Liu, Sun, Loh, Lu and Lima (Eds.):
Advances of Computational Intelligence in Industrial Systems
Berlin, ISBN 978-3-540-78296-4, pp. 139-167, ©Springer 2008.

Model Construction for Knowledge-Intensive Engineering Tasks

Benno Stein

Bauhaus University Weimar
Faculty of Media, Media Systems
D-99421 Weimar
benno.stein@medien.uni-weimar.de

Summary. The construction of adequate models to solve engineerings tasks is a field of paramount interest. The starting point for an engineering task is a given system, S, or a set of systems, $\mathcal{S}$, along with a shortcoming of information, often formulated as a question:

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

If such an analysis or synthesis question shall be answered automatically, both adequate algorithmic models along with the problem solving expertise of a human problem solver must be operationalized on a computer. Often, the construction of an adequate model turns out to be the key challenge when tackling the engineering task. Model construction—also known as model creation, model formation, model finding, or model building—is an artistic discipline that highly depends on the reasoning job in question.

Model construction can be supported by means of a computer, and in this chapter we present a comprehensive view on model construction, characterize both existing and new paradigms, and give examples for the state of the art of the realization technology. Our contributions are as follows:

- In Section 2 we classify existing model construction approaches 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).
- Domain experts and knowledge engineers rarely start from scratch when constructing a new model; instead, they develop an appropriate model by modifying an existing one. Following this observation we analyzed various projects and classified the found model construction principles as model simplification, model compilation, and model reformulation. In Section 3 we introduce these principles as *horizontal modeling construction* and provide a generic characterization of each.
- Section 4 presents real-world case studies to show horizontal model construction principles at work. The underlying technology includes, among others, hybrid knowledge representations, case-based as well as rule-based reasoning, and machine learning.

Key words: automated model construction, diagnosis tasks, design tasks, sensible modeling

1 Introduction

"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 chapter, the interesting objects, A, are technical systems. The observer, B, is a domain expert who works on an engineering task, such as a diagnosis or design problem. The questions are embedded in a ternary way: they relate to the technical system, to the engineering task, and to the domain expert.

A system, S, can be considered as a clipping of the real 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. The process of developing a model for a system is called model formation, model creation, or modeling.

If we are given a system S along with a question about S, an answer can be found by performing tailored experiments with S. Experiments are purposeful excitations of S while observing its reactions or modifications, which are called behavior. Note that experimenting with a system is not possible if a question is of hypothetical nature and relates to an, up to now, non-existing system. And, even if S exists, various reasons can forbid experimenting with S: the system's reactions may be too slow or too fast to be observed, the experiment is too expensive or too risky to be executed, or the experiment's influence onto the system cannot be accepted. A common a way out is the creation of a model M from the system S and to execute the experiment on M (see Figure 1). Performing an experiment on a model is called simulation [21].

As shown in Figure 1, a model does not depend on the system as the only source of information but is formed purposefully, in close relation to the interesting question and/or the experiment to be performed on it. Under the problem-solving view of Artificial Intelligence this is common practice: the interesting questions (diagnosis or planning problems) prescribe the problem representation (the model) and yet the problem-solving method (the "simulation" algorithm) [31].

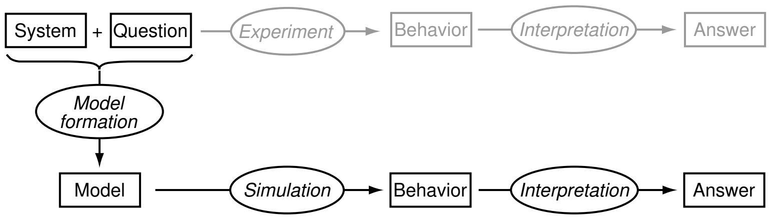


Fig. 1. Simulation as a concept to eliminate information deficits with respect to a system.

2 Top-Down Model Construction

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 ⇒ black-box model.
   This step relates to the modeling focus and serves two main purposes: complexity reduction and behavior isolation. In the field of engineering, focusing is realized among others by the well-known Method of Sections, typically used to cut out a part of a truss [17].
2. Identification of the subsystems and relations of S ⇒ structure model.
   This step relates to the structural granularity and defines all subsystems of S that can be identified in the model as such, and, at which level of detail the interplay between subsystems is represented. For this purpose the system's and subsystems' parameters along with input, output, and state variables are defined.
3. Characterization of constraints between variables ⇒ behavior model.
   This step relates to the modeling accuracy or modeling fidelity and determines the complexity of the mathematical relations that define the constraints between a model's state variables and algebraic variables. Both structural granularity and modeling fidelity define the modeling deepness.

The outlined modeling steps happen in our mind, and, a model at this stage is called a *mental model*. 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 that are stated in the third model formation step.

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 one encounters 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 is an experiment at a symbolic model and hence a simulation. We

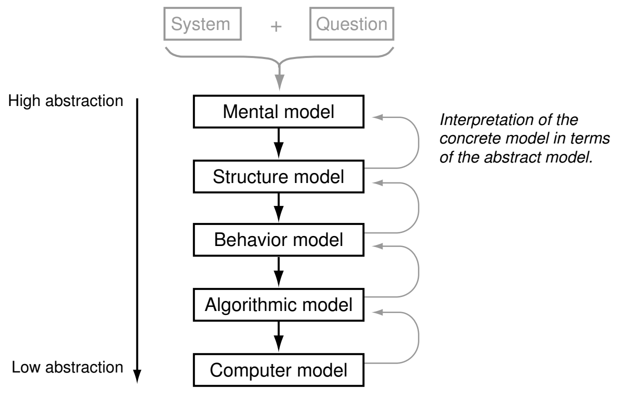


Fig. 2. A hierarchy of models. Top-down model construction means to map abstract models onto less abstract models. Final objective is the operationalization of the mental model in the form of a computer program.

call a model construction process that maps a model M onto a less abstract model *top-down model construction*; it is closely related to system analysis [20].

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 2, which shows a modified version from [47]). Typically, the human designer materializes his mental model as a textual or graphical structure model, for instance as a drawing or a diagram of the system S. The structure model, which defines subsystems and relations of S, becomes a behavior model of S by specifying a state prescription function for each subsystem. Behavior models are also designated as mathematical models [47]. Usually, 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. Finally, the specification of an algorithmic model within a computer language or a simulation language yields a computer model.

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 so-called association mappings [51]. As already mentioned, model construction needs not to be a vertical process; in Section 3 we consider model construction as a mapping between models at the same level.

Top-Down Model Construction Support: a Classification Scheme

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. We relate top-down approaches for model construction to the three areas shown in Figure 3, for which we state an abstract description, the known representatives, and a running example from the field of fluidic engineering.

1. *System + Question* $\rightarrow$ *Mental Model* $\rightarrow$ *Structure Model.* Support at this place relates to the model construction steps 1 and 2. Creating a mental model includes the problem of phenomena selection, which in turn is closely connected to the problem of identifying adequate system boundaries. Note that no sharp line can be drawn between mental models and graphical structure models.
   Known representatives: the model fragment idea for the organization of phenomena [29], the reasoning on hypothetical scenarios for the determination of system boundaries by [35], the CFRL paradigm that maps from function to behavior [18], the approaches to ontological reasoning [37], the case-based design approaches that map from demand specifications to technical devices [24].
   Example: For a hydraulic lifting platform the fail-save behavior for overload operation shall be investigated. This requires the analysis of maximum pressures and the functioning of relief valves. Hence, the system structure should represent the basic hydraulic building blocks such as pumps, pipes, cylinders, or valves.

2. *Structure Model → Behavior Model.* Support at this place relates to the model construction steps 2 and 3. 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.
   Known representatives: the model composition approach and its variants [5], the graphs of models paradigm [2], the use of model hierarchies in the field of diagnosis [45], the selection of local behavior models with respect to the desired accuracy for predefined phenomena [46].
   Example: For the given question the hydraulic components need only to be described by their stationary behavior, which includes continuity conditions, pressure drops according to Bernoulli, and the balance of forces.
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: continuity and compatibility constraint introduction as realized in FLUIDSIM and standardized in MODELICA [10], task-oriented selection of efficient numerical procedures, symbolic manipulation of equation systems such as BLT-partitioning or automatic tearing [23], minimum coordinate determination in kinematics computations, coherent synthesis of local behavior models [22], automation of Jordain's principle as realized in AUTOLEV [1].
   Example: The mathematical equations of the hydraulic components are collected, variables are unified according to the circuit structure, and a causal ordering is determined.

Remarks. The operationalization of model construction knowledge at the mental model level does not hit the mark in many modeling problems. Especially when

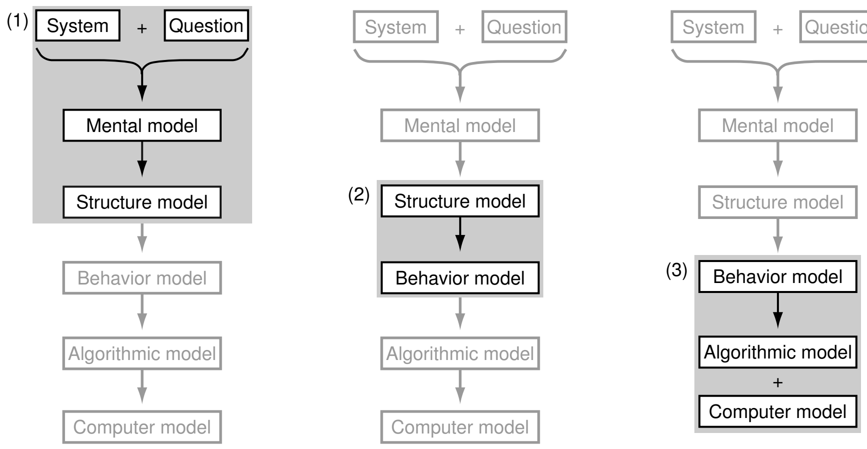


Fig. 3. The shaded areas indicate the three major places where top-down model construction is supported (from left to right): system boundaries and relevant phenomena, model deepness and model coherence, model synthesis and model processing.

diagnosing or configuring a 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 tackled by a computer at present.

Recall the three main places where top-down model construction can be supported, shown in Figure 3. According to this view we distinguish top-down model construction methods as follows.

1. *Structure-defining* methods derive necessary phenomena from the interesting question, structural information from necessary phenomena, or structural information from the desired function.
2. *Structure-filling* methods derive behavioral information from necessary phenomena, behavioral information from functional requirements, or behavioral information from structural information.
3. *Structure-propagating* methods derive algorithmic information from behavioral information.

This classification tells us a lot about how these methods work. Structure-propagating methods is a collective term for mathematical procedures that form a global algorithmic model by "propagating" the implications of the local behavior models along a given structure. Since structure-propagating methods operate on the mathematical model level, the largest part of them is domain-independent and task-independent. Structure-defining methods, as well as structure-filling methods, 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.

Example. A model of an analog low-pass 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.

The models in the libraries are tagged by qualifiers that encode knowledge about their applicability. 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 take different forms:

- In case-based design the qualifiers encode an abstract 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 [32]. Qualifier processing here relates to the computation of similarity measures and to case retrieval [34].
- Model fragment qualifiers as used by [29] encode relations among 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 NP-complete.

- 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. Qualifier processing here means assumption management.

Remarks. (1) 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. (2) 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 human support [44]. AUTOLEV [38] and the IMECH toolbox [3] 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 [16]. (3) 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.

3 Horizontal Model Construction

We now describe in which ways from a given model M, called the *source model*, a new model M' can be constructed, which is particularly suited to solve a given analysis or synthesis problem (see Figure 4). For instance, structure information extracted from a behavior model can 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. Our text deals with the following horizontal model construction principles:

- *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.

These principles are applied to overcome computational intractability, to improve runtime performance, to simplify model maintenance, and the like. At the end of the next paragraphs, the passage ”Formal description” provides details about the pros and cons of each principle, while Table 2 provides for a short characterization. Note that these principles are not complete, but they are—often subconsciously—applied in engineering practice. Moreover, both the compilation principle as well as the reformulation principle have not been described in an abstract or formal way so far. Other horizontal model construction methods include model refinement or model envisioning.

While model construction support as depicted in Figure 3 means moving down the model abstraction hierarchy, model construction that is based on source models operationalizes a horizontal mapping. A source model, M, which guides the model construction, establishes the primary knowledge source when building the new model M'. However, additional knowledge sources, which give answers to the following questions, are inevitable:

- *Model Simplification.* Which parts shall be stripped off?
- *Model Compilation.* Where is a hidden shortcut?
- *Model Reformulation.* How do the migration rules look like?

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 the interesting question then the top-down approaches of the previous section. 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*. The case studies presented in Section 4 contain new ideas for model construction, which pertain either to conceptual aspects or to the realized algorithms.

Methods for model construction based on source models transform a structure model, M_S, into another structure model, M'_S, and a behavior model, M_B, into another behavior model, M'_B:

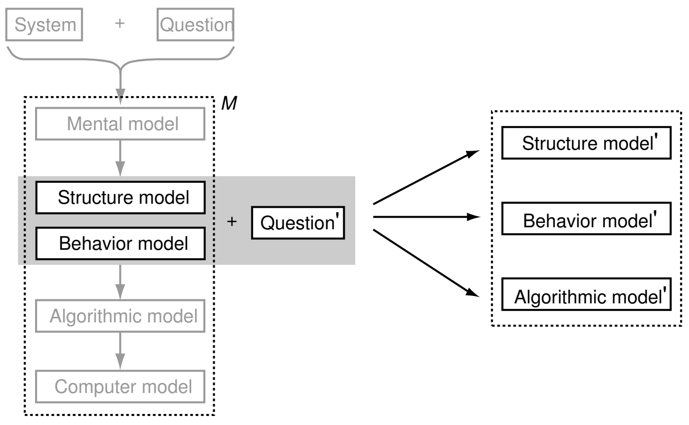


Fig. 4. Horizontal model construction uses a given source model to construct an adequate model regarding some question. Question' designates an analysis or synthesis task; the shaded area indicates the underlying knowledge source.

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

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: the effects of a structure model mapping, γ_S, and a behavior model mapping, γ_B, may range from a superficial variation up to an entire reformulation of the source model. Both functions address deficits of 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. To render horizontal model construction approaches comparable to each other they are characterized according to the properties listed in Table 1.

Property	**Semantics**
Characteristics	Modification of the structure model and the behavior model.
Modeling effects	Effects of a modification from the modeling perspective.
Processing effects	Effects of a modification with respect to model processing: 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	Problem classes, where the model construction approach is used.
Techniques	Techniques, algorithms, and strategies to implement the model construction approach; say, the functions γ_S and γ_B.

Table 1. Properties to characterize the model construction approaches.

Remarks. The functions γ_S and γ_B can be compared to the “system morphism” idea [49, 51]. System (homo-, iso-) morphisms are a concept to transform one system description into another. The main contribution in [49, 51] 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.

Model Simplification

Model simplification aims at a reduction of a model’s analytic complexity, 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 are crucial for 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 design problem. The concept of model simplification has been discussed by several researches before [12]. Formal description:

1. *Characteristics.* Simplification of behavior models usually happens within two respects. First, the set of functionalities is restricted to a subset, which leads to a simpler structure model (cf. Figure 5). Second, the complexity of the functions in the state prescription function are reduced. The former establishes an aggregation of function and directly leads to a structure simplification [14]; the latter falls into the class of behavior aggregation.
2. *Modeling Effects.* The lessened interaction between submodels results in a disregard of physical effects. The simplification results in a coarsening of physical phenomena or in physically wrong connections. The behavior is rendered inaccurately up to certain degree, but easier to understand [12].
3. *Processing Effects.* Both structure model and behavior model can be processed more efficiently; the processing difficulty is reduced; 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.
5. *Techniques.* γ_S: elimination of feedback loops; equalization of the node degree in the structure model; elimination of edges to create a causal ordering, i. e., a unique computation sequence when determining unknown functionalities [28].
   γ_B: fuzzification of equations the state prescription; piecewise linear reasoning [36]; linearization of higher order polynomials; balancing of dominant terms in complex equations [50]; state combination by aggregating similar states; numerical coarsening by means of scaling down numerical precision of the functionalities; order of magnitude reasoning [33]; reduction to a structure model by omitting behavior descriptions, which is called "representational abstraction" [12].

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 must 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: processing effort is shifted from the model utilization phase to the model construction phase. Model

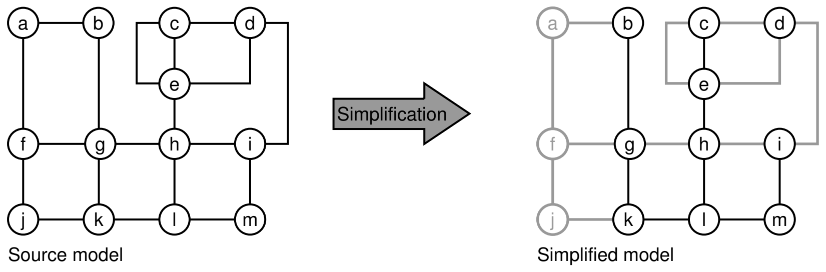


Fig. 5. Illustration of the source structure model (left) and a simplified structure model (right).

compilation is a powerful principle to address a model's analytic complexity, its constraint complexity, or the search space complexity. A compiled model is created 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. The idea 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 forms.

- Hints that use memorization to short-circuit involved state prescription constraints between functionalities of different submodels (see Figure 6). Cause effect chains are shortened, possibly to simple associations.
- Hints that suppose an order within a sequence of tentative search space decisions. 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.

Model compilation methods can also be characterized by their *scalability*. Using a scalable method, there is a trade-off between the effort for model *pre*processing and model processing at runtime. The scalable character of a compilation method is 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 usually not scalable. Formal description:

1. *Characteristics.* The set of functionalities remains unchanged. The behavior model is enriched with additional constraints that encode numerical or search-specific hints.
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 faster. However, 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 require a renewed preprocessing.

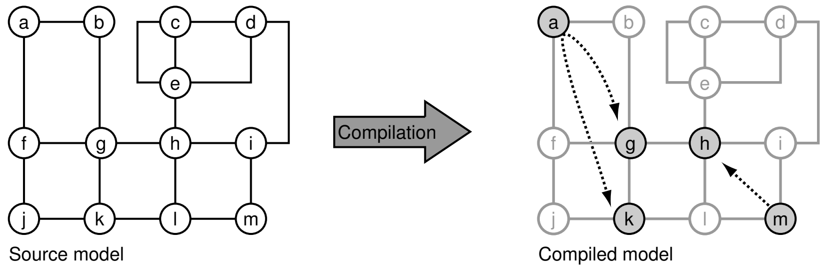


Fig. 6. Computational short cuts: hints in the compiled behavior model short-circuit the computation of constraints between functionalities of different submodels. For example, the input parameters could directly be mapped onto the output parameters.

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, further prerequisites must be fulfilled: (a) The problem solving task can be split 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. (b) The number of problem instances that require a recompilation is small.
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 [25]; 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; model decomposition by causal analysis [8].
   γ_B (computational short cuts): 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 [13]; utilization of an assumption-based truth maintenance system (ATMS) in order to organize results of computational expensive inference problems [19]; sharing of computation results by identifying and replacing instances of similar submodels, a concept that can be realized algebraically; case-based learning of characteristic features to select suited inference methods for simulation [43].
   γ_B (exploratory short cuts): behavior aggregation by a precedent detection of repeating cycles [48]; coalescing a system by methods from the field of inductive inference [4]; 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 [9].

Remarks. 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. While the rete-algorithm has been developed for rule languages whose interpretation is defined on the recognize-and-act cycle, the compilation method in [52] exploits the fixed-point convergence of the rule language to be compiled.

The following examples give an idea of the spectrum of knowledge forms where compilation methods can be applied: (1) The syntactic analysis and substitution of algebraic terms. (2) A still more basic knowledge compilation method is based on Horn approximations [40]; its prescribed knowledge representation form are formulas in propositional form. (3) If graphs are the interesting knowledge form, knowledge inference means graph matching or isomorphism analysis. For the latter problem a compilation method with scalable preprocessing effort is presented in [26].

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; rather, 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. Formal Description:

1. *Characteristics.* The set of functionalities may or may not be altered. Typically, the state prescription function is reformulated, leading to 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.

As opposed to 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.

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 efficiency, of course. We speak about model reformulation, if a model *must* be transformed into another representation in order to be processed within the problem solving task.

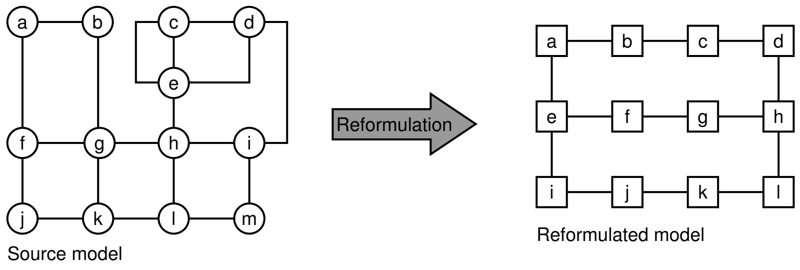


Fig. 7. Model reformulation is usually bound up with a paradigm shift in model processing, entailing both a new structure model and behavior model.

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

Table 2. Short characterization of three horizontal model construction principles.

Discussion and Related Work

Table 2 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. The table shows the dependencies in an oversimplified way and should only be used as a road map.

The three horizontal model construction principles are process-centered: 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.

Figure 8 shows the model transformation theory from Zeigler et al. [51]. The vertical arrows in the 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: 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. 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

$$\begin{tikzcd}
\text{Structured system} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \text{Structured system}' \arrow[d, leftrightarrow] \\
\text{I/O system} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \text{I/O system}' \arrow[d, leftrightarrow] \\
\text{I/O function} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \text{I/O function}' \arrow[d, leftrightarrow] \\
\text{I/O relation} \arrow[r, leftrightarrow] & \text{I/O relation}'
\end{tikzcd}$$

Fig. 8. Hierarchy of system specifications according to Zeigler et al. [51]. The horizontal lines represent mappings for model construction, called morphisms here.

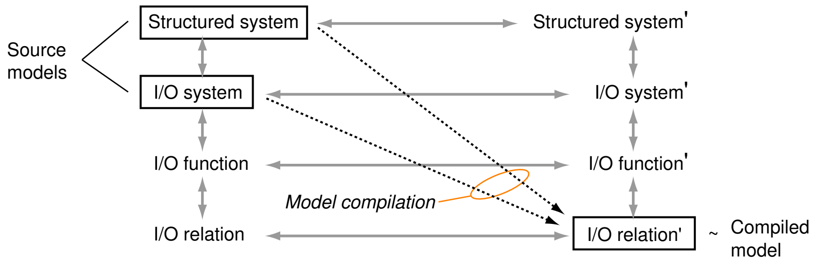


Fig. 9. Integrating the idea of model compilation into Zeigler et al.'s hierarchy of system specifications: a high-level system description, e. g. an equation model, is broken down to plain I/O relations by means of model compilation.

model on the left-hand side onto the I/O-relation model at the lowermost level on the right-hand side (see Figure 9).

4 Case Studies

This section presents three case studies where the horizontal model construction paradigm is applied. Note that the section cannot serve with detailed introductions and discussions of algorithmic, technical, or problem-specific background; for this purpose we refer to the respective literature. The case studies have illustrative character and shall sensitize the interested reader to identify the use of horizontal model construction principles in other situations. Nevertheless, they describe non-trivial, knowledge-intensive real-world applications that were tackled in the outlined way.

4.1 Case Study 1: Plant Design in Chemical Engineering[1]

A chemical plant can be viewed as a graph whose nodes represent devices or unit-operations, while the edges correspond to pipes responsible for the material flow. Typical unit-operations are mixing (homogenization, emulsification, suspension, aeration), heat transfer, and flow transport. The task of designing a chemical plant is defined by a demand set 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 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 subtasks; they relieve the human designer from special simulation or configuration tasks, and the effort involved there is high enough.

Instead of deriving a concrete solution for supplied demands at the modeling level, the physical model is simplified to an abstract model. On this abstract level,

[1] Supported by DFG grants PA 276/23-1 and KL 529/10-1

a solution can be efficiently calculated and transferred back to the physical level, although some adjustments may be necessary at this point. See Figure 10 for an illustration.

Model Simplification

The following model simplification steps include structure model simplifications (S1-S4) and behavior model simplifications (B1-B6), resulting in a tractable design problem. They are oriented at the taxonomy of model abstraction techniques listed in [14].

(S1) *Single Task Assumption.* It is general practice to combine different chemical processes in order to share by-products or partial process chains. A combined design corresponds to the solution of different tasks simultaneously—a procedure which belongs to optimization. Here, design is restricted to the n:1-case, and overlapping plant structures are split and dealt with separately as shown in Figure 11.

(S2) *Model Context.* The way how models are embedded into a context is clearly defined. Pumps, for example, have a strict structural relationship; they have single input and output, and the predecessor of a pump must be another device.

(S3) *Limited Input Space.* During the design process, 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.

(S4) *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. 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 yields a good approximation, even in the complicated cases.[2]

[2] The symbols φ_i and η_i designate the volume portion and the viscosity of input i.

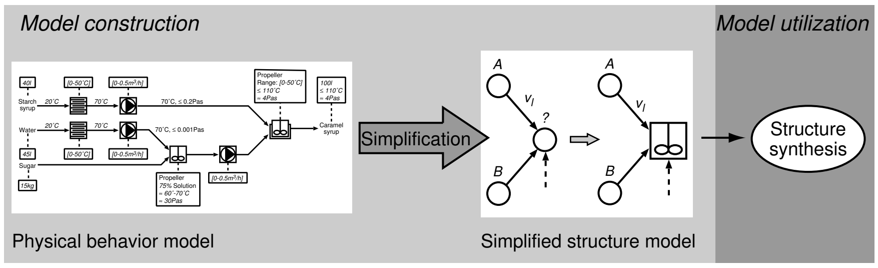


Fig. 10. The behavior model is abstracted to a structure model where physical quantities are represented as linguistic variables. The synthesis space is restricted to a set of labeled graphs.

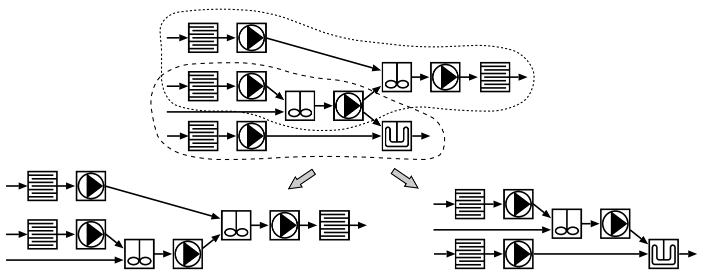


Fig. 11. Single task: overlapping plant structures are split and treated separately.

(B1) *Causal Decomposition.* To prevent components from exerting influence on themselves, feedback loops are not allowed.

(B2) *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.

(B3) *State Aggregation.* In general, the material being processed within a device is in different states. This behavior is simplified by assuming that, inside a device, a material will be in one single state.

(B4) *Temporal Aggregation.* Time is neglected, rendering statements about continuous changes to material properties impossible; changes to material properties are connected to entry and exit points within the plant structure.

(B5) *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.

(B6) *Function Aggregation.* In contrast to entity aggregation by function, where devices are represented by a special device, we aggregate functions. For instance, mixers are capable of performing different functions, such as homogenization, emulsification, aeration, or suspension.

The DIMOD *Workbench*

Rationale of the aforementioned model simplifications is to automate the generation of adequate structure models for a given set of demands. We developed tools to automate this synthesis step with graph grammars: they allow for knowledge modeling, manipulation, and systematic exploration of the search space, which are essential requirements for a successful synthesis of structure models [39]. Graph grammars generate graphs by applying transformation rules on, initially, some start graph. Here, the start graph represents the unknown chemical plant where the abstracted demands in the form of input and output substance properties are connected to. The successive application of transformation rules corresponds to the application of domain knowledge with the goal to transform the start graph into a design fulfilling all demands.

Figure 12 illustrates the use of graph grammars in chemical engineering design; it shows graphical representations of a small fraction of the transformation rules

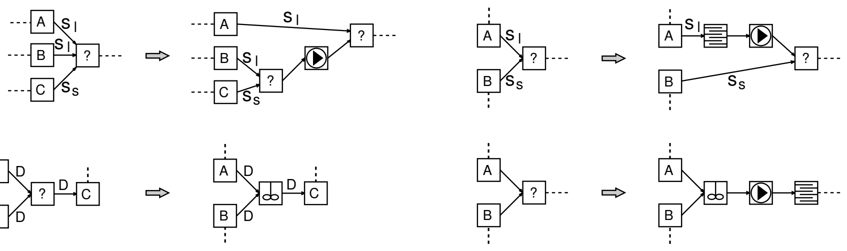
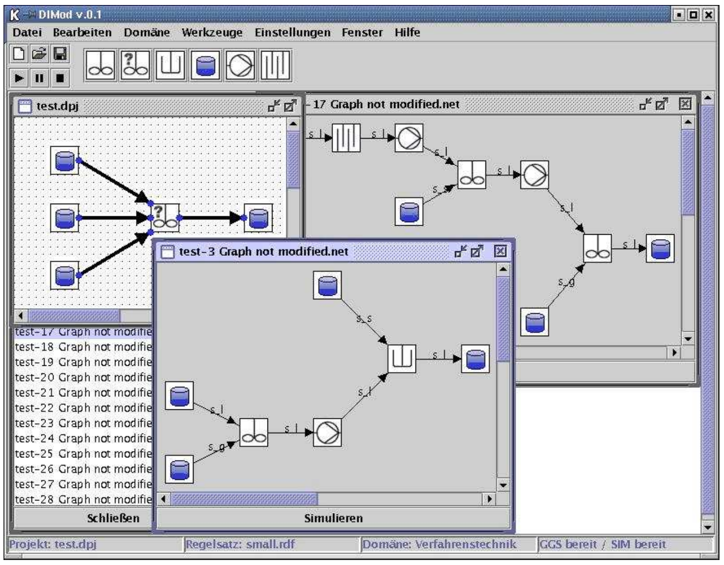


Fig. 12. Top: synthesis rules for the splitting of mixing jobs (left) and the insertion of a heating chain to improve dissolution (right). Bottom: refinement of a generic mixer as propeller mixer (left) and refinement of a generic mixer as propeller mixer with trailing heating chain (right).

in the design knowledge base of DIMOD. The rules, which have been developed in close cooperation with domain experts, are used for synthesis, refinement, and optimization purposes when designing plants for food processing.

Graph grammars can be seen as a collection of rules that define some search space. Note, however, that they provide no means to *navigate* within this search space; each domain requires a dedicated search method exploiting domain knowledge. Figure 13 shows a snapshot of the DIMOD workbench that has been developed to support human designers when solving synthesis tasks in chemical engineering. The core of the system consists of a generic graph grammar engine, used for modeling and application of knowledge, and a domain-specific module used to guide the search process. A generated structure model is completed towards a tentative behavior model by attaching behavior model descriptions to the components of the


DIMod v.0.1
Datei Bearbeiten Domäne Werkzeuge Einstellungen Fenster Hilfe
test.dpj
-17 Graph not modified.net
test-3 Graph not modified.net
s_l
s_s
s_g
Schließen
Simulieren

test-17 Graph not modifie
test-18 Graph not modifie
test-19 Graph not modifie
test-20 Graph not modifie
test-21 Graph not modifie
test-22 Graph not modifie
test-23 Graph not modifie
test-24 Graph not modifie
test-25 Graph not modifie
test-26 Graph not modifie
test-27 Graph not modifie
test-28 Graph not modifie

Projekt: test.dpj | Regelsatz: small.rdf | Domäne: Verfahrenstechnik | GGS bereit / SIM bereit

Fig. 13. The DIMOD workbench. The upper left window represents the abstracted demands, the windows to the right and center show two generated structure models.

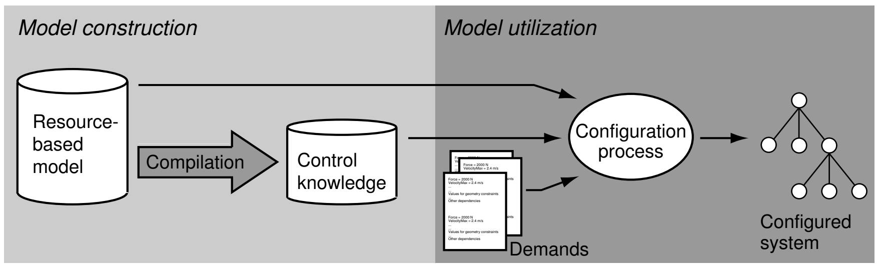


Fig. 14. Partitioning the resource-based configuration process.

structure model. Tentative behavior models are validated by simulation, for which the ASCEND IV simulator is employed [30].

4.2 Case Study 2: Generating Control Knowledge for 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 constraints. Aside from technical restrictions a customer's demands form a larger part of these constraints [6, 15]. A configuration process can rely on a structure model or on a behavior model. If emphasis is on the former the configuration approach is called skeleton configuration; if emphasis is on the latter the configuration task can be solved with a so-called resource-based configuration approach.

Within resource-based configuration the involved components are characterized by simplified functional dependencies, the resources. E. g. when configuring a small technical device such as a computer, one functionality of a power supply unit may be its power output, f_1, and one functionality of a plug-in card may 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 balanced with the supplied positive numbers. Within a technically sound model the balance of each resource must be zero or positive.

Resource-based modeling provides powerful and user-friendly mechanisms to formulate configuration tasks. However, the solution of resource-based configuration problems is NP-complete, which means that no efficient algorithms exist to solve a generic instance of the problem [41]. When given a real-world configuration task formulated within a resource-based description, the search for an optimum configuration is often realized efficiently with heuristics that are provided by human experts. Put another way, a resource-based system description can be enriched with control knowledge, and model compilation is the idea to generate control knowledge automatically. Consequently, the configuration process is divided into a preprocessing phase, where heuristics are generated, and in a configuration phase, typically at the customer's site, where a concrete configuration problem is solved. Figure 14 illustrates this view.

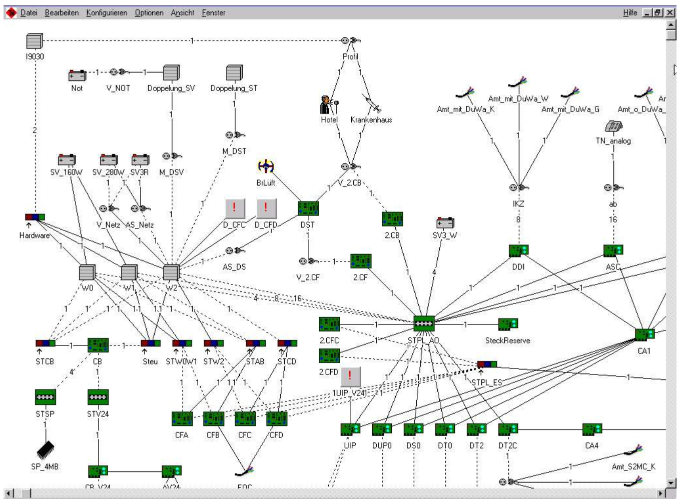


Fig. 15. Resource model representation of a telecommunication system in PREAKON.

The Bosch Telecommunication Application

The configuration of telecommunication systems requires 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: constraints can be considered as a finite set of functionality-value-pairs, which are supplied or demanded from the components.

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 the development of the resource-based configuration system PREAKON. Figure 15 shows a part of the knowledge base in PREAKON's acquisition mode. The field tests showed the necessity of a heuristic search space exploration if optimum configurations should be found in an acceptable time. For the following reasons we refrained from a *manual* integration of control knowledge:

1. Control knowledge of domain experts is often contradictory or incomplete.
2. The additional effort bound up with maintenance of heuristic knowledge complicates the configuration system's introduction.
3. Each modification of the knowledge base (e. g. new components) can potentially invalidate existing heuristics.

Instead, the model compilation paradigm was pursued—the automatic generation of control knowledge by means of preprocessing.

Model Compilation

Basically, resource-based configuration works as follows. First, the demand set of a virtual balance is initialized with all demanded functionalities. Second, with respect to some unsatisfied resources a component is selected and its functionalities are added to the corresponding resources of the balance. Third, it is checked whether all resources are satisfied. If so, the selected components form a solution of the configuration problem; otherwise, the configuration process is continued with the second step. Without powerful heuristics that control the functionality and component selection steps, only toy 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.

Heuristics for resource selection are derived from graph-theoretical considerations which base on the analysis of the strong components in the dependency graph. Heuristics for component selection are derived from a best first search analysis,where for particular demand values the recursive follow-up cost of component alternatives are estimated. These sampling points are used to interpolate for each component and each demanded resource a function. The result of the analysis is a family of functions, which can be evaluated at configuration runtime.

The control knowledge must be recomputed each time the knowledge base is modified; this preprocessing phase takes several minutes on a standard PC. Altogether, the model compilation led to a significant speed-up for realistic instances of the configuration problem: PREAKON was the first configuration system at Bosch Telenorma that provided realistic means for being used at the customer site.

4.3 Case Study 3: Synthesis of Wave Digital Structures[3]

Wave digital structures have their origins in the field of filter design, where they are designated more specifically as wave digital filters [11]. 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 considerable numerical advantages. However, since neither the modeling accuracy nor its granularity is affected, the translation into a wave digital structure establishes a model reformulation.

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 16). This reformulation aims at the analysis, say, simulation of S, as illustrated at Gero's design cycle in Figure 17.

The construction of a wave digital structure is a design task, namely the design of a sophisticated algorithmic model. Since this design task is not trivial and needs

[3] Supported by DFG grant KL 529/10-1

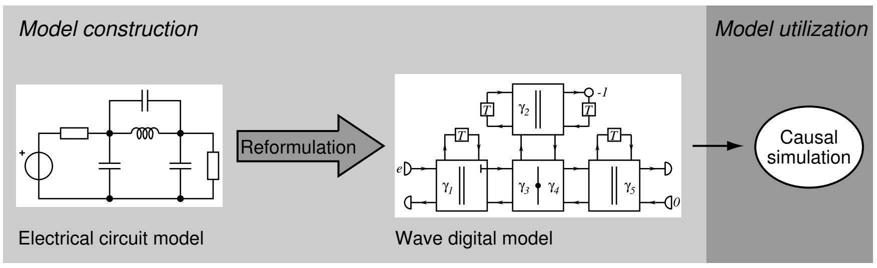


Fig. 16. Reformulation of an electrical circuit model as wave digital structure for model processing reasons.

experience, its automation is a worthwhile undertaking. In [42], the necessary concepts and algorithms for a WDS design automation are explained in detail. At the same place an algorithm is presented which computes for a given electrical circuit the optimum wave digital structure in linear time—a result which cannot be further improved. In the following we outline the reformulation procedure.

Model Reformulation

As already noted, a wave digital structure is a special kind of signal flow graph. Its topology is realized with series and parallel connectors; the signals that are processed when traveling along the signal flow graph are wave quantities. The reformulation of an electrical circuit as a wave digital structure involves three 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.

These reformulation steps divide into local operations (Step 2 and 3), which pertain to 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, say, their order of application can be interchanged.

Topology Reformulation. Let S be an electrical circuit. The reformulation of the Kirchhoff interconnecting network of S starts with the identification of subsystems in S that are connected in series or in parallel. Both series connections and parallel

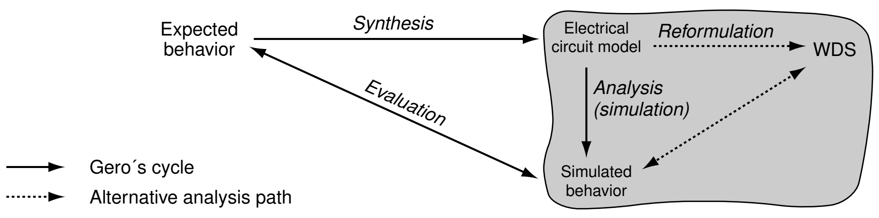


Fig. 17. Left: Gero's widely-accepted model of the human design process [15]. The automatic synthesis of WDS provides a numerically attractive alternative to operationalize the analysis step (right, shown gray).

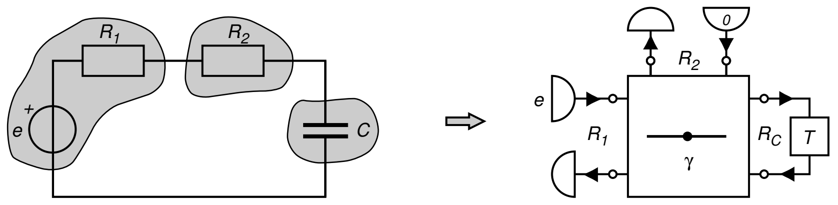


Fig. 18. Electrical circuit with two resistances and a capacity (left) and the related wave digital structure (right). The shaded areas in the circuit indicate the decomposition into three ports.

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 respective connectors, which guarantee that Kirchhoff's laws are fulfilled and which permit a special network analysis approach. Common network analysis approaches are based on mesh equations, node equations, or state equations [7]. Following a common approach means to set up and transform matrices, such as 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.

For the part of S that is realized with series and parallel connections model processing effort can be significantly decreased: 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 by local propagation, forms a causal behavior model. Such a behavior model is the most efficient algorithmic model possible.

Transfer to the a/b-Wave Domain. The electrical quantities voltage, v, and current, i, can be expressed by so-called wave quantities, a, b, which are linear combinations of v and i. Common is a transformation into voltage waves,

$$a = v + Ri, \qquad b = v - Ri,$$

where a and b represent the incident and reflected wave respectively; R is called the port resistance. When applying these relations in the above topology reformulation step, 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 have 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. Figure 18 shows an electrical circuit and its related wave digital structure containing one series adaptor.

Note that, aside from performance reasons, the reformulated counterpart of circuit S in the a/b-wave domain comes along with superior numerical properties, which simplify the operationalization of S in the form of an integrated circuit.

5 Summary

Model construction is an artistic discipline whose overall objective is to find the right model in order to give an answer to an open question. Our research discussed the classical top-down model construction approach and introduced the idea of horizontal model construction. Horizontal model construction starts with an already developed model, which then is improved in different respects, be it efficiency, handling, complexity, or others. Hence, horizontal model construction does not follow a fixed scheme but takes very different forms from which we introduced three important principles: model simplification, model compilation, and model reformulation. For each of these principles we introduced its basic characteristics and exemplified its application within a knowledge-intensive engineering task.

Among experienced engineers and problem solvers horizontal model construction is common practice and may often be applied subconsciously. In this connection our research shall contribute to both a better understanding and a formal theory of model construction within complex and knowledge-intensive tasks.

References

1. 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.
2. 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.
3. 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.
4. D. Angluin and C. H. Smith. Inductive Inference: Theory and Methods. *Computational Surveys*, 15(3):237–269, September 1983.
5. 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.
6. David C. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.
7. François E. Cellier. *Continuous System Simulation*. Springer, 1991.
8. 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.

9. 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.
10. Hilding Elmquist. Object-Oriented Modeling and Automated Formula Manipulation in Dymola. In *SIMS'93*, Kongsberg, Norway, June 1993. Scandinavian Simulation Society.
11. Alfred Fettweis. Wave Digital Filters: Theory and Practice. *Proceedings of the IEEE*, 74 (2):270–327, February 1986.
12. Paul A. Fishwick. The Role of Process Abstraction in Simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 18:18–39, February 1988.
13. Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.
14. 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.
15. John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.
16. Stefan Goldschmidt. Modellbildung am Beispiel von Starrkörpersystemen. Systematische Darstellung und Untersuchung von Software-Unterstützung. Study work, University of Paderborn, November 1996.
17. R.C. Hibbeler. *Engineering Mechanics: Statics and Dynamics*. MacMillan, New York, 6 edition, 1992.
18. Yumi Iwasaki and Alon Y. Levy. Automated Model Selection for Simulation. Knowledge Systems Laboratory (KSL), QS93, 1993.
19. Johan de Kleer. Problem Solving with the ATMS. *Artificial Intelligence*, 28:197–224, 1986.
20. G. J. Klir. *Architecture of Systems Complexity*. Sauders, New York, 1985.
21. Granino A. Korn and John V. Wait. *Digital Continuous-System Simulation*. Prentice-Hall, Englewood Cliffs, N.J., 1978.
22. 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.
23. R. S. H. Mah. *Chemical Process Structures and Information Flows*. Butterworths, 1990.
24. Mary Lou Maher and Pearl Pu, editors. *Issues and Applications of Case-Based Reasoning in Design*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1997.
25. 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.
26. 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.
27. Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.
28. P. Panduring Nayak. Causal Approximations. *Artificial Intelligence*, 70:277–334, 1994.
29. P. Panduring Nayak. *Automated Modelling of Physical Systems*. Springer, 1995.
30. 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.

31. Frank Puppe. *Systematic Introduction to Expert Systems, Knowledge Representations and Problem-Solving Methods*. Springer, 1993.
32. 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.
33. Olivier Raiman. Order of Magnitude Reasoning. *Artificial Intelligence*, 51:11–38, 1991.
34. 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, 1996.
35. 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.
36. 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.
37. 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.
38. D. B. Schaechter, D. A. Levinson, and T. R. Kane. AUTOLEV *User's Manual*, 1991.
39. L. C. Schmidt and J. Cagan. Configuration Design: An Integrated Approach Using Grammars. *ASME Journal of Mechanical Design*, 120(1):2–9, 1998.
40. 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.
41. Benno Stein. *Functional Models in Configuration Systems*. Dissertation, University of Paderborn, Institute of Computer Science, 1995.
42. Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation, Department of Computer Science, University of Paderborn, Germany, June 2001. URL `http://ubdata.uni-paderborn.de/ ediss/17/2001/stein/`.
43. 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, September 1997.
44. 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.
45. Peter Struss. Multiple Models for Diagnosis. SPQR-Workshop on Multiple Models, FRG Karlsruhe, March 1991.
46. Michael Suermann. Wissensbasierte Modellbildung und Simulation von hydraulischen Schaltkreisen. Diploma thesis, University of Paderborn, 1994.
47. J. Wallaschek. Modellierung und Simulation als Beitrag zur Verkürzung der Entwicklungszeiten mechatronischer Produkte. *VDI Berichte, Nr. 1215*, pages 35–50, 1995.
48. Daniel S. Weld. The Use of Aggregation in Causal Simulation. *Artificial Intelligence*, 30:1–34, 1986.

49. A. Wayne Wymore. *Systems Engineering for Interdisciplinary Teams*. John Wiley & Sons, New York, 1976.
50. Kenneth Man-kam Yip. Model Simplification by Asymptotic Order of Magnitude Reasoning. *Artificial Intelligence*, 80:309–348, 1996.
51. Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation*. Academic Press, New York, 2000.
52. Blaž Zupan. Optimization of Rule-Based Systems Using State Space Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):238–253, March 1998.