Configuration Based on Simplified Functional Models

Hans Kleine Büning Daniel Curatolo Benno Stein

University of Paderborn
Department of Mathematics and Computer Science
Warburger Straße 100
33095 Paderborn, Germany

Report No. tr-ri-94-155
Series Computer Science
November 1994

Configuration Based on Simplified Functional Models

Hans Kleine Büning Daniel Curatolo Benno Stein

University of Paderborn
Department of Mathematics and Computer Science
Warburger Straße 100
33095 Paderborn, Germany

1 Introduction

Configuration is the process of composing a technical system from a predefined set of objects. The result of such a process is called configuration too and has to fulfill a set of constraints given. Aside from technical restrictions a customer's demands constitute a large part of these constraints [6], [12], [13].

There exist a lot of methodologies that describe in which way configuration problems can be tackled. Their adequacy depends on the configuration task, the domain, and, of course, the description of the single configuration objects, called components. The type of the component description plays a key role in configuration since it determines both the quality of knowledge acquisition and the efficiency of knowledge processing.

In this place we deal with configuration problems where the components involved are characterized by simplified functional dependencies—so-called *resource-based* descriptions. We will discuss the philosophy of resource-based descriptions as well as their efficient processing.

The paper is organized as follows. Section 2 introduces structure-based and resource-based descriptions in configuration and elaborates on the related advantages and drawbacks. Section 3 presents concepts and methods of how a straightforward algorithm for resource-based configuration can be speedup decisively.

2 Structure-based versus Resource-based Descriptions

In a nutshell the mentioned types of component description can be characterized as follows. Let O denote the set of components from which a desired system is to be configured. Then, structure-based descriptions define *taxonomic* and *compositional* relations on particular subsets of O.[1] By contrast, resource-based descriptions are *local* to each component; they model the *properties* of the elements in O.

In the following both approaches are outlined.

[1] Sometimes, these relations are referred to as *is-a* and *has-parts* relations respectively.

Structure-based Descriptions

A structure-based description defines a (de)composition tree, that is, the skeleton of the system to be configured. It is convenient to represent the skeletal structure by means of an and-or-graph [8]. The and-nodes realize compositional descriptions while the or-nodes are suitable to realize taxonomic descriptions. Such a combined taxonomic/compositional hierarchy explicitly represents parts of the configuration problem's search space. Figure 1 shows a system that on its first layer is composed of the objects $o_1 \ldots o_3$, where o_1 and o_3 in turn can be realized by alternative subcomponents.

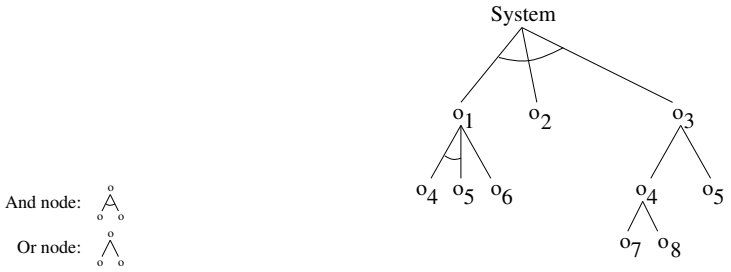

Figure 1: *A system's and-or-graph defines the possible configurations*

A basic method to process structure-based component descriptions is skeletal configuration. Among others it has been operationalized in the configuration systems WIST [5], PLAKON [1], [2], and R1/XCON [9]. Skeleton configuration can be put into practice by two main strategies:

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

2. *Bottom-Up.* If there is information about particular components that must be part of a configuration, the corresponding leaf nodes are initially marked. Secondly, all nodes sharing an and-relation with nodes already marked are also marked within a recursive bottom-up process. Thirdly, a top down refinement according to strategy 1 is invoked, which additionally considers all previously marked nodes.

Resource-Based Descriptions

Resource-based descriptions model a component's properties; properties are realized by means of "resources". Example: One property of power supply units is their power output; one property of plug-in cards is their power input. These properties can be modeled using the resource "power": A power supply unit *supplies* some power value, while each plug-in card *demands* some power value. I.e., for every interesting property of a component some resource f has to be defined, which is either supplied or demanded at a particular amount. In figure 2 resource-based component descriptions are depicted graphically.

Such a dependency network represents a simplified *functional* model of the domain. Actually, configuration means the instantiation and simulation of this functional model.

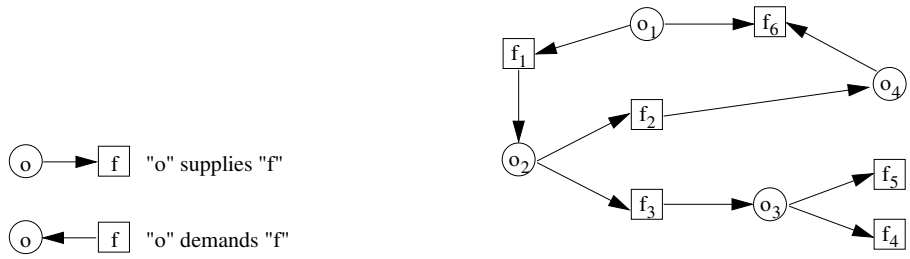

Figure 2: *Resource-based component descriptions*

Resource-based descriptions are suitable for a configuration problem if the following conditions are fulfilled: (*i*) Structural information plays a minor role only, (*ii*) the components can be characterized by simple resources that are supplied or demanded, and (*iii*) the components' properties have to be combined in order to provide the system's entire functionality.

Balance processing is a basic configuration method for resource-based component descriptions. It has been operationalized in the configuration systems COSMOS, CCSC, AKON, and MOKON [3], [7], [13], [10].

Balance processing operationalizes a generate-and-test strategy. The generate part, controlled by propose-and-revise heuristics, is responsible for selecting both an unsatisfied resource f and a set of objects that supply f. The test part simulates a balance.

Simplifying matters configuration works as follows. Initially all demanded resources are written on the demand side of the balance. Then, for an unsatisfied f, an object set is generated and the supplies and demands of these objects are also written on the corresponding sides of the balance. Identical resources are accumulated according to some rule, e.g. the algebraic "+"-operation. In a next step each resource on the balance is checked (e.g. via a "$\leq$"-comparison) whether the demanded value can be satisfied by the supplied one or not. Figure 3 depicts the generate and the test phase of this configuration method.

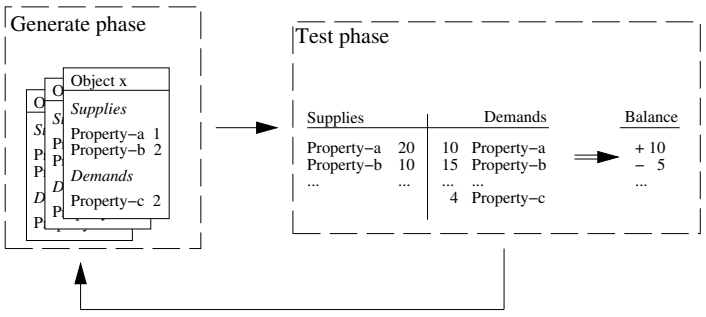

Figure 3: *The main phases of balance processing*

If all demands are fulfilled, the associated object set will represent a solution of the configuration problem. If not, the unsatisfied demands will form the input for the next step. The generate-and-test cycle is repeated until either a solution is found or no further object set can be generated.

Example

In order to understand some complexity problems bound up with resource-based component descriptions, it is useful to take a closer look at balance processing. Let us consider that 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 fulfill this demand; they are defined in the following table:

	Supplies	Demands
o_1	$2 \times f_1$, $1 \times f_2$,	-
o_2	$4 \times f_1$	$1 \times f_2$

Initially no node of the search space is still explored. The first balance contains only one entry:

Supplies	Demands	
	6	f_1

Now the configuration algorithm has to choose a component that fulfills the unsatisfied demand at property f_1. If we assume that component o_2 is chosen, the balance and the actually explored search space will look as depicted in figure 4.

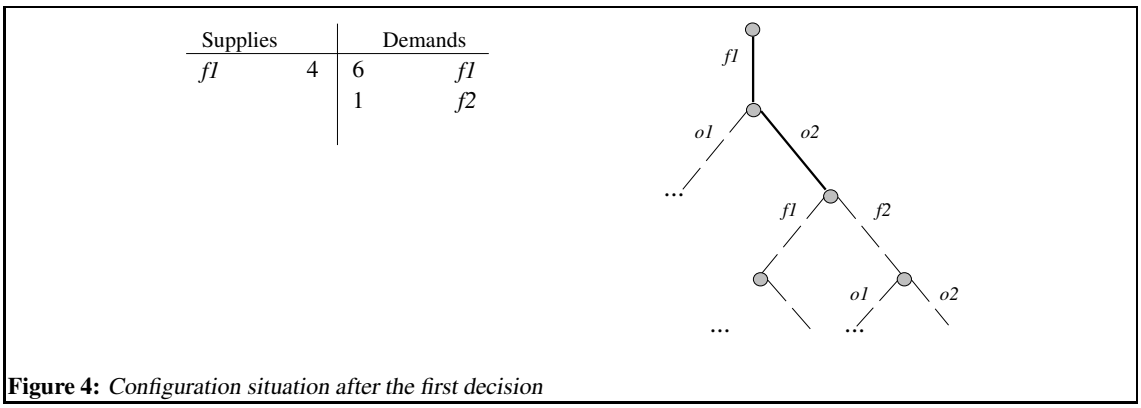

Figure 4: *Configuration situation after the first decision*

Given the configuration situation of figure 4, not only a decision regarding the component selection has to be made, but also the following question has to be answered: Which demand shall be satisfied next? A solution of our configuration example is given by a component set that contains o_1 once and o_2 once. Note that a set comprised of three components o_1, for example, establishes a solution too.

Discussion

From the standpoint of knowledge representation, structure-based descriptions define a *global* view on the system to be configured while resource-based descriptions rely on *local* connections only. From the standpoint of knowledge processing, structure-based descriptions form an *explicit* definition of the configuration process while resource-based descriptions constrain the configuration process *implicitly*.

Processing structure-based descriptions is efficient because of the semantics of compositional and taxonomic relationships (remember the top-down and the bottom-up strategy previously described). Nevertheless, knowledge acquisition is not easy since the modification, the exchange, or the addition of a component will affect the component's entire sub-skeleton. Moreover, structure-based descriptions establish no causal dependencies. Hence the configuration knowledge cannot be used easily to generate explanations of configuration steps.

Within resource-based descriptions the components' local properties are used to *derive* the necessary compositional and taxonomic dependencies. Compared to the structure-based approach, the configuration process exploits *deeper* dependencies of the domain. Note that configuration knowledge can be acquired and maintained easily here: Modification, exchange, and addition of components does not affect other parts (components, relations) of the configuration knowledge. Also expressive explanations of configuration decisions can be generated, since the configuration process's underlying model is functional, what's to say, causal.

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

Loosely speaking, there is a tradeoff between knowledge processing cost on the one hand and knowledge acquisition cost on the other. When realizing a resource-based component description instead of a structure-based one, the benefit of efficient knowledge processing is given up for the—often more desirable—benefit of user-friendly knowledge acquisition. Figure 5 illustrates this tradeoff qualitatively.

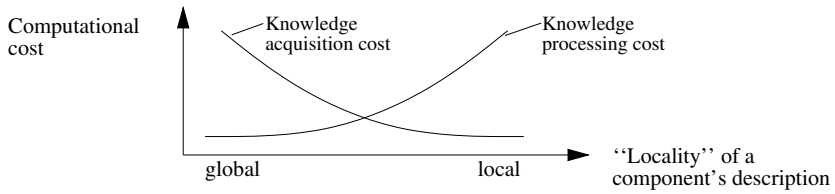

Figure 5: *Tradeoff between knowledge processing and knowledge acquisition cost*

3 Speeding Up Balance Processing

In the course of balance processing a lot of backtracking usually takes place. When selecting an unsatisfied property f, or when selecting components that supply f, several alternatives stand to reason. However, in order to find an *optimum* solution the entire search space must be investigated. Since the computational complexity of chronological backtracking is exponential we need techniques that explore the search space more intelligent [11].

Aside from standard techniques regarding efficient tree search, this section presents preprocessing techniques that we have developed to speedup balance processing. These considerations are not of a purely theoretical nature but have been operationalized within the configuration system AKON [4].

Balance Processing and Efficient Tree Search

Cutting Subtrees. In order to prune parts of the search space, a measure for the evaluation of expanded nodes is required. In connection with configuration tasks, a useful criterion is the cost of the single components and, consequently, the cost of a readily configured system. Note that due to the nature of configuration, the cost of a partially configured system will monotonically increase during the search.[2] Thus the minimum cost of all solutions found defines the bound for the subtrees to be cut.

As the following consideration shows, this procedure can be refined. If a property is unsatisfied, at least one additional component is needed. Hence, the value $c^* - \min\{c_o \mid o \in O\}$ establishes the actual upper bound at which backtracking must be invoked; c^* denotes the minimum cost of all solutions found up until that point, c_o denotes the cost of component o.

Preventing Multiple Investigation of Nodes. Different nodes in the search space may designate the same partial configuration. Put another way, there are different paths by which a node can be reached, i.e. a configuration can be constructed.[3] Consequently the nodes already investigated need to be marked.

It is useful to realize such a visited-flag by an entry in a hash table. For efficiency reasons, the hash function must be able to incrementally adapt hash keys: When a component is added and removed respectively from a partial configuration C', the new hash key should be computed on the base of the hash key that corresponds to C'. The configuration system AKON operationalizes this idea with a hash algorithm that is based upon an array of random numbers: Dependent on the component type and its frequency in a partial configuration, random numbers are chosen and computed by means of the XOR-operation.

Preprocessing

Preprocessing of configuration knowledge addresses the two indeterministic selection steps of balance processing: property selection and component selection. Property 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. Experience has shown that a simple local optimization is not sufficient here.

Note that the techniques subsequently presented establish *domain-independent* concepts. They can be improved and refined by additional domain knowledge, of course.

Property Selection Consider the component-property graph in figure 6. A demand at property f should be processed only if all components of the system that also need f are already determined. E.g., since component o_3 supplies nothing it should be selected first while o_1 demands nothing and should be selected last.

Obviously a component's occurrence in a configuration can be finally determined, if its outdegree in the component-property graph is zero (on condition that components selected and func-

[2] On condition that no backtracking is invoked, of course.

[3] The search space is rather represented by a directed acyclic graph instead of by a tree.

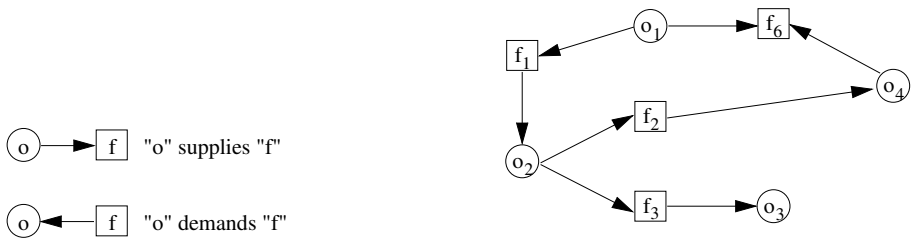

Figure 6: *Component-property graph*

tionalities processed are deleted in the graph). The sequence of nodes that we obtain by successively deleting nodes whose out-degree is zero constitutes a reversed topological sorting of the component-property graph. Note that a directed graph's topological sorting is not necessarily definite; it may not define a sufficient condition for a selection order of the properties.

Component Selection An example for a local optimization strategy regarding component selection is the following: *"To satisfy an open demand at property f, select from all components which supply f the cost minimum one."*

In fact, such a local strategy is often too shortsighted. What we are looking for is an estimation function that has *global configuration knowledge* compiled in. Ideally such a function should compute for each node of the search space a reliable estimation of the follow-up costs bound up with the selection of a particular component. In practice, for reasons of time and space complexity, such a function cannot be computed exactly.

The following simplifications are a reasonable compromise when constructing an estimation function: (*i*) A configuration situation shall solely be characterized by those resources that are actually unsatisfied, (*ii*) a resource 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 resource 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 an estimation function $h(o, f, n)$ for the computation of follow-up costs. f denotes the demanded resource, 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 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 properties of o. Then, of course, we would like each demand $v_d(o, g)$ of component o at property $g \in d(o)$ 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 resources 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 resource 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 resource f the number of components o that are necessary to satisfy f:

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

Now solely the composition of the above terms remains to be done. As a result, we obtain the following 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 resources of o
$o(f)$	components that supply f
$v_s(o, f) \in \mathbf{R}^+$	supply at resource f of component o
$v_d(o, f) \in \mathbf{R}^+$	demand at resource f of component o

The rationale of h is summed up as follows. The function h considers both a component's local cost and its follow-up costs. Thus it ensures the mathematical expectation of the search effort—which is equivalent to the estimation function's quality—being reasonable.

Example

The process of balancing and the effect of the estimation function h is illustrated now. Remember the example of section 2. We presented a simple knowledge base containing two components, o_1 and o_2, and two resources, f_1 and f_2. In this place we will elaborate on the same example. Notice that the table below defines aside from the components' supplies and demands also their costs.

	Supplies	Demands	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 derived above, the function h is defined as follows:

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

The demands on the system searched for shall be $6 \times f_1$. The resulting search tree is depicted in figure 7.

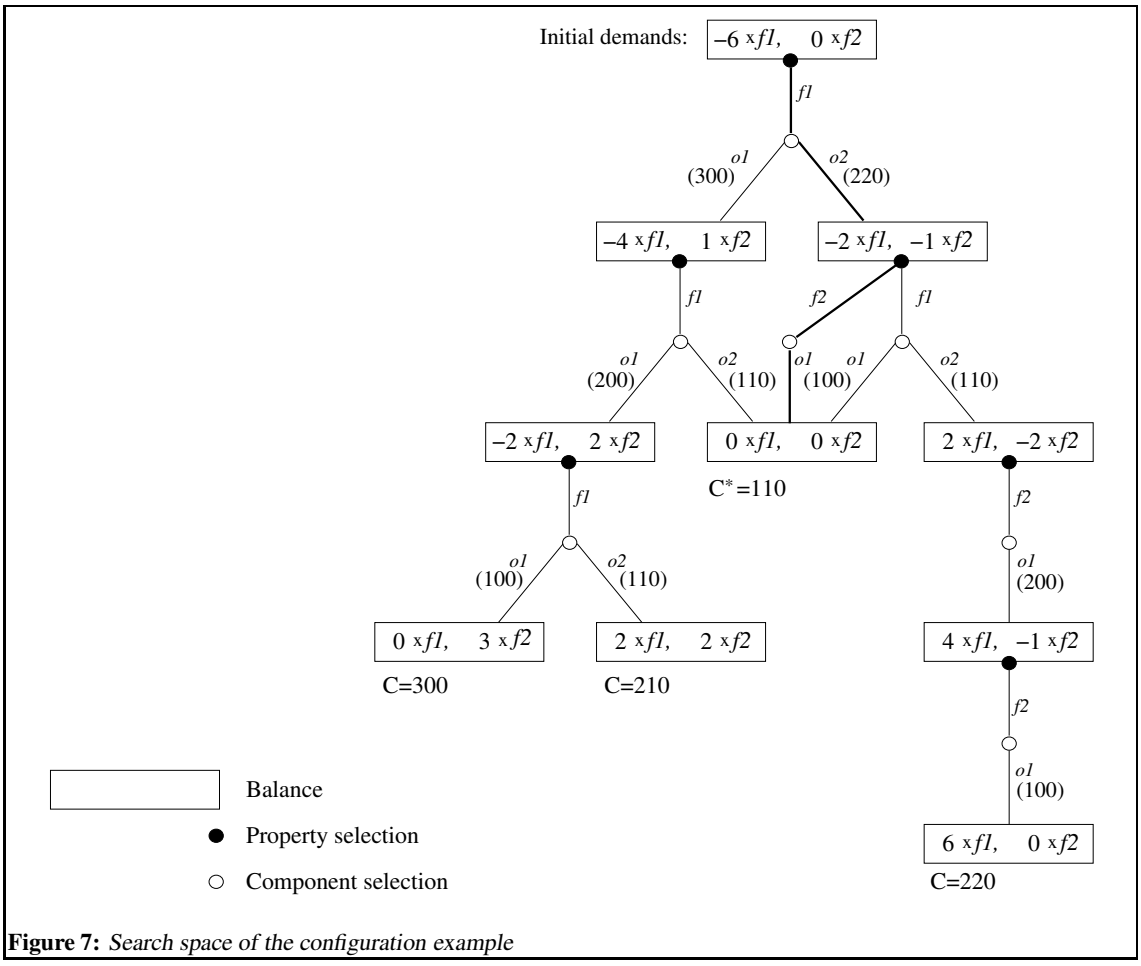

Figure 7: *Search space of the configuration example*

The search tree is two-layered and consists of two types of nodes:

- *Filled.* Nodes of this type establish choice points regarding the *resource* to be satisfied next. The related balance is shown framed above the node.
- ○ *Outlined.* Nodes of this type 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 search tree shows in which way the information of h comes to effect: The global optimum is found without backtracking, i.e. by a minimum search effort.

Discussion

The formerly introduced estimation function h should not be seen as an absolute search control for balance processing. Rather it establishes a useful heuristic that is based on the following assumptions: (*i*) a component's occurrence in a configuration is not bound to a fixed number, (*ii*) there are no restrictions between the components other than their supplies and demands, and (*iii*) no user-defined constraints need to be considered.

Dependent on an actual problem or domain, h can be improved or constructed according to other paradigms:

- Aside from a preprocessing that first selects a resource f and then decides which of the components o is suited best, a combined consideration of f and o is conceivable.
- Instead of computing an optimum estimation function with respect to *single* resources it might be useful to simultaneously consider particular combinations of *several* resources.

However, a total computation of the simplified estimation function h introduced above is hardly possible; even for rather small knowledge bases, which contain about hundred components, a few millions of values had to be recorded. To further complicate matters, for each situation characterized by o, f, and n, the determination of $h(o, f, n)$ requires a complete search.

There are two possibilities of how the computational effort regarding the evaluation of h can be decreased:

1. The step function $h(o, f, n)$ can be evaluated for a few n only. For all other values h is approximated via some interpolation method. This will bound the number of functions $h_{o,f}(n)$ to be computed and recorded by $|O| \cdot |F|$, where O (F) is a set comprised of all objects (resources) of the configuration problem.
2. The search space's total depth can be bound by some number k. If a search depth of k is reached while the balance is still unsatisfied, an approximate value estimating the remaining cost can be assumed. Also, dependent on the domain, it might be useful to predefine particular "obligatory" resources that need to be satisfied in any case. Note that early pruning may lead to the same problems as local optimization does.

4 Summary

In many configuration systems a taxonomic and a compositional hierarchy of the domain form the basis for the process of configuration. Such a modeling approach is no longer adequate if functional connections make up the major part of the domain knowledge.

One possibility to operationalize functional dependencies is resource-based modeling. The components are characterized by properties that are realized with supplied and demanded resources.

We pointed out that a resource-based modeling approach is superior to a structure-based one with respect to maintenance and acquisition of configuration knowledge. However, since the necessary structural dependencies have to be derived from the components' local properties, processing resource-based descriptions requires greater computational effort.

This additional computation effort can be mastered using techniques for efficient tree search and, in particular, through a *preprocessing* of configuration knowledge. We developed the idea of preprocessing related to resource-based configuration problems and showed in which way it is put into practice.

Bibliography

[1] 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.

[2] A. Günter. *Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen*. Dissertation, Universität Hamburg, Fachbereich Informatik, 1992.

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

[4] 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.

[5] H. Kleine Büning and S. Schmitgen. Konzept zur Lösung des Variantenproblems in der Stücklistenverarbeitung. In *CIM Management 2/88*. Oldenbourg Verlag, 1988.

[6] H. Kleine Büning and B. Stein. Entwicklung von Konfigurierungssystemen. In *WI '93 – Innovative Anwendungen, Technologie, Integration*, pages 287–302. Physica-Verlag, Heidelberg, 1993.

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

[8] F. Puppe. *Problemlösungsmethoden für Expertensysteme*. Springer-Verlag, 1990.

[9] E. Soloway, J. Bachant, and K. Jensen. Assessing the Maintainability of XCON-IN-RIME: Coping with the Problems of a VERY Large Rule-Base. In *Proceedings AAAI—Sixth National Conference on Artificial Intelligence*. Morgan Kaufmann, July 1987.

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

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

[12] W. Tank. *Modellierung von Expertise über Konfigurierungsaufgaben*. Dissertation, Universität (TU) Berlin, Fachbereich Informatik, 1992.

[13] J. Weiner. *Aspekte der Konfigurierung technischer Anlagen*. Dissertation, Gerhard-Mercator-Universität - GH Duisburg, FB 11 Mathematik / Informatik, 1991.