stock and Future Directions*, Glasgow, Scotland, August 1996.

[168] Mary Lou Maher and Andres Gomez de Silva Garza. Case-Based Reasoning in Design. *IEEE Expert*, 12(2), 1997.

[169] Mary Lou Maher and Pearl Pu, editors. *Issues and Applications of Case-Based Reasoning in Design*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1997.

[170] Sandra Marcus. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 81–123. Kluwer Academic, Norwell, Massachusetts, 1988.

[171] Sandra Marcus and John McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1989.

[172] Sandra Marcus, Jeffrey Stout, and John McDermott. VT: An Expert Elevator Designer that Uses Knowledge-based Backtracking. *AI Magazine*, pages 95–112, 1988.

[173] W. Marquardt. Trends in Computer-Aided Process Modeling. *Computers Chemical Engineering*, 20(6/7):591–609, 1996.

[174] Hugh Martin. *The Design of Hydraulic Components and Systems*. Ellis Horwood, 1995.

[175] Hans Jürgen Matthies. *Einführung in die Ölhydraulik*. Teubner, Stuttgart, 1995.

[176] John McDermott. Preliminary Steps Towards an Taxonomy of Problem-Solving Methods. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–266. Kluwer Academic, Norwell, Massachusetts, 1988.

[177] J. McHugh. *Algorithmic Graph Theory*. Prentice-Hall, New York London Tokyo, 1990.

[178] Klaus Meerkötter and Dietrich Fränken. Digital Realization of Connection Networks by Voltage-Wave Two-Port Adaptors. *Archiv für Elektronik und Übertragungstechnik*, 50:362–367, November 1996.

[179] Kurt Mehlhorn. *Data Structures and Algorithms*, volume 2 Graph Algorithms and NP-Completeness. Springer, Berlin Heidelberg New York, 1984.

[180] Merriam-Webster Publishing Company. *Merriam-Webster's Collegiate Dictionary, 10th Edition*. Merriam-Webster, Springfield, MA, 1998.

[181] B. T. Messmer and H. Bunke. Subgraph Isomorphism in Polynomial Time. Technical Report AM-95-003, Research Group of Computer Vision and Artificial Intelligence, University of Bern, 1995.

[182] Sven Meyer zu Eißen. Natürliche Graphpartitionierung am Beispiel von Aufgabenmodellen in Unternehmensnetzwerken. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn, Germany, 2000.

[183] Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.

[184] Boris Mirkin. *Mathematical Classification and Clustering*. Kluwer Academic, 1996.

[185] E. E. L. Mitchell and J. S. Gauthier. Advanced Continuous Simulation Language (ACSL). *Simulation*, 25:72–78, 1976.

[186] S. Mitchell Hedetniemi, E. J. Cockayne, and S. T. Hedetniemi. Linear Algorithms for Finding the Jordan Center and Path Center of a Tree. *Transportation Science*, 15:98–114, 1981.

[187] S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1395–1401, San Mateo, California, 1989.

[188] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics*. Freeman, New York, 1993.

[189] Jack Mostow. Toward Better Models of the Design Process. *AI Magazine*, 6: 44–56, 1985.

[190] D. J. Murray-Smith. *Continuous System Simulation*. Chapman & Hall, London, London New York Tokyo, 1995.

[191] Seshashayee S. Murthy and Sanjaya Addanki. Diagnoses with Behavioral Models. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI 87)*, pages 631–636, Seattle, Washington, July 1987. AAAI Press.

[192] Oliver Najmann and Benno Stein. A Theoretical Framework of Configuration. In Fevzi Belli, editor, *Proceedings of the Fifth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEAAIE 92)*, volume 604, pages 441–450, Berlin Heidelberg New York, June 1992. Springer.

[193] P. Panduring Nayak. *Automated Model Selection*. Phd thesis, Stanford University, 1992.

[194] P. Panduring Nayak. Causal Approximations. *Artificial Intelligence*, 70:277–334, 1994.

[195] P. Panduring Nayak. *Automated Modelling of Physical Systems*. Springer, Berlin Heidelberg New York, 1995.

[196] Bernhard Nebel. Plan Modification versus Plan Generation. In Alexander Horz, editor, *Seventh Workshop "Planen und Konfigurieren" (PUK 93)*, number 723 in Arbeitspapiere der GMD, 1993.

[197] U. A. Nickel, J. Niere, and A. Zündorf. Tool Demonstration: The FUJABA Environment. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 742–745. ACM Press, 2000.

[198] Oliver Niggemann. *Visual Data Mining of Graph-Based Data*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 2001.

[199] Oliver Niggemann and Benno Stein. A Meta Heuristic for Graph Drawing. In Vito Di Gesù, Stefano Levialdi, and Laura Tarantino, editors, *Working Conference on Advanced Visual Interfaces (AVI 00), Palermo, Italy*, pages 286–289, New York, May 2000. Association of Computing Machinery, ACM. ISBN 1-58113-252-2.

[200] Oliver Niggemann, Benno Stein, and Michael Suermann. On Resource-based Configuration: Rendering Component-Property Graphs. In Jürgen Sauer and Benno Stein, editors, *Twelfth Workshop "Planen und Konfigurieren" (PUK 98)*, number tr-ri-98-193 in Notes in Computer Science, pages 65–72. Department of Mathematics and Computer Science, University of Paderborn, Germany, April 1998.

[201] Oliver Niggemann, Benno Stein, and Jens Tölle. Visualization of Traffic Structures. In *To appear in the Proceedings of the IEEE International Conference on Communications, Helsinki*, Piscataway, NJ, June 2001. IEEE.

[202] N.J. Nilsson. *Principles of Artificial Intelligence*. Springer, Berlin Heidelberg New York, 1982.

[203] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, California, 1998.

[204] V. Oh, P. Langdon, and J. Sharpe. SCHEMEBUILDER: An Integrated Computer Environment for Product Design. In *Computer Aided Conceptual Design*, 1994.

[205] Martin Otter. *Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter*. PhD thesis, Institut für Robotik und Systemdynamik der DLR, Oberpfaffenhofen, Düsseldorf, 1995.

[206] Bernd Page. *Diskrete Simulation*. Springer, Berlin Heidelberg New York, 1991.

[207] Gerhard Pahl and Wolfgang Beitz. *Konstruktionslehre. Methoden und Anwendung*. Springer, Berlin Heidelberg New York, 1997.

[208] C. C. Pantelides. SPEEDUP—Recent Advances in Process Simulation. *Computers Chemical Engineering*, 12(7):745–755, 1988.

[209] Judea Pearl. *Heuristics*. Addison-Wesley, Massachusetts, 1984.

[210] Linda Petzold. Differential/Algebraic Equations are not ODE's. *SIAM Journal on Numerical Analysis*, 3(3):277–385, September 1994.

[211] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language. *Computers Chemical Engineering*, 15(1):53–72, 1991.

[212] Anita Pos, Pim Borst, Jan Top, and Gans Akkermans. Reusability of Simulation Models. *Knowledge-Based Systems*, 9:119–125, 1996.

[213] Frank Puppe. *Problemlösungsmethoden für Expertensysteme*. Springer, Berlin Heidelberg New York, 1990.

[214] Frank Puppe. *Systematic Introduction to Expert Systems, Knowledge Representations and Problem-Solving Methods*. Springer, Berlin Heidelberg New York, 1993.

[215] Lisa Purvis and Pearl Pu. An Approach to Case Combination. In *Proceedings of the Workshop on Adaptation in Case Based Reasoning, European Conference on Artificial Intelligence (ECAI 96)*, Budapest, Hungary, 1996.

[216] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[217] Olivier Raiman. Order of Magnitude Reasoning. *Artificial Intelligence*, 51:11–38, 1991.

[218] B. Raphael and B. Kumar. Indexing and Retrieval of Cases in a Case-Based Design System. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 10:47–63, 1981.

[219] A. Reckmann. Ähnlichkeitsmaße und deren Parametrisierung für die fallbasierte Diagnose am Beispiel einer medizinischen Anwendung. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1999.

[220] E. Reingold and J. Tilford. Tidier Drawing of Trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

[221] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[222] J. Rekers and A. Schürr. A Graph Grammar Approach to Graphical Parsing. Technical Report 95-15, Department of Computer Science, University of Leiden, 1995.

[223] Michael M. Richter. The Knowledge Contained in Similarity Measures, October 1995. Some remarks on the invited talk given at ICCBR'95 in Sesimbra, Portugal.

[224] Michel M. Richter. Introduction to CBR. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Weß, editors, *Case-Based Reasoning Technology. From Foundations to Applications*, Lecture Notes in Artificial Intelligence, LNAI, pages 1–15. Springer, Berlin Heidelberg New York, 1998.

[225] Jeff Rickel and Bruce Porter. Automated Modeling for Answering Prediction Questions: Selecting the Time Scale and System Boundary. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 93)*, pages 1191–1198, Cambridge, Massachusetts, 1994. AAAI Press.

[226] Raul Rojas. *Theorie der neuronalen Netze*. Springer, Berlin Heidelberg New York, 1993.

[227] Tom Roxborough and Arunabha. Graph Clustering using Multiway Ratio Cut. In Stephen North, editor, *Graph Drawing*, Lecture Notes in Computer Science, LNCS, Berlin Heidelberg New York, 1996. Springer.

[228] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1 Foundations. World Scientific, Singapore, 1997.

[229] Grzegorz Rozenberg and E. Welzl. Boundary NLC Graph Grammars—Basic Definitions, Normal Forms, and Complexity. *Information and Control*, 69:136–167, 1986.

[230] Stuart J. Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, N.J., 1995.

[231] Reinhard Sablowski and Arne Frick. Automatic Graph Clustering. In Stephan North, editor, *Graph Drawing*, Lecture Notes in Computer Science, LNCS, Berlin Heidelberg New York, 1996. Springer.

[232] Elisha Sacks. Piecewise Linear Reasoning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI 87)*, pages 655–659, Seattle, Washington, July 1987. AAAI Press.

[233] S. L. Salzberg. A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6, 1991.

[234] Warren S. Sarle. Neural Networks and Statistical Models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, pages 1538–1550, Cary, NC, 1994. SAS Institute Inc.

[235] Munehiko Sasajima, Yoshinobu Kitamura, Mitsuru Ikeda, and Shinji Yoshikawa. An Investigation on Domain Ontology to Represent Functional Models. In Toyoaki Nishida, editor, *Proceedings of Eighth International Workshop on Qualitative Reasoning about Physical Systems (QR 94)*, pages 224–233, Nara, Japan, June 1994.

[236] D. B. Schaechter, D. A. Levinson, and T. R. Kane. AUTOLEV *User's Manual*, 1991.

[237] Herbert Schlitt. *Regelungstechnik*. Vogel, Würzburg, 1993.

[238] Thomas Schlotmann. Formulierung und Verarbeitung von Ingenieurwissen zur Verbesserung hydraulischer Systeme. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1998.

[239] Sabine Schmitgen. *Räumliche Fragestellungen bei der Konfigurierung*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, December 1993.

[240] André Schulz. *On the Automatic Design of Technical Systems*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 2001.

[241] André Schulz and Benno Stein. On Automated Design of Technical Systems. Notes in Computer Science tr-ri-00-218, Department of Mathematics and Computer Science, University of Paderborn, Germany, December 2000.

[242] André Schulz, Benno Stein, and Annett Kurzok. On Automated Design in Chemical Engineering. In R. J. Howlett and L. C. Jain, editors, *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies*, pages 261–266, Piscataway, NJ, September 2000. IEEE.

[243] A. Schürr, A. Winter, and A. Zündorf. Visual Programming with Graph Rewriting Systems. In *Proceedings of the Eleventh International IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1995.

[244] Andy Schürr. Introduction to PROGRES, an Attribute-Graph-Grammar-Based Specification Language. In M. Nagl, editor, *Proceedings of the Fifteenth International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science, LNCS*, pages 151–165, Berlin Heidelberg New York, 1989. Springer.

[245] Andy Schürr. PROGRES: A VHL-Language Based on Graph Grammars. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proceedings of the Fourth International Workshop on Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science, LNCS*, pages 641–659, Berlin Heidelberg New York, 1991. Springer.

[246] Andy Schürr. Developing Graphical (Software Engineering) Tools with PROGRES. In *Proceedings of the ICSE*, pages 618–619. IEEE Computer Society Press, 1997.

[247] Andy Schürr. Programmed Graph Replacement Systems. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 479–546. World Scientific, Singapore, 1997.

[248] K. Seisenberger. Komprimierte Darstellung von planaren Graphen. Dissertation, University of Passau, 1991.

[249] Bart Selman and Henry Kautz. Knowledge Compilation Using Horn Approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 91)*, pages 904–909, Anaheim, California, 1991. AAAI Press.

[250] E. H. Shortliffe. *Computer Based Medical Consultations:* MYCIN. Elsevier, New York, 1976.

[251] A. O. Slisenko. Context-Free Grammars as a Tool for Describing Polynomial-Time Subclasses of Hard Problems. *Inf. Proc. Letters*, 14:52–56, 1982.

[252] P. H. A. Sneath. The Application of Computers to Taxonomy. *Journal Gen. Microbiology*, 17, 1957.

[253] Torsten Söderström and Petre Stoica. *System Identification*. Prentice-Hall, New York London Tokyo, 1989.

[254] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In H. V. Jagadish and I. S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Canada, June 1996. ACM Press.

[255] C. Stanfill and D. Waltz. Toward memory-based learning. *Communications of the ACM*, 29:1213–1228, 1986.

[256] Mark Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann, 1995.

[257] Benno Stein. *Functional Models in Configuration Systems*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, June 1995.

[258] Benno Stein. Optimized Design of Fluidic Drives: Objectives and Concepts. Notes in Computer Science tr-ri-97-189, Department of Mathematics and Computer Science, University of Paderborn, Germany, August 1996.

[259] Benno Stein. Supporting Hydraulic Circuit Design by Efficiently Solving the Model Synthesis Problem. In *International ICSC Symposium on Engineering of Intelligent Systems (EIS 98)*, pages 1274–1280, Canada, February 1998. ICSI Academic Press. ISBN 3-906454-11-8.

[260] Benno Stein and Daniel Curatolo. Model Formulation and Configuration of Technical Systems. In Jürgen Sauer, Andreas Günter, and Joachim Hertzberg, editors, *Tenth Workshop "Planen und Konfigurieren" (PUK 96)*, volume 3 of *Proceedings in Artificial Intelligence*, pages 56–70, Bonn, Germany, April 1996. Infix. ISBN 3-92037-97-1.

[261] Benno Stein and Daniel Curatolo. Selection of Numerical Methods in Specific Simulation Applications. In José Mira, Angel Pasqual del Pobil, and Moonis Ali, editors, *Proceedings of the Eleventh International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEAAIE 98)*, volume 1416 of *Lecture Notes in Artificial Intelligence, LNAI*, pages 918–927, Berlin Heidelberg New York, June 1998. Springer. ISBN 3-540-64574-8.

[262] Benno Stein, Daniel Curatolo, and Marcus Hoffmann. Simulation in FLUIDSIM. In Helena Szczerbicka, editor, *Workshop on Simulation in Knowledge-Based Systems (SIWIS 98)*, number 61 in ASIM Notes, Bremen, Germany, April 1998. Technical committee 4.5 ASIM of the GI.

[263] Benno Stein, Daniel Curatolo, and Hans Kleine Büning. Speeding up the Simulation of Fluidic Systems by a Knowledge-Based Selection of Numerical Methods. In *Euromech Colloquium 370 Synthesis of Mechatronic Systems*. University of Duisburg, Germany, September 1997.

[264] Benno Stein and Marcus Hoffmann. On Adaptation in Case-Based Design. In R. Parenti and F. Masulli, editors, *Third International ICSC Symposia on Intelligent Indsutrial Automation (IIA 99) and Soft Computing (SOCO 99)*, Canada, June 1999. ICSI Academic Press. ISBN 3-906454-17-7.

[265] Benno Stein and Oliver Niggemann. On the Nature of Structure and its Identification. In Peter Widmayer, Gabriele Neyer, and Stefan Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science, LNCS*, pages 122–134, Berlin Heidelberg New York, June 1999. Springer. ISBN 3-540-66731-8.

[266] Benno Stein, Oliver Niggemann, and Uwe Husemeyer. Learning Complex Similarity Measures. In Reinhold Decker and Wolfgang Gaul, editors, *Classification and Information Processing at the Turn of the Millenium (selected Papers from the 23th Annual Conference of the German Classification Society (GfKl), Bielefeld, March 1999)*, pages 254–263, Berlin Heidelberg New York, 2000. Springer. ISBN 3-540-67589-2.

[267] Benno Stein and André Schulz. Topological Analysis of Hydraulic Systems. Notes in Computer Science tr-ri-98-197, Department of Mathematics and Computer Science, University of Paderborn, Germany, July 1998.

[268] Benno Stein and André Schulz. Modeling Design Knowledge on Structure. In Gregor Engels, Andreas Oberweis, and Albert Zündorf, editors, *Proceedings of the Workshop "Modellierung 2001"*, volume P-1 of *Lecture Notes in Informatics, LNI*, pages 38–48, Bonn, March 2001. Gesellschaft für Informatik. ISBN 3-88579-330-X.

[269] Benno Stein and Elmar Vier. Computer-Aided Control Systems Design for Hydraulic Drives. In Luc Boullart, Mia Loccufier, and Sven Erik Mattsson, editors, *Proceedings of the Seventh IFAC Symposium on Computer-Aided Control Systems Design (CACSD 97)*, Gent, Belgium, April 1997. University of Gent and the Belgian Institute for Automatic Control (BIRA).

[270] Benno Stein and Elmar Vier. An Approach to Formulate and to Process Design Knowledge in Fluidics. In Nikos E. Mastorakis, editor, *Recent Advances in Information Science and Technology*, pages 237–242, London WC2H 9HE, October 1998. World Scientific Publishing Co. Pte. Ltd. ISBN 981-02-3644-1.

[271] Benno Stein and Elmar Vier. Structural Analysis in Control Systems Design of Hydraulic Drives. *Engineering Applications of Artificial Intelligence, EAAI*, 13(6): 741–750, December 2000.

[272] Benno Stein and Jügen Weiner. MOKON – Eine modellbasierte Entwicklungsplattform zur Konfiguration technischer Anlagen. Notes in Computer Science SM-DU-178, University of Duisburg, Germany, September 1990.

[273] Benno Stein and Jürgen Weiner. Model-Based Configuration. In Gerhard Friedrich and Franz Lackinger, editors, *Proceedings of the Seventh Conference of the Austrian Society for Artificial Intelligence (OEGAI 91), Workshop for Model-Based Reasoning*, pages 63–73, Vienna, Austria, 1991. Christian Doppler Laboratory for Expert Systems.

[274] G. Stephanopoulos, G. Henning, and H. Leone. MODEL.LA. A Modeling Language for Process Engineering—I. The Formal Framework. *Computers Chemical Engineering*, 14(1):813–846, 1990.

[275] Peter Struß. Model-Based Diagnosis—Progress and Problems. In *Proceedings of the International GI-Convention*, volume 3, pages 320–331, October 1989.

[276] Peter Struss. Multiple Models for Diagnosis. SPQR-Workshop on Multiple Models, FRG Karlsruhe, March 1991.

[277] Peter Struß and Oskar Dressler. "Physical Negation"—Integrating Fault Models into the General Diagnostic Engine. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1318–1323, 1989.

[278] Martin Sueper. *Effiziente Lösungsstrategien für ressourcenorientierte Konfigurierungsprobleme*. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1994.

[279] Michael Suermann. Wissensbasierte Modellbildung und Simulation von hydraulischen Schaltkreisen. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1994.

[280] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernectics*, 11(2), 1981.

[281] Roberto Tamassia. On Embedding a Graph in the Grid with the minimum Number of Bends. *SIAM Journal of Computing*, 16(3):421–444, 1987.

[282] Roberto Tamassia. Advances in the Theory and Practice of Graph Drawing. Technical report, Department of Computer Science, Brown University, Providence, Providence, Rhode Island, 1996.

[283] Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, June 1972.

[284] Robert E. Tarjan and J. Hopcroft. Finding the Triconnected Components of a Graph. Technical Report 140, Department of Computer Science, Cornell University, Ithaca, New York, 1972.

[285] Juha-Pekka Tolvanen. *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. Jyväskylä Studies in Computer Science, Economics and Statistics, 1998.

[286] Rolf Unbehauen. *Grundlagen der Elektrotechnik 1*. Springer, Berlin New York, 1994.

[287] Marcos Vescovi, Yumi Iwasaki, Richard Fikes, and B. Chandrasekaran. CFRL: A Language for Specifying the Causal Functionality of Engineering Devices. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI 93)*, pages 626–633. AAAI Press, 1993.

[288] Elmar Vier. *Automatisierter Entwurf geregelter Hydrostatischer Systeme*, volume 795 of *Fortschritt-Berichte VDI, Reihe 8*. VDI, Düsseldorf, 1999.

[289] Elmar Vier and Benno Stein. Knowledge-based Control Systems Design for Fluid Power Systems. In Nikos E. Mastorakis, editor, *Recent Advances in Information Science and Technology*, pages 231–236, London WC2H 9HE, October 1998. World Scientific Publishing Co. Pte. Ltd. ISBN 981-02-3644-1.

[290] Elmar Vier and Benno Stein. Modeling of Design Strategies for Hydraulic Control Systems. In M. H. Hamza, editor, *Proceedings of the Seventeenth IASTED International Conference on Modelling, Identification and Control (MIC 98)*, pages 213–216. ACTA Press, February 1998. ISBN 0-88986-248-6.

[291] Elmar Vier, Benno Stein, and Marcus Hoffmann. Strukturelle Formulierung von Anforderungen an hydrostatische Antriebe. Technical Report MSRT 8/97, Gerhard Mercator University of Duisburg, FB 7 MSRT, November 1997.

[292] Angi Voß and Werner Karbach. MODEL-K: KADS Grows Legs. In G. Schreiber, B. Wielinga, and J. Breuker, editors, KADS*: Knowledge Acquisition and Design Structuring*. Academic Press, 1992.

[293] J. Wallaschek. Modellierung und Simulation als Beitrag zur Verkürzung der Entwicklungszeiten mechatronischer Produkte. *VDI Berichte, Nr. 1215*, pages 35–50, 1995.

[294] Jürgen Weiner. *Aspekte der Konfigurierung technischer Anlagen*. Dissertation, University of Duisburg, Germany, 1991.

[295] Daniel S. Weld. The Use of Aggregation in Causal Simulation. *Artificial Intelligence*, 30:1–34, 1986.

[296] Daniel S. Weld and Johan deKleer. Multiple Ontologies and Automated Modeling. In Daniel S. Weld, editor, *Readings in Qualitative Reasoning about Physical Systems*, pages 481–483. Morgan Kaufmann, 1990.

[297] P. E. Wellstead. *Physical System Modelling*. Academic Press Inc., London New York, 1979.

[298] Stefan Wess. *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Entscheidungen*. Dissertation, University of Kaiserslautern, 1995.

[299] B. Wessling. Modern Design of Cold Rolling Mills for Stainless Steel. In *Technologies for the Enhancement of Rolling Mills and Processing Lines*, pages 71–78. Mannesmann Demag AG, Ratingen, 1995.

[300] B. J. Wielinga and J. A. Breuker. Models of Expertise. *Advances in Artificial Intelligence*, pages 497–509, 1987.

[301] D. Randall Wilson and Tony R. Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[302] T. Wonnacott and R. Wonnacott. *Regression: A Second Course in Statistics*. John Wiley & Sons, New York, 1981.

[303] D. R. Woods. Drawing Planar Graphs. Technical Report STAN-CS-82-943, Computer Science Department, Stanford University, 1981.

[304] Zhenyu Wu and Richard Leahy. An Optimal Graph Theoretic Approach to Data Clustering: Theory and its Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, November 1993.

[305] A. Wayne Wymore. *Systems Engineering for Interdisciplinary Teams*. John Wiley & Sons, New York, 1976.

[306] Kenneth Man-kam Yip. Model Simplification by Asymptotic Order of Magnitude Reasoning. *Artificial Intelligence*, 80:309–348, 1996.

[307] C. T. Zahn. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers*, C-20(1), 1971.

[308] Bernard P. Zeigler. Towards a Formal Theory of Modeling and Simulation: Structure Preserving Morphisms. *Association for Computing Machinery*, 19(4): 742–764, October 1972.

[309] Bernard P. Zeigler. *Multifaceted Modeling and Discrete Event Simulation*. Academic Press, New York, 1984.

[310] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation*. Academic Press, New York, 2000.

[311] G. Zinßmeister. DLayout of Trees with Attribute Graph Grammars. In *Proceedings of the International Workshop of Graph Drawing (GD 93)*, pages 99–102, 1993.

[312] Blaž Zupan. Optimization of Rule-Based Systems Using State Space Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):238–253, March 1998.