Journal of Applied Intelligence , 10, 247–255 (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Generating Heuristics to Control Configuration Processes[⋆]

BENNO STEIN
Department of Mathematics and Computer Science—Knowledge-based Systems Group
University of Paderborn, 33095 Paderborn, Germany

stein@uni-paderborn.de

Abstract. Configuration is the process of composing a system from a set of components such that the system fulfills a set of desired demands. The configuration process relies on a particular component model, which is a useful abstraction of the domain and the technical system to be composed.

In this place we deal with configuration problems where the components involved are characterized by simplified functional dependencies, so-called *resource-based* descriptions. On the one hand, the resource-based component model provides for powerful and user-friendly mechanisms to formulate configuration tasks. On the other hand, the solution of resource-based configuration problems is NP-complete, which means that no efficient algorithms exist to solve a generic instance of that problem.

In practice, given a concrete resource-based component model, the search for an optimum configuration can be realized efficiently by means of heuristics that have been developed by domain experts. The paper in hand picks up that observation: It presents a method to automatically generate heuristics that guide the search when solving complex resource-based configuration problems.

Keywords: configuration, knowledge-based systems, heuristic search, preprocessing, design

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 given constraints. Aside from technical restrictions a customer's demands constitute a large part of these constraints [2, 3, 4, 7, 9].

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 [16, 12]. The formulation of adequate component models is a highly creative job and cannot be automated in its universality [13].

In this place we deal with the *resource-based* component model; within this model the components involved are characterized by simplified functional dependencies, so-called resources. On the one hand, the resource-based component model provides for powerful and user-friendly mechanisms to formulate configuration tasks [6]. On the other hand, the solution of resource-based configuration problems is NP-complete, which means that no efficient algorithms exist to solve a generic instance of that problem [12].

Actually, when given a real-world configuration problem formulated within the resource-based component model, the search for an optimum configuration can be realized efficiently by means of heuristics that were developed by domain experts. Stated another way, a concrete resource-based component model can be *compiled* by enriching it with control knowledge. The pa-

[⋆] Reworked and extended version of a conference paper originally published at the 11th IEA/AIE, 1998.

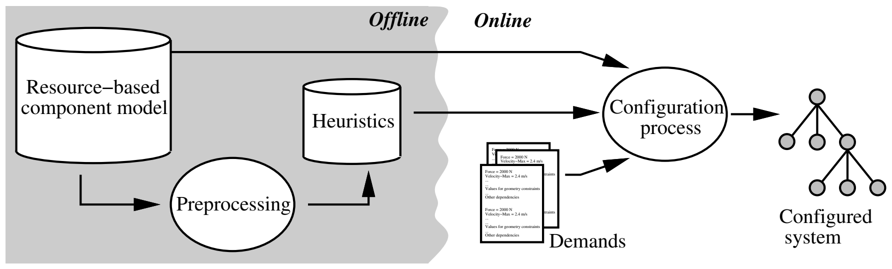

Fig. 1. Partitioning the resource-based configuration process.

per in hand picks up that observation: It presents a method to *automatically generate heuristics* that guide the search when solving complex resource-based configuration problems.

Our approach emphasizes the view that a resource-based configuration problem can be attacked at two different scenes: At a preprocessing stage, where heuristics are generated, and at a configuration stage, usually at the customer's site, where a concrete configuration problem is solved. Figure 1 illustrates this view.

That a partitioning of the configuration process is possible is in the nature of most configuration problems: Input of the preprocessing step is the entire component model, and preprocessing must be re-applied whenever the component model is changed. Such changes come along when a new components is added or when properties of existing components are modified. Input of the configuration step are demand sets, which are customer-dependent. The ratio of component model changes and demand set changes is $\ll 1$.

The purpose of this paper is twofold.

1. In the subsequent section, resource-based configuration is introduced, its pros and cons are discussed, and the balance algorithm, a method to process resource-based component models, is outlined.
2. The performance of the standard balance algorithm can be significantly improved by exploiting heuristics that provide a decision base during the search. This is the starting point of Section 3, where we show in which way such heuristics can be derived within a preprocessing step.

The concepts presented in this paper have been operationalized and evaluated for a real-world configuration problem [7], which is outlined in Section 4.

2. Resource-Based Configuration

There exist a lot of methodologies that describe in which way configuration problems can be tackled. Their adequacy depends on the configuration task, the domain, and, of course, the description of the single configuration objects, called components. Especially when configuring modular technical devices, resource-based configuration is an important configuration methodology.

2.1. The Resource-Based Component Model

The resource-based component model establishes an abstraction level that reduces a domain to a finite set of functionality-value-pairs. More precisely, all technical properties that are relevant for the configuration process form a set of resources, which are supplied or demanded by the components [5].

E. g. when configuring a small technical device such as a computer, one property of power supply units could be their power output, and one property of plug-in cards could be the cards' power consumption. Both properties are reflected by the resource "power": A power supply unit *supplies* some power value, while each plug-in card *demands* some power value. Figure 2 depicts some resource-based descriptions of computer components.

Note that dependency networks as shown in Figure 2 represent a simplified functional models of the domain. Actually, resource-based configuration means the instantiation and simulation of such a functional model.

The resource-based component model is suitable for a configuration problem if the following conditions are fulfilled:

- Structural information plays only a secondary role.
- The components can be characterized by resources that are supplied or demanded.

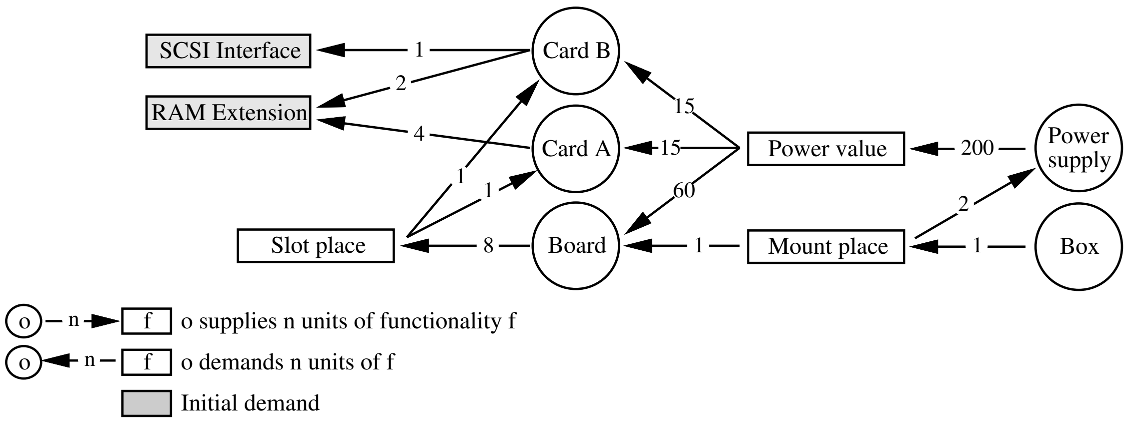

Fig. 2. Resource-based modeling of simple computer components.

- The components' properties are combined in order to provide the system's entire functionality.

In the following we give a precise specification of the simplified resource-based configuration problem and its solution.[1]

Definition 2.1 (Configuration Problem). A simplified resource-based configuration problem Π is a tuple $\langle O, F, P, D\rangle$ whose elements are defined as follows.

1. O is an arbitrary, finite set. It is called the *object set* of Π.
2. F is an arbitrary, finite set. It is called the *functionality set* of Π.
3. For each object o there is a *property set*, p_o, which contains pairs (f, x), where $f \in F$ and $x \in \mathbf{Z}$, and each functionality $f \in F$ occurs at most once in p_o. $P = \{p_o \mid o \in O\}$ is comprised of these property sets. A property set specifies the values of certain functionalities of a given object.
4. D is an arbitrary, finite set of *demands*. Each demand d is a pair (f, x), where $f \in F$ and $x \in \mathbf{Z}$, and each functionality $f \in F$ occurs at most once in D. A demand set describes the desired properties of the system to be configured.

The resource-based component model distinguishes between supplied and demanded properties of the components. This semantics is not reflected explicitly by the objects' property sets p_o, but can be modeled easily by using positive and negative functionality values for supplies and demands respectively.

Definition 2.2 (Configuration). Let $\Pi = \langle O, F, P, D\rangle$ be an instance of the above configuration problem. A configuration C is a set of *items* of the form (o, k), stating that object $o \in O$ is used k times in the configured system. In analogy to an object's property set, p_C denotes the configuration's property set, and its elements are canonically defined as follows:

$$\text{If } (f, x) \in p_o, (o, k) \in C \text{ then } (f, z) \in p_C,$$
$$\text{with } z = \sum \{k_i \cdot x_i \mid (o_i, k_i) \in C \wedge (f, x_i) \in p_{o_i}\}.$$

Definition 2.3 (Solution). A configuration C is a solution of a configuration problem $\Pi = \langle O, F, P, D\rangle$ if and only if for each demand $d = (f, x) \in D$ there exists a property $(g, y) \in p_C$, such that $f = g$ and $x \leq y$.

2.2. Processing Resource-Based Descriptions

If there exists a configuration C that solves the resource-based configuration problem, C can be determined with the balance algorithm. This algorithm operationalizes a generate-and-test strategy and has been implemented in the configuration systems COSMOS, CCSC, AKON, and MOKON [5, 8, 17, 14]. The generate part, controlled by propose-and-revise heuristics or simply by backtracking [10], is responsible for selecting both an unsatisfied functionality f and a set of objects that supply f. The test part simulates a virtual balance. A functionality (resource) is called unsatisfied, if its supplied amount x is smaller than its demanded amount y.

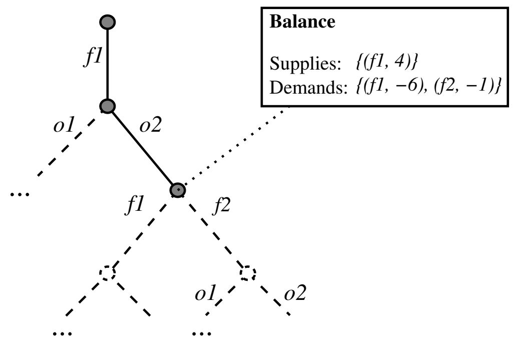

Fig. 3. Configuration situation after the first decision.

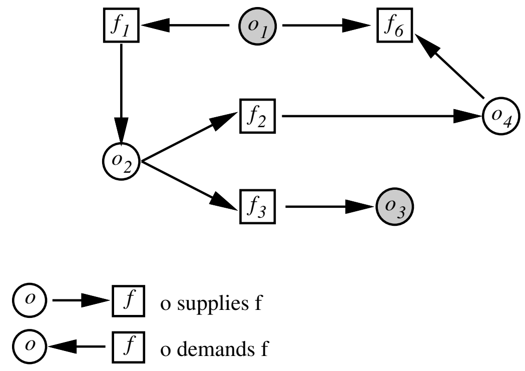

Fig. 4. Object-functionality graph.

Basically, configuration works as follows. First, the demand set of the virtual balance is initialized with all demanded functionalities, and C is set to the empty set. Second, with respect to some unsatisfied f, an object set is formed; its related supplies and demands are added to the corresponding functionalities of the balance, and C is updated by the object set. Third, it is checked whether all functionalities are satisfied. If so, C establishes a solution of the configuration problem. Otherwise, the configuration process is continued with the second step.

Consider a simple configuration problem where an initial demand set $D = \{(f_1, -6)\}$ is given. Two components, o_1 and o_2, can be used to fulfill this demand; they are defined in the following table:

Properties o_1	$\{(f_1, 2), (f_2, 1)\}$
Properties o_2	$\{(f_1, 4), (f_2, -1)\}$

After initialization, the balance has a single entry, $(f_1, -6)$. 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 3.

Note that in the configuration situation of Figure 3, both a decision regarding component selection and functionality selection must be made. A solution of the example is given with $C = \{(o_1, 1), (o_2, 1)\}$ or with $C = \{(o_1, 3)\}$.

3. Speeding Up Balance Processing

Resource-based component models provide great knowledge acquisition support since the configuration knowledge consists of local connections for the very most part [12]. However, the basic balance processing algorithm is of exponential time complexity.

In many real-world configuration problems the *optimum* solution must be found for a given demand set D. Hence, if no powerful heuristics are at hand that control the functionality and component selection steps, merely small configuration problems can be tackled by resource-based configuration.

Functionality selection is related to the search space's total depth in first place; component selection affects the effort necessary for backtracking. An "intelligent" strategy within these selection steps is the major engine of efficient balance processing [15].

In this connection the preprocessing idea comes into play. By preprocessing the resource-based component model, *implicit knowledge* relating a selection strategy *can be made explicit*, e. g. in the form of an estimation function. The following subsections outline preprocessing techniques. The considerations are not of a purely theoretical nature but have been operationalized within the configuration system PREAKON [7].

3.1. Preprocessing Relating Functionality Selection

Consider the object-functionality graph in Figure 4. A demand $(f, x) \in D$ should only be processed, if all components of the system that also need this functionality are already determined. E. g., since component o_3 supplies nothing, it should be selected first, and while o_1 demands nothing, it should be selected last.

Obviously, the number of o's instances required to satisfy a functionality can be determined without backtracking, if o's outdegree in the object-functionality graph is zero (on condition that the components selected and the functionalities processed are deleted in the graph). Note that the sequence of nodes we get by this procedure corresponds to a reversed topological sorting of the graph. The order by which functionalities occur in this sorting defines the succession by which unsatisfied functionalities should be selected from the balance.

Remarks. Because a object-functionality graph $g(V, E)$ might contain cycles, all strongly connected components of g must be detected first. This computation can be done in $O(|E|)$ for a connected directed graph [1]. Then, given the strongly connected components, the condensed graph can be constructed where each strongly connected component is represented by one node.

3.2. Preprocessing Relating Component Selection

One example for a component selection strategy is the following:

> *"To satisfy an open demand at functionality f, select from all components that 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:

1. A configuration situation shall solely be characterized by those functionalities that are currently unsatisfied.
2. A functionality shall only be satisfied by components of the same type.
3. Components shall be regarded as suppliers of a single resource.

Remarks. Point 1 neglects that unused resources in a partial configuration may be exploited in a further course of the configuration process. Point 2 neglects that a combination of different components may constitute a more adequate solution for an unfulfilled functionality than a set of components of the same type. Point 3 neglects that a component may supply several resources each of which is demanded in the partial configuration.

Based on the above simplifications an estimation function $h(o, f, n)$ for the computation of follow-up costs can be directly constructed.[2] f denotes the demanded functionality, n denotes the amount to which f is demanded, and o denotes a component that supplies f and that is 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 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 $c(o) \in \mathbf{R}^+$ is the local cost of component o, $d(o)$ the demanded functionalities of o, and $o(f)$ are 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 introduce a term that computes for a given amount n at functionality f the number of components o that are necessary to satisfy f:
$$\left\lceil \frac{n}{v_s(o, f)} \right\rceil$$

Putting the pieces together results in an 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 $c(o) \in \mathbf{R}^+$ is the local cost of component o, $d(o)$ the demanded functionalities of o, $o(f)$ are components

that supply f, $v_s(o, f) \in \mathbf{R}^+$ is the supply at functionality f of component o, and $v_d(o, f) \in \mathbf{R}^+$ is the demand at functionality f of component o.

3.3. Configuration Example

Remember the example of Section 2 where a simple knowledge base containing two components, o_1 and o_2, and two functionalities, f_1 and f_2 was given. In this place we elaborate on the same example, but we define aside from the components' properties also their local costs c:

	Properties p_o	Local cost
o_1	$\{(f_1, 2), (f_2, 1)\}$	100
o_2	$\{(f_1, 4), (f_2, -1)\}$	10

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

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

The demands at the system searched for are $D = \{(f_1, -6), (f_2, 0)\}$. The resulting search tree is depicted in Figure 5.

Remarks. The search tree is two-layered and consists of two types of nodes. Filled nodes establish choice points regarding the functionality to be satisfied next. The related balance is shown framed above the node. Outlined nodes establish choice points regarding the component to be selected next. The edges of the search tree are labeled with the configuration decisions. Below the actually selected components, put in parentheses, the estimation function's values are annotated.

The tree shows in which way the control information of h is exploited. If alternative components are given to satisfy an open demand, h defines an order by which components shall be tried. In the example, component o_2 is chosen at the first choice point, while o_1 is chosen at the next. The earlier a solution is found, the earlier its cost information c can be used to cut off partial configurations exceeding c. Note that h cannot be a function that estimates a configurations total cost.

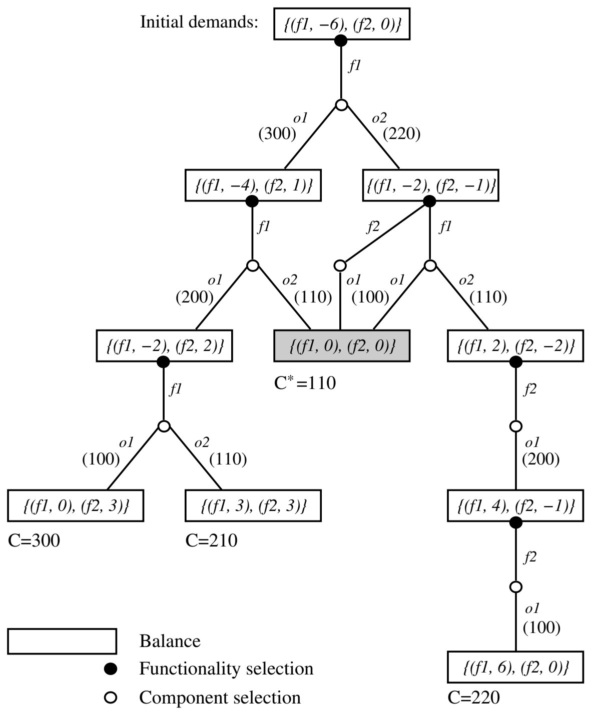

Fig. 5. Search tree of the configuration example.

4. Application

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 real-world applications, since an exhausting search for all values in o, f, and n had to be performed.

The telecommunication application outlined below is such a real-world application.

4.1. 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 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.

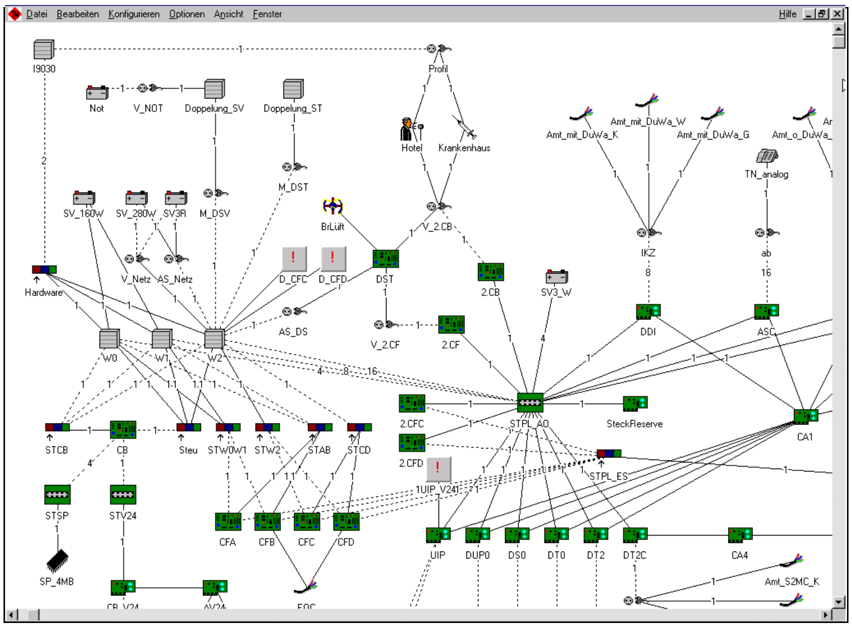

Fig. 6. Graphical representation of a knowledge base of a Telenorma telecommunication system. Dashed lines represent supply dependencies, solid lines indicate demand dependencies.

For this kind of domain and configuration problem the resource-based component model establishes the right level of abstraction: Technical constraints are allowed to be reduced to 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 along with our working group the development of the resource-based configuration system PREAKON [7]. Early versions of PREAKON showed the necessity of a heuristic search space exploration, if optimum configurations should be found in an acceptable time when given realistic demand sets D.

However, for the following reasons we refrained from a *manual* integration of such heuristic control knowledge:

1. Discussions with domain experts revealed that their control knowledge often was contradictory and incomplete.
2. The additional maintenance effort would have complicated the configuration system's introduction.
3. Each modification of the knowledge base (e. g. because of new components) could have invalidated existing heuristics.

Instead, we pursued the concept presented in this paper—the automatic generation of control knowledge by means of preprocessing.

Order of Magnitude Background. A typical knowledge base in this telecommunication application consists of more than 100 components providing about 200 functionalities; on average each component supplies and demands 8 functionalities; and a configuration problem usually starts with more than 20 initial functionalities each of which representing a customer demand. Figure 6 shows a part of the graphical representation of a telecommunication knowledge base in the PREAKON configuration system.

Clearly, an exact computation of the estimation function h is not possible here. As a way out, aside from the simplifying assumptions already made, h can be approximated in the following way:

Depending on both the functionalities, $f \in F$, and the components, $o \in O$, bound the recursion depth of h 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 is assumed. Based on an evaluation in a few points, interpolate h.

These simplifications result in a family of $\mathcal{O}(|O| \cdot |F|)$ functions $h_{o_f}(n)$, which can be evaluated at configuration runtime.

Exactly this way estimation functions have been employed within PREAKON. The result was a significant speed-up for realistic instances of Telenorma's configuration problem; PREAKON was the first configuration system at Telenorma that provided realistic means for being used at the customer site. Technical details, an evaluation, and a comparison of the outlined as well as of related estimation functions can be found in [15].

4.2. Discussion

The estimation function h should not be seen as an absolute search control for balance processing. However, it establishes a useful heuristic that is based upon several assumptions:

- A component's occurrence in a configuration is not constrained to a fixed number.
- There are no restrictions between the components other than their supplies and demands.
- No user-defined constraints need to be considered.

Depending on both the actual configuration problem and specialties of the 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.
- When bounding the recursion depth of h, it is useful to predefine "obligatory" resources that need to be satisfied in any case.

5. Summary

Each configuration process relies on a particular component model, which realizes a useful abstraction of the domain and the technical system to be composed. In this paper we focused on the resource-based component-model.

The resource-based component-model comes along with important configuration benefits respecting knowledge acquisition and knowledge maintenance. These advantages are bought with a considerable increase in knowledge processing cost. However, when taking a look at configuration practice, we see domain experts formulating heuristics that make resource-based configuration a working concept.

This observation has been picked up here. We developed the idea of a preprocessing related to resource-based configuration problems, and we showed how it is put into practice:

1. Off-line, within a preprocessing phase, heuristics that control the configuration process are generated automatically. These heuristics, which are encoded in the form of estimation functions, give an approximate estimate of the follow-up cost for each configuration object o. The follow-up cost recursively sum up o's total cost consequences, if o is used in the system to be configured.
2. Online, during the actual configuration process, this heuristic control knowledge is used to guide component selection, when several alternatives stand to reason to satisfy an open demand.

A particular approximation of the estimation function was realized and evaluated for the configuration of large telecommunication systems at Bosch/Telenorma, Frankfurt. The operationalization of this kind of control knowledge emerged to be the key factor for an efficient configuration process.

Notes

1. Compared to the original definitions in [11], the definitions here are weakened within the following respects:
(a) The value set of all functionalities is $\mathbf{Z}$, (b) the only way to combine two functionalities is the addition of their values, and (c) supplies and demands are compared with the "$\leq$"-operator.
Thus symbolic functionalities or sophisticated configuration constraints cannot be formulated straightforward. However, the definition reflects a great deal of the required modeling power for typical resource-based configuration problems.
2. The estimation function discussed here was proposed and operationalized by D. Curatolo [6].

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Massachusetts, 1983.
2. David C. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.
3. R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—An Approach to Domain-independent Construction. Technical Report 21, BMFT cooperation project, University of Hamburg, Department of Computer Science, March 1989.

4. John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.
5. 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.
6. Hans Kleine Büning, Daniel Curatolo, and Benno Stein. Configuration Based on Simplified Functional Models. Technical Report tr-ri-94-155, University of Paderborn, Department of Mathematics and Computer Science, 1994.
7. Hans Kleine Büning, Daniel Curatolo, and Benno Stein. Knowledge-Based Support within Configuration and Design Tasks. In *Proc. ESDA '94, London*, pages 435–441, 1994.
8. T. Laußermair and K. Starkmann. Konfigurierung basierend auf einem Bilanzverfahren. In *6. Workshop "Planen und Konfigurieren", München*, FORWISS, FR-1992-001, 1992.
9. Mary Lou Maher. Process Models for Design Synthesis. *AI Magazine*, pages 49–58, 1990.
10. Sandra Marcus and John McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1989.
11. Oliver Najmann and Benno Stein. A Theoretical Framework of Configuration. In *Proc. IEAAIE '92*, Paderborn, 1992.
12. Benno Stein. *Functional Models in Configuration Systems*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1995.
13. Benno Stein and Daniel Curatolo. Model Formulation and Configuration of Technical Systems. In Jürgen Sauer, Andreas Günter, and Joachim Hertzberg, editors, *10. Workshop "Planen und Konfigurieren", Bonn*, volume 3 of *Proceedings in Artificial Intelligence, ISBN 3-92037-97-1*, 1996.
14. Benno Stein and Jürgen Weiner. Model-based Configuration. In *OEGAI '91, Workshop for Model-based Reasoning*, 1991.
15. Martin Sueper. *Effiziente Lösungsstrategien für ressourcenorientierte Konfigurierungsprobleme*. Diploma thesis, University of Paderborn, Department of Mathematics and Computer Science, 1994.
16. Christopher Tong. Towards an Engineering Science of Knowledge-based Design. *Artificial Intelligence in Engineering*, 2(3):133–166, 1987.
17. Jürgen Weiner. *Aspekte der Konfigurierung technischer Anlagen*. Dissertation, Gerhard-Mercator University of Duisburg, Department of Computer Science, 1991.

Benno Stein is researcher in the Knowledge-based Systems Group at the Department of Mathematics and Computer Science, University of Paderborn. He conducts research in both knowledge-based analysis and synthesis tasks, focusing on the automation of complex design problems. His special field of research is the operationalization of engineering know-how within model formulation problems.
Dr. Stein has received degrees from the Universities of Karlsruhe and Paderborn.

⋆ Reworked and extended version of a conference paper originally published at the 11th IEA/AIE, 1998.