Anniversary Workshop on “Planning and Configuration” (PUK 96).
Bonn, April 1996. Edited by J. Sauer, A. Günter, and J. Hertzberg.
pp. 56-70, ISBN 3-92037-97-1, Infix, 1996.

Model Formulation and Configuration of Technical Systems

Benno Stein✉ Daniel Curatolo

Abstract

Each configuration process relies on a particular component model. A component model is a useful abstraction of the domain and the technical system to be composed, and it must be tailored to the configuration problem. The formulation of adequate component models is a highly creative job and cannot be automated in its universality.

However, model formulation can be automated to some extent. This paper elaborates on three concepts of model formulation support within configuration tasks: component model refinement, component model compilation, and component model synthesis. To each concept a configuration problem along with the realized model formulation support is sketched out.

Introduction

A lot of research is related to the configuration of technical systems. This effort is justified. The process of selecting and composing building blocks in a way that the resulting system meets some given requirements can turn out to be a demanding job [3], [5], [16], [18].

Actually, the main thread of this paper is not composition; instead it concentrates on *component models*, which are subject to all kinds of composition processes. A component model comprises the descriptions of the components, the concepts of the domain, and the environmental restrictions. The following example will illustrate this term in greater detail.

> In a loudspeaker production, among other things, frequency switches have to be configured. A frequency switch in a loudspeaker system splits the frequency band into two, three, and four ranges respectively; basically, it consists of a circuit board and different capacitors and coils. These electrical building blocks must be selected and connected regarding various aspects, such as the number and the transition frequencies of all loudspeaker chassis involved, the intended electrical power, and the loudspeaker system’s price category.
>
> Clearly, from a configurational standpoint, the composition of the electrical building blocks follows another rule (relies on another component model) than from an electrical or a physical point of view.
>
> In the former case the composition knowledge needs to be oriented at the company’s product spectrum only; it is likely that a small set of rules will be sufficient to model all configuration restrictions. Put another way, the component model is both specialized and simple. It operationalizes a *mapping* between the different loudspeaker variants and the different types of frequency switches. For generalization and efficiency issues such a mapping is usually constructed by means of a few characteristic attributes (cf. figure 1).

✉University of Paderborn
Knowledge-Based Systems Group
stein@upb.de

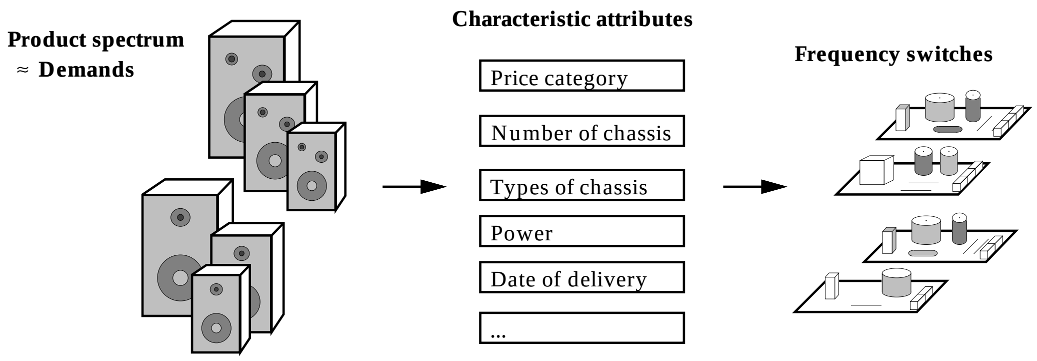

Figure 1: Configuration of frequency switches using a simple component model.

On the other hand, when taking an electrotechnical standpoint, a component model will reflect the *physical behavior* of frequency switches. Such a component model operationalizes knowledge about alternating current resistances, complex resistances, rates of damping, capacities, etc. A composition process based on that kind of "first principles" rather establishes a demanding design problem than a simple configuration problem.

Note that the simplified component model has *design knowledge compiled in*. It enables non-experts to configure correctly working frequency switches by merely applying some rules. The simplified component model does obviously reflect all physical laws relevant for the configuration process.

By *model formulation* we denote the creation of a component model for a given physical system. Of course, model formulation is always a teleological procedure. The example above reveals that the adequacy of a component model with respect to its correctness, its level of abstraction, and its maintenance plays a key role when solving a configuration task:

❑ *Correctness.* The component model reflects a particular part of the concepts of the real system and the domain. The implications of the component model may be incomplete, but they must not contradict the system's real behavior.

❑ *Level of Abstraction.* To make knowledge processing efficient the component model should be as simple as possible. It is an artistic discipline to determine an abstraction level that models the entire configuration-relevant knowledge and that cannot be simplified further. In fact, the creation of new modeling concepts often makes up a large part of a configuration problem.

❑ *Maintenance.* The maintenance, i.e. the modification, the update, and the checking of configuration knowledge is an important part of each configuration problem. The component model should simplify knowledge maintenance as far as possible. E.g., the roles that particular portions of knowledge play should be made explicit, configuration knowledge should rather be local than global, metaknowledge should be declared and separated.

The formulation of an adequate component model related to a given configuration problem is a highly creative job. I.e., this job cannot be automated in its universality. However, model formulation can be automated to some extent. In this place we elaborate on three concepts of model formulation support within configuration tasks:

1. *Component Model Refinement.* Given is an incomplete specification of a component model. By component model refinement we designate a completion process to such an extent that the configuration problem can be mastered with the refined component model.

2. *Component Model Compilation.* Given is a complete specification of a component model. By component model compilation we designate a process that adds (control-) knowledge to the component model to make the configuration process more efficient.

3. *Component Model Synthesis.* Given is a set of model fragments and the topology of a complex system. By component model synthesis we designate the process of selecting and composing model fragments to a new, global component model that correctly models the system.

Remarks. Model formulation has been—and still is—an object of research. One example is the development of the system SALT, a knowledge-acquisition tool for generating expert systems that use a propose-and-revise strategy [15], [14]. Model formulation due to SALT is based on the following elements: the characteristic domain parameters, the computation rules including constraints for these parameters, and the revision instructions to resolve inconsistencies. Given a clearly defined problem, SALT controls the composition of knowledge pieces to a complete component model by means of a guided interview.
Another example is the "How Things Work Project" (Stanford University, [6], [7], [17]), in particular the work of Nayak, which does also touch different aspects of an automated model formulation. The related research is concerned with the construction of adequate models regarding a given simulation or explanation task.

Model formulation may not be recognized as such; it has very different appearances, as the subsequent sections show. The paper in hand identifies automated model formulation as an important concept when supporting particular configuration tasks.

Note that this paper shall neither serve as a collection of configuration techniques nor propose suited models for dedicated configuration problems. Its overall objective can be summed up by the following question: Given a problem along with an adequate modeling concept—which part of the model formulation process can be accomplished automatically?

To unfold the connecting dependencies, the three examples are arranged in a similar manner: First, a short introduction to the domain and the problem is given. Second, within the "Modeling Concept" subsection, the modeling approach developed from domain experts and knowledge engineers is outlined. The third subsection then illustrates the idea of the related model formulation support, while the last subsection, "Realization", shows how this idea can be operationalized.

Example 1 Impeller Design of Mixing Machines

Mixing machines are employed in a wide area of industrial applications [2]. Their configuration primarily depends on the mixing operations, the products involved, and the quantity to be mixed. There are only a few conceptual elements that make up a mixing machine, but the number of possible variants is extremely high. In this place we will solely concentrate on the design of the impeller.

Impeller design constitutes a classification problem if it is realized as a direct mapping between mixing tasks and possible impeller types. Nevertheless, when taking a closer physical view, impeller design happens within a synthesis and an analysis step. In the first step several impeller characteristics are derived and combined with respect to the mixing task; in the second step these characteristics are interpreted, and an appropriate impeller is selected or constructed. Figure 2 illustrates these steps.

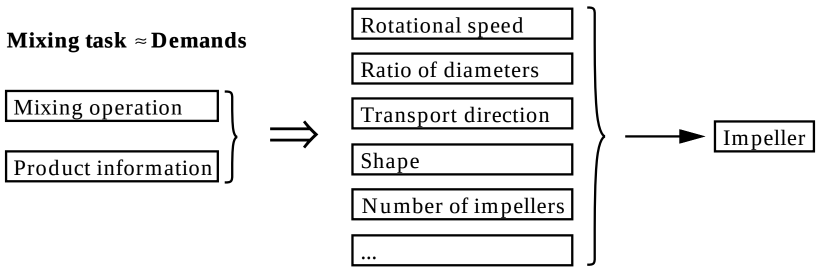

Figure 2: Characteristic parameters in impeller design.

Modeling Concept

Quantitative models that describe the underlying design process rely on fluid mechanics and thermodynamics and lead to complex partial differential equations. At the moment such models can hardly be formulated and processed respectively. Thus, impeller design is a matter of experience and engineering know-how.

In order to obtain a reliable model of this expertise, real design examples should be exploited during some kind knowledge of acquisition process. Each design example defined a set of mixing operations, the viscosities, gravities, and quantities of the products involved, and a description of the installed impeller. Our main objective was not a mere classification of the given examples but the creation of a component model that makes design decisions comprehensible.

The discussions with domain experts pointed to the rule paradigm as a suited modeling concept. Based on our experience in knowledge engineering, we decided to use Fuzzy rules for knowledge representation and processing—based on their experience in impeller design, the domain experts abstracted characteristic attributes together with significant value ranges from the examples.

Note that the Fuzzy rule paradigm along with the characteristic attributes, the associated value ranges, and a reasonable range segmentation (the membership functions), determine the abstraction level of the component model. Hence, the most creative part within the model formulation process is still managed. Given both this knowledge about the abstraction level and the design examples, the creation of a knowledge base for impeller design can be automated through *component model refinement*.

Component Model Refinement

In our case the process of component model refinement was operationalized within the following three steps:

1. Creation of a set of generic rules.
2. Refinement of the generic rules towards meaningful design rules.
3. Pruning of the set of design rules.

Remarks. (*i*) The term “generic rule” denotes a rule which *may* embody a piece of useful design knowledge. Generic rules are generated automatically from the design examples according to an interpretation scheme developed by the domain experts. Example of a generic rule:

```
IF  operation is suspending  AND  viscosity is very_low
THEN  ratio_of_diameters is small
```

(*ii*) Generic rules become design rules by evaluating the set of the given design examples. In particular, weight values are computed and assigned to a rule’s premises, its conclusions, and the rule as a whole. (*iii*) Since in step 1 all possible rules that match a certain pattern are syntactically generated, the set of design rules can be reduced. Thus, within step 3, relevant design rules are separated from irrelevant and contradictory rules, using the information gained within step 2.

Note that the utilization of the design examples is twofold. On the one hand, the examples control the generation of the generic rules; on the other hand, they are exploited for the rules’ evaluation.

Also note that the reduced rule set serves two purposes. In the first place it realizes a decision system for impeller design. Secondarily it represents a compact domain model that explicitly formulates design knowledge.

Realization[2]

Rule Creation. Two types of generic rules had to be created: rules that express the synthesis dependencies between mixing tasks and impeller characteristics, and rules that map from impeller characteristics on impeller types. Both types of rules have multiple premises and a single conclusion. Due to the fuzzification procedure, which relies on domain-specific definitions of membership functions, the exact parameter values of a design example may lead to several instantiations of their linguistic counterpart.

In the following a small example is sketched out. Let us assume that a blending operation, among others, is characterized by a viscosity value of 50mPas and a diameter ratio of 0.25. Then the following linguistic rules are generated:

```
IF op is blending AND viscosity is very_low THEN ratio_of_diam is very_small
IF op is blending AND viscosity is very_low THEN ratio_of_diam is small
IF op is blending AND viscosity is low THEN ratio_of_diam is very_small
IF op is blending AND viscosity is low THEN ratio_of_diam is small
```

More than 300 rules of the first type were generated from the design examples. Note that the majority of these rules apply to *several* design examples, and therefore reasonable membership values cannot be supplied easily. This problem is addressed within the next step.

[2]We will not engage in the details. An exact description of the outlined procedures and related problems may be found in [1], [4].

Rule Refinement. The refinement of the rules is the most crucial step within the automated part of this model formulation procedure. We realized and compared the following two approaches:

(a) Computation of the Fuzzy membership values such that both multiple rule matching and the distribution of the design examples are taken into account.

(b) Training of a neural network with the design examples.

Both approaches lead to an assessment of a rule's plausibility in the form of weight values for the rule's premises, its conclusion, and the rule as a whole. In this place the second approach is outlined.

As the following considerations show, the generic rules can be refined by means of a neural network. (*i*) An edge in a neural network establishes a directed connection, just as our rules do. (*ii*) The edges in the network are weighted; these values can be interpreted as the searched rule weights. (*iii*) The neural network can be trained with the existing set of design examples. Figure 3 illustrates the network.

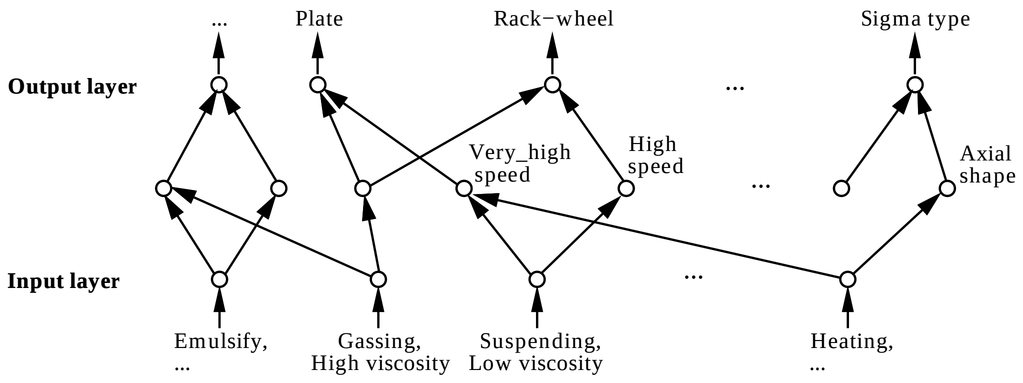

Figure 3: Mapping generic design rules onto a network.

The input neurons of the network comprise the premises of the rules that embody synthesis dependencies; as a result, the single hidden layer encodes the impeller characteristics; the output layer's neurons correspond to the impeller types.

This neural network along with the set of design examples formed the input for a back propagation algorithm; the activation potentials of the input neurons were derived via fuzzification of the exact example values. The training resulted in large positive edge weights for rules that were supported from various examples; it led to negative weights for rules responsible for classification errors.

Application. Either refinement approach delivered plausibility values that were scaled and added to the rules. A classification test of the design examples showed that refinement approach (a) was superior to approach (b) regarding the design support: For about 80% of the existing examples the same solution was proposed; the remaining 20% contained many acceptable—but also wrong solutions. On the other hand, refinement approach (b) showed a more robust behavior regarding different pruning bounds. According to a domain expert the resulting rule sets both provide a useful design support.

Example 2 Configuration of Telecommunication Systems

The configuration of telecommunication systems is grounded on technical know-how since the right boxes, plug-in cards, cable adapters, etc. have to be selected and put together according to spatial and other constraints. Typically, there exists a lot of alternative systems that fulfill a customer's demands from which—with respect to some objective—the optimum has to be selected.

The components of a telecommunication system can be divided into mandatory technical components for the switchboards, demand-dependent plug-in cards and related material, and components that embody services and extensions that a customer can choose. Figure 4 shows the dependencies.

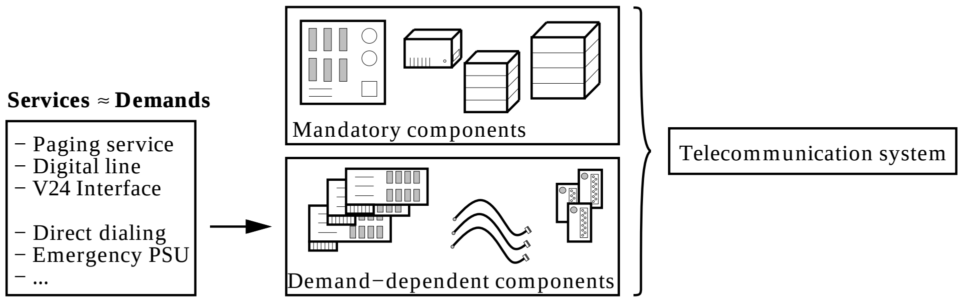

Figure 4: Technical dependencies within telecommunication systems.

Modeling Concept

For this kind of domain and configuration problem the resource-based modeling concept has been successfully employed [8], [12], [19]. A resource-based component model establishes an abstraction level that reduces a domain to a finite set of functionality-value-pairs, which are supplied or demanded from the components. The functionalities may be of symbolic or of numerical type, i.e., their range can be a number field or a list of symbols. Figure 5 gives a modeling example.

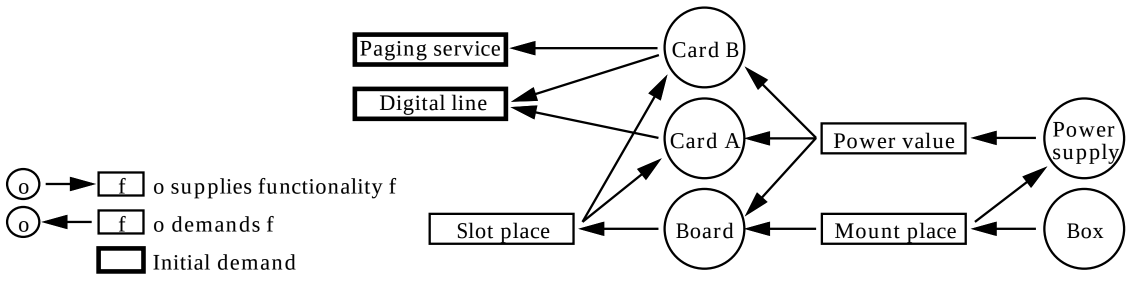

Figure 5: Resource-based modeling of simplified telecommunication dependencies.

A basic configuration method that copes with a resource-based component model is balance processing. Balance processing operationalizes a generate-and-test strategy. The generate part, controlled by propose-and-revise heuristics, is responsible for selecting both an unsatisfied functionality f and a set of components that supply f. The test part is realized as follows: The supplies and demands of the selected components are put on the corresponding sides of a balance, and identical functionalities are accumulated due to some rule, e.g. the algebraic $+$-operation. Then each row of the balance is checked whether the demanded value can be satisfied by the supplied one or not.

Component Model Compilation

Resource-based component models provide great knowledge acquisition support since the configuration knowledge consists of local connections for the very most part. However, the basic balance processing algorithm has an exponential time complexity. I.e., if no powerful heuristics are at hand which control the functionality and component selection, merely small configuration problems can be tackled by this method.
Functionality selection is related to the search space's total depth in the first place; component selection affects the effort necessary for backtracking. An "intelligent" decision strategy within these selection steps is the major engine of efficient balance processing.

Our experiences have shown that domain experts can hardly formulate powerful selection heuristics for large knowledge bases. A possible way out is the off-line-generation of heuristics, by means of *component model compilation*.

Key idea of the component model compilation presented here is the computation of an estimation function (off-line) that controls the component selection step during the configuration process (online). Put another way, the major part of the search effort is anticipated within a preprocessing stage.

Realization[3]

One example for a component selection strategy is the following: "*To satisfy an open demand at functionality f, select from all components which supply f the cost minimum one.*"

In fact, such a local strategy is often too shortsighted. Thus 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: (*i*) A configuration situation shall solely be characterized by those functionalities that are actually unsatisfied, (*ii*) a functionality shall only be satisfied by components of the same type, and (*iii*) components shall be regarded as suppliers of a single resource.

Remarks. Point (*i*) neglects that unused resources in a partial configuration may be exploited in a further course of the configuration process. Point (*ii*) neglects that a combination of different components may constitute a more adequate solution for an unfulfilled functionality than a set of components of the same type. Point (*iii*) neglects that a component may supply several resources each of which is demanded in the partial configuration.

Based on the above simplifications we now construct the estimation function $h(o, f, n)$ for the computation of follow-up costs. f denotes the demanded functionality, n denotes the amount to which f shall be demanded, and o denotes a component that supplies f and that shall be used to satisfy the open demand. We will construct h within three steps:

1. Each component 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.
2. A component's follow-up costs result from its demands. More precisely: Let o be a component and $d(o)$ the demanded functionalities of o. Then, of course, we would like each

[3]This approach has successfully been operationalized within the configuration system PreAKON [11]. A more detailed description of the preprocessing concepts can be found in [10] and [20].

demand $v_d(o, g)$ of component o at functionality $g \in d(o)$ to be satisfied at minimum costs. Note that all components that will be selected to satisfy g entail follow-up costs on their turn. I.e., if we selected a component o in order to satisfy a required demand f, we would expect the following total cost c_t:

$$c_t(o, f) := c(o) + \sum_{g \in d(o)} \min_{\omega \in o(g)} \{c_t(\omega, g)\},$$

where		
	$c(o) \in \mathbf{R}^+$	local cost of component o
	$d(o)$	demanded functionalities of o
	$o(f)$	components that supply f

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. A component o may require the functionality f to an arbitrary amount $v_s(o, f) \in \mathbf{R}^+$. Hence we need a term that computes for a given amount n at functionality f the number of components o that are necessary to satisfy f:

$$\left\lceil \frac{n}{v_s(o, f)} \right\rceil$$

Now only the composition of the above terms remains to be done. As a result, we obtain the estimation function h that computes for a component o and a demand f at the amount of n the total costs:

$$h(o, f, n) := \left\lceil \frac{n}{v_s(o, f)} \right\rceil \cdot \left(c(o) + \sum_{g \in d(o)} \min_{\omega \in o(g)} \{h(\omega, g, v_d(o, g))\} \right),$$

where		
	$c(o) \in \mathbf{R}^+$	local cost of component o
	$d(o)$	demanded functionalities of o
	$o(f)$	components that supply f
	$v_s(o, f) \in \mathbf{R}^+$	supply at functionality f of component o
	$v_d(o, f) \in \mathbf{R}^+$	demand at functionality f of component o

Configuration Example

The use of the estimation function h is discussed now. Let us consider we had to solve a simple configuration problem where an initial demand of $6 \times f_1$ is given. Two components, o_1 and o_2, can be used to realize a system that fulfills this demand; they are defined in the following table:

	Supplies	Demands	Local cost
o_1	$2 \times f_1,\ 1 \times f_2,$	-	100
o_2	$4 \times f_1$	$1 \times f_2$	10

According to the formula previously derived, h is defined as follows:

$$\begin{array}{rcll} h(o_1, f_1, n) & = & \lceil \frac{n}{2} \rceil \cdot 100 & \text{(no follow-up costs)} \\ h(o_1, f_2, n) & = & n \cdot 100 & \text{(no follow-up costs)} \\ h(o_2, f_1, n) & = & \lceil \frac{n}{4} \rceil \cdot (10 + 100) & \text{(follow-up costs for } f_2\text{)} \\ h(o_2, f_2, n) & = & \infty & (f_2 \text{ can never be satisfied by } o_2\text{)} \end{array}$$

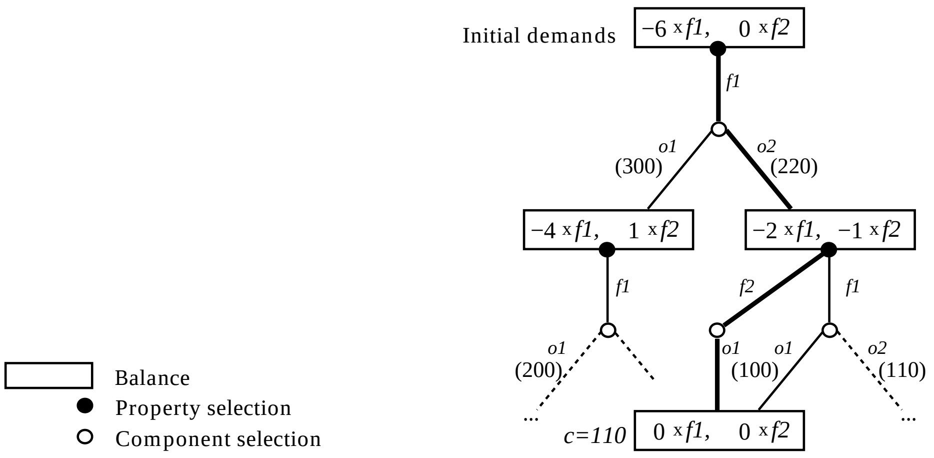

Figure 6: Part of the search tree of the configuration example.

A part of the resulting search tree is depicted in figure 6.

Remarks. The search tree is two-layered and consists of two types of nodes. (*i*) Filled nodes establish choice points regarding the functionality to be satisfied next. The related balance is shown framed above the node. (*ii*) 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 components shall be tried. In the example component o_2 is chosen at the first choice point, while $o1$ is chosen at the next. The earlier a solution is found, the earlier its cost information c can be used to cut partial configurations exceeding c. Note that h cannot be a function that estimates a configurations total cost.

The computation of h with respect to our example was very easy. As argued earlier, h can neither be computed precisely nor represented totally for large knowledge bases, since an exhausting search for all values in o, f, and n had to be performed. Hence we need an approximation of h.

Within our telecommunication application, this approximation was achieved as follows. (*i*) Depending on both the functionalities, $f \in F$, and the components, $o \in O$, the recursion depth of h was bound. (*ii*) Based on an evaluation in a few points, h was interpolated. These simplifications resulted in a family of $O(|O| \cdot |F|)$ functions, $h_{o_f}(n)$, which were evaluated at configuration runtime.

Example 3 Checking of Hydraulic Systems

A hydraulic system consists of mechanical, hydraulic, and electronic components. Configuring a hydraulic system is a complex process that starts with design concepts and ends with setting the installed system into operation. At present this process cannot be automated, and, given a readily configured hydraulic plant, even the checking of this plant with respect to its behavior and other demands turns out to be a difficult job [13]. In this place we will show how the checking of hydraulic systems can be supported.

Usually, the demands on a hydraulic system are specified by means of different diagrams. These diagrams indicate the course of forces at the cylinders, the piston accelerations, the switching positions at the valves, etc. During different checking stages a hydraulic engineer has to investigate if all geometrical connections fit, if the switching logic realizes the desired behavior, which maximum pressure values occur, and other connections. Figure 7 illustrates the checking task; it shows a hydraulic circuit and the desired behavior in the form of switching diagrams.

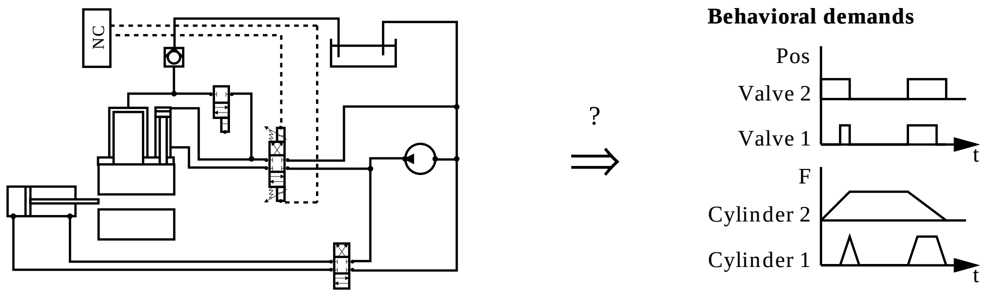

Figure 7: A hydraulic circuit with switching diagrams.

Modeling Concept

Hydraulic components are the building blocks of the systems to be checked. The abstraction level at which these building blocks need to be modeled is determined by the checking task: Since the checking of hydraulic systems requires the investigation of their exact behavior, modeling must take place at the level of physical behavior equations.

In cooperation with domain experts we developed a building block model and a behavior description language which simplify the description of hydraulic components. A component can be seen as a box that has gates of different types to communicate with its environment. Within each box physical parameters are associated with the gates. Using the description language, numerical and symbolic constraints can be formulated between all parameters of a box. Subsequently a part of a relief valve's behavior description is given[4]:

```
Q[gate1] = Q[gate2]   AND
IF excess_pressure    THEN  state[self] is open AND
                            p[gate1] - p[gate2] = RH[self] * Q[gate1]^2
                      ELSE  state[self] is closed AND
                            Q[gate1] = 0
```

Notice that behavior is modeled locally to a component. Also notice that hydraulic components have states each of which corresponds to a particular behavior alternative of the component.

[4]p, Q, and RH denote the pressure, the flow, and the hydraulic resistance respectively.

Component Model Synthesis

Loosely speaking, checking a hydraulic circuit means simulating the circuit. This simulation is based on a global model of the circuit, which in turn must be created from the local component descriptions. In this context component model synthesis is comprised of all steps that are necessary to set up a global model that is both correct in a physical sense and *locally unique*.

Even though a circuit diagram may establish a correct physical model, it is not locally unique as a rule: The system's components are defined by a set of behavior constraints from which the actually relevant ones must be selected. The indeterminacy of local behavior descriptions originates from the components' usage, the circuit topology, and physical thresholds. Verifying the correctness of a local behavior description needs an expensive simulation of the entire system in most cases.

Thus, model synthesis requires a cycle of the following steps: model selection, model simulation, physical evaluation, and model modification (cf. figure 8).

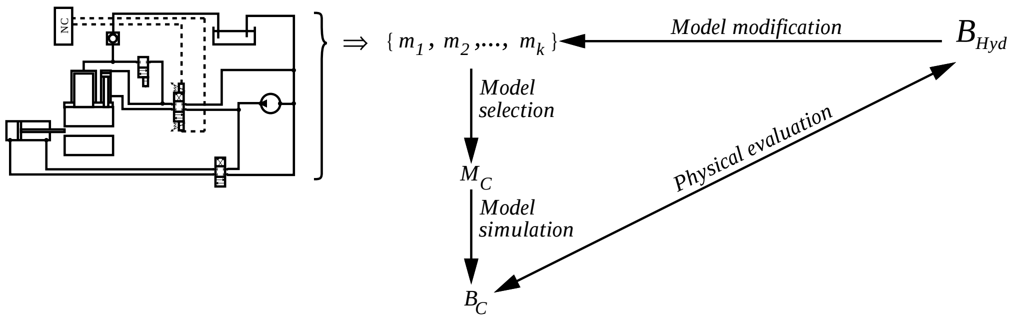

Figure 8: What model synthesis actually means.

Remarks. $\{m_1, \dots, m_k\}$ comprises the behavior descriptions, say, models of all hydraulic components in a circuit C. From this set a subset is selected ($= M_C$), simulated ($\Rightarrow B_C$), and compared to B_{Hyd}, which stands for the universal behavior laws of hydraulics. In the case that the simulated behavior B_C is physically contradictory or undetermined, M_C must be modified.

The cycle above constitutes an inherently combinatorial problem; it is solved when the physically correct behavior descriptions according to component usage, circuit topology, physical thresholds, and B_{Hyd} are determined.

Realization[5]

Model synthesis is not a deterministic procedure. There exist choice points where the valid behavior alternative must be selected, depending on the actual input values, parameter alternatives, or physical regularities. For each component o of a hydraulic circuit C, let $M_o = \{m_{o_1}, m_{o_2}, \dots, m_{o_k}\}$ be comprised of the k behavior alternatives of o. If a component o has a locally unique model, say, a pipe for instance, $|M_o| = 1$. Let $\mathbf{M}_C$ be the Cartesian product of the $M_o, o \in O$. Obviously, $\mathbf{M}_C$ defines the total synthesis search space.

Before all physical parameters of a hydraulic system C can be computed, a physically consistent model $M_C \in \mathbf{M}_C$ has to be determined. Conversely, whether a behavior model M_C is physically consistent can solely be verified via simulation. To illustrate the search for a consistent behavior model in $\mathbf{M}_C$, it is useful to think of the components' behavior constraints being divided

[5]The described concepts have been realized within the system $^{\text{art}}$*deco* [9].

into model selection constraints and behavior constraints. The search procedure can be outlined as a cycle containing the following inference steps:

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

Figure 9 illustrates the search process graphically.

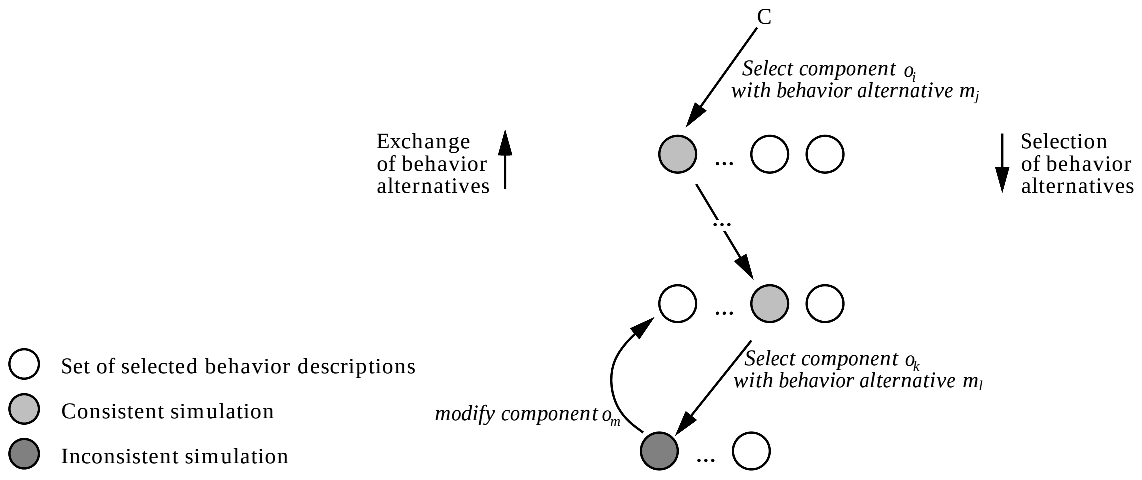
C
Select component o_i with behavior alternative m_j
Exchange of behavior alternatives
Selection of behavior alternatives
Select component o_k with behavior alternative m_l
modify component o_m
Set of selected behavior descriptions
Consistent simulation
Inconsistent simulation

Figure 9: Exploring the synthesis search space M_C.

Different components constrain the model synthesis step in a different manner. Hence, the order by which undetermined components are processed may play a crucial role. The challenge is the combination of standard numerical routines with AI-methods in such a way that the intelligent search behavior of a human expert is emulated.

Remarks. The evaluation of behavior constraints, mentioned in the simulation step of the above search procedure, is sophisticated. The components' behavior descriptions must be parsed, symbolic relations must be separated from numerical relations, equations have to be transformed, equation systems have to be formulated in some normal form, and rules have to be processed. Moreover, through the variety of constraint types and constraint dependencies, the backtracking mentioned in the model modification step will become a sophisticated job, too.

Application. The working document of a human expert is the circuit diagram. Consequently, it would be fair to specify hydraulic checking problems at the same level of abstraction. $^{\text{art}}$*deco* achieves this objective by a graphic problem specification. While drawing a circuit diagram, a knowledge base containing all necessary physical connections is created. The associated hydraulic system can immediately be checked concerning individual demands—the model formulation process is made transparent.

Discussion

A common idea to all presented configuration problems is that of a simplified knowledge acquisition procedure. The domain experts shall be supported within the complex and fault-prone process of the formulation of correct component models.

Note that, in our examples, the idea of a simplified knowledge acquisition goes always together with the concept of "locality". This is not a mere coincidence. For human experts local connections and local dependencies can often be easier formulated or verified than global connections—more precisely:

- ❑ Within the first example, the design of impellers for a given mixing task, the overall importance and the global weighting factors of the design rules are based on a collection of existing design cases. A single design case may be the most specialized, that is to say, the most local portion of design knowledge. Such an example can be checked easily by a domain expert. However, the generalization of this design knowledge is a model formulation problem for which no straightforward theory is at hand.
- ❑ Within the second example, the configuration of telecommunication systems, knowledge acquisition means the description of components by local functionalities. Clearly, the local impact of a functionality can be predicted much easier than its global consequences. If, on the other hand, the global role of each component had to be defined as well, the power of the resource-based component model would be lost.
- ❑ Also within the third example, the checking of hydraulic systems, the engineer is relieved from the problem of predicting the global role that a component will play in a large circuit. Instead the computer automates this part of the model formulation problem as it selects and composes all components' local models within a search process.

Obviously the advantages of a simplified knowledge acquisition procedure are bought with a considerable increase in the effort for knowledge processing. Loosely speaking, there is a tradeoff between knowledge processing cost on the one hand and knowledge acquisition cost on the other.

When realizing a component model that, primarily, falls back on local connections, the benefit of efficient knowledge processing is partly given up for the—often more desirable—benefit of user-friendly knowledge acquisition. Nevertheless, the examples two and three show how by component model compilation or by efficient component model synthesis the additional processing effort can be compensated.

References

[1] M. Brandner. Fallbasierte Klassifikation in der Verfahrenstechnik am Beispiel der Rührorganauswahl. Diploma thesis, Universität-GH Paderborn, 1994.

[2] A. Brinkop, N. Laudwein, and R. Maassen. Routine Design for Mechanical Engineering. In *IAAI '94 Proc. of the Sixth Annual Conference on Innovative Applications of AI, Seattle, August 1-3*, 1994.

[3] D. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.

[4] M. E. Cohen and D. L. Hudson. Approaches to the Handling of Fuzzy Input Data in Neural Networks. *IEEE International Conference on Fuzzy Systems*, 1992.

[5] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—An Approach to Domain-independent Construction. Technical Report 21, BMFT Verbundsprojekt, Universität Hamburg, FB Informatik, Mar. 1989.

[6] R. Fikes, T. Gruber, Y. Iwasaki, and A. L. P. Nayak. How Things Work - Project Overview. Technical Report KSL-91-70, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Nov. 1991.

[7] T. R. Gruber. Model Formulation as a Problem Solving Task: Computer-assisted Engineering Modeling. *International Journal of Intelligent Systems*, 8(1):105–127, 1992.

[8] M. Heinrich and E. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems for Modular Components. In *Proc. CAIA '91*, pages 257–264, 1991.

[9] H. Kleine Büning, D. Curatolo, M. Hoffmann, R. Lemmen, M. Suermann, and B. Stein. ARTDECO – Entwurfsunterstützung in der Hydraulik. *KI 5/95*, 1995.

[10] H. Kleine Büning, D. Curatolo, and B. Stein. Configuration Based on Simplified Functional Models. Technical Report tr-ri-94-155, Universität-GH Paderborn, Fachbereich Informatik, 1994.

[11] H. Kleine Büning, D. Curatolo, and B. Stein. Knowledge-Based Support within Configuration and Design Tasks. In *Proc. ESDA '94, London*, pages 435–441, 1994.

[12] T. Laußermair and K. Starkmann. Konfigurierung basierend auf einem Bilanzverfahren. In *6. Workshop "Planen und Konfigurieren", München*, FORWISS, FR-1992-001, 1992.

[13] R. Lemmen. Checking the Static and Dynamic Behaviour of a Hydraulic System. In *Proceedings of the 1th Asian Control Conference, Tokyo*, 1994.

[14] S. Marcus. *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, 1988.

[15] S. Marcus and J. McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1989.

[16] J. McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*, 19:39–88, 1982.

[17] P. P. Nayak. *Automated Model Selection*. Dissertation, Stanford University, 1992.

[18] B. Stein. *Functional Models in Configuration Systems*. Dissertation, University of Paderborn, Institute of Computer Science, 1995.

[19] B. Stein and J. Weiner. Model-based Configuration. In *OEGAI '91, Workshop for Model-based Reasoning*, 1991.

[20] M. Sueper. *Effiziente Lösungsstrategien für ressourcenorientierte Konfigurierungsprobleme*. Diploma thesis, Universität-GH Paderborn, FB 17 Mathematik / Informatik, 1994.