

A Theoretical Framework for Configuration*

O. Najmann B. Stein

FB 17, Praktische Informatik, Universität-GH-Paderborn

Abstract: We develop a general theory of configuration and give a precise methodology for studying the phenomenon of configuration from a viewpoint which is independent of any knowledge representation formalism. One main result is that we show that the classical skeleton-oriented approach and the resource-oriented approach are in some sense equivalent. We formulate a number of typical configuration problems, like finding a cost-minimal configuration, and show the NP-completeness of one of them.

1 Introduction

Manufacturers of complex technical systems are faced with the problem to satisfy the exceptional demands of their customers in order to stay competitive. Due to the enormous number of possible variants, the process of configuring a technical system became a sophisticated problem. Therefore, many systems have been developed which assist humans in this process. However, only few attempts have been made to formally analyze the nature of typical configuration problems.

The main contribution of this paper is that it provides two models of configuration. A basic idea of these models is that all involved parts of the technical system are solely described by their functionalities. Since we want to describe the configuration problem in a general way, the presented approach does not presuppose any particular knowledge representation formalism.

The paper is arranged as follows: First, a short introduction to configuration problems is given and the idea of a formal description is motivated. Second, two models, M1 and M2, are presented. Model M1 allows to formulate *resource-oriented* configuration problems [Heinrich, 1991; Stein & Weiner 1990], while model M2 allows to formulate *skeleton-oriented* configuration problems [Puppe, 1990]. Several problems that can be formulated under these models are presented. Although model M2 additionally contains restriction rules, it is shown that M1 and M2 are equivalent. This means that it is principally possible to transform explicit knowledge about the structure of a technical system into a set of object functionalities without loss of information. Finally we consider the computational complexity of configuration and show that the general configuration problem is NP-complete.

The operational point of view. Configuration problems are characterized by the fact that the solution (which is the completely configured object) is composed of smaller subobjects. A typical example is the configuration of computers: Particular components such as a harddisk, a processor, etc. are given. The problem is to select components in such a way that their composition fulfills a customer's demand.

A lot of studies in the field of configuration deal with technical aspects of the problem, i.e., description of ideas, concepts, and techniques *how* a certain configuration problem can be solved. Such a pragmatismal, teleological point of view is justified. It is caused by the complexity of the general problem "configuration"; most of the configuration systems represent solutions of particular cases.

In this context, PLAKON [Cunis et al., 1991] seems to be the highest developed system dealing with configuration tasks. Also AMOR (cf. [Tank et al., 1989]), a description language for technical systems, should be mentioned here. Brown and Chandrasekaran [1989] developed DSPL, a knowledge representation language for routine design problems.

A theoretical approach. Rather than specifying a particular method of processing or acquiring configuration knowledge, we aim at a formal description of general configuration problems. Such a description could be used to distinguish between different configuration approaches or to compare certain configuration problems with respect to their complexity. One approach, pointing at a similar direction, is the configuration model of Dörner [1991].

In our approach, the central idea is the notion of *functionality*. Objects are described solely by functionalities. When objects are selected, functionalities are composed in order to specify the functionality of the whole system. Because of the functionality-centered approach, our model is especially appropriate to describe problems where the *selection* of components comes to face.

Definition: A configuration problem consists basically of three things, (i) a set of objects, (ii) a set of functionalities which are used to describe certain properties of these objects, and (iii) a set of demands which describe the desired properties of the system to be configured.

2 Model M1

Subsequently, we give a formal definition of the notion *configuration problem*. We will distinguish between the problem itself and the notion *configuration*, which is viewed as the solution of a particular configuration problem.

The system MOKON [Stein & Weiner 1990] operationalizes important parts of model M1. So, the given formalization can be used to distinguish between different classes of configuration problems on the one hand as well as to determine limits of particular domain descriptions on the other (regarding their computer-supported solution).

Definition: A *configuration problem* Π is a tuple $\langle O, F, V, P, A, T, D \rangle$ whose elements are defined as follows:

- O is an arbitrary finite set, it is called the *object set* of Π .
- F is an arbitrary finite set, it is called the *functionality set* of Π .
- For each functionality $f \in F$ there is an arbitrary finite set v_f , called the *value set* of f . $V = \{v_f | f \in F\}$ comprises these value sets.
- For each object o there is a *property set*, p_o , which contains pairs (f, x) , where (i) $f \in F$ and $x \in v_f$, and (ii) each functionality $f \in F$ occurs at most once in p_o . A property set specifies the values of certain functionalities of a given object. $P = \{p_o | o \in O\}$ comprises these property sets.
- For each functionality f there is an *addition operator* a_f which is a partial function $a_f : v_f \times v_f \rightarrow v_f$. An addition operator specifies how two values of a functionality can be composed to a new value if a new object is *added* to a given collection of objects which themselves describe a part of the system to be configured. $A = \{a_f | f \in F\}$ comprises all addition operators.

- For each functionality f there is a *test* t_f which is a partial function $t_f : v_f \times v_f \rightarrow \{\text{TRUE}, \text{FALSE}\}$. A test t_f specifies under what condition a demand (see below) is fulfilled. $T = \{t_f | f \in F\}$ comprises all tests.
- D is an arbitrary, finite set of *demands*. Each demand d is a pair (f, x) , where $f \in F$ and $x \in v_f$. Additionally, the demand set must have the property that no functionality occurs more than once in D . A demand set D describes the desired properties of the system to be configured.

Remarks: O contains objects which can be composed to a system meeting a certain requirement. For example, if we had to configure a computer system, then typical objects of O would be a harddisk, a CPU, a power supply, etc.

Furthermore, functionalities describe certain properties of the objects in O . For example, the harddisk may have the functionalities “capacity,” “access time,” etc. Then for example, $v_{\text{capacity}} = \{10, 20, 30, 40\}$ [Megabytes].

A typical example of an addition operator is the calculation of the entire capacity of a set of harddisks. For example, a_{capacity} can be defined as follows (where the symbol \perp is used wherever a_{capacity} is undefined):

$$a_{\text{capacity}}(x, y) = \begin{cases} x + y, & \text{if } x + y \leq 40; \\ \perp, & \text{otherwise.} \end{cases}$$

The addition operator in this example can be used to allow multiple use of a harddisk in order to increase the amount of harddisk capacity. An addition operator needs not necessarily specify an *addition* between two numbers, but any kind of operation is possible.

Every object $o \in O$ is characterized by its set of properties, p_o . For example, a particular harddisk can have the following set of properties: $p_{\text{harddisk}} = \{(\text{capacity}, 10), (\text{access-time}, 7)\}$. Note that every functionality can occur at most once in a property set.

Since we have to compose the values of functionalities with respect to a given demand, it is necessary to introduce a test t_f for each functionality $f \in F$. For example, a typical test for “capacity” is the “ \geq ” predicate.

An example of a demand set is $D = \{(\text{capacity}, 30), (\text{mouse}, \text{yes}), (\text{keyboard}, \text{english})\}$.

So far we have only defined the notion “configuration problem”; now we have to define what a solution of such a problem is.

Definition: A solution of a configuration problem must fulfill two conditions: 1. It must be a *configuration*, which is defined below. 2. This configuration must meet all demands of the demand set D .

A configuration contains both, objects and functionalities. Before we give an inductive definition of a configuration, we define how properties can be composed.

Definition: Let $(f, x), (g, y)$ be two properties.

$$\varphi((f, x), (g, y)) = \begin{cases} \{(f, a_f(x, y))\}, & \text{if } f = g; \\ \{(f, x), (g, y)\}, & \text{otherwise.} \end{cases}$$

is called the *composition* of the properties (f, x) and (g, y) .

Remarks: The composition of two properties is a set. This set contains either a single property if the functionalities are equal, or it contains these two properties if they are not equal. The rationale of this composition is as follows: If two objects, which have some properties in common, are included in a set containing the configuration objects, then it is necessary to “compute” the values of these properties in some way. The computation is done by the addition operator. If the addition operator is not defined for the given value constellation, these two objects may not both occur in the set of configuration objects. For example, the definition of a_{capacity} does not allow two 40 Megabyte harddisks.

According to our definition, a configuration must specify 1. which objects are parts of the system to be configured, and 2. the entire functionality of the system.

Therefore, a *configuration* is a pair $C = \langle I, Q \rangle$, where I is a set of *items* of the form (k, o) and Q is a set of *qualities* of the form (f, x) . An item (k, o) means that object $o \in O$ is used k times in the configured system. A quality (f, x) means that the configured system has the functionality f with value x .

Although qualities and properties are syntactically equal, we distinguish between them since a property is the feature of an object, while a quality (f, x) is the result of the composition of several objects having the functionality f in their property sets.

Based on the above definition of composition, we are now ready to formally introduce the notion *configuration*.

Definition: Let $\Pi = \langle O, F, V, P, A, T, D \rangle$ be a configuration problem. A *configuration* C is inductively defined as follows:

1. $C = \langle \emptyset, \emptyset \rangle$ is a configuration.
2. If $C = \langle I, Q \rangle$ is a configuration and o is an object of O , then $C' = \langle I', Q' \rangle$ is a configuration if the following conditions hold:
 - (i) For every $(f, x) \in p_o$ and for every $(g, y) \in Q$, the composition $\varphi((f, x), (g, y))$ is defined or $Q = \emptyset$.
 - (ii)

$$I' = \begin{cases} I \setminus \{(k, o)\} \cup \{(k+1, o)\}, & \exists (k, o) \in I; \\ I \cup \{(1, o)\}, & \text{otherwise.} \end{cases}$$

(iii)

$$Q' = \begin{cases} \varphi((f, x), (g, y)), & Q \neq \emptyset; \\ p_o, & Q = \emptyset. \end{cases}$$

3. Nothing else is a configuration.

Remarks: Condition (i) guarantees that only those objects $o \in O$ are added to a given configuration C if all object’s properties p_o can be combined with all qualities of C . Condition (ii) specifies how one new object can be added to a given set of items I . Condition (iii) specifies how a new set of qualities can be constructed if a new object is added to the configuration.

Next we give a precise definition of the notion *solution* of a configuration problem.

Definition: A configuration $C = \langle I, Q \rangle$ is a *solution* of a configuration problem $\Pi = \langle O, F, V, P, A, T, D \rangle$ if and only if for each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \text{TRUE}$. The set $S(\Pi) = \{C \mid C \text{ is a solution of } \Pi\}$ is called the *solution space* of Π .

Remarks: The above condition guarantees that all demands are fulfilled. Generally there exists more than one solution of a configuration problem Π . Sometimes $S(\Pi)$ is called the “space of variants.”

General configuration problems

Based on the above definitions, the following problems may be stated:

Problem CONF

Given: A configuration problem Π .

Question: Does there exist a solution of Π ?

Problem FINDCONF

Given: A configuration problem Π .

Task: Find a solution of Π , if one exists.

Problem COSTCONF

Given: A configuration problem Π , a cost function $c : O \rightarrow \mathbf{Q}$, and maximum cost $c^* \in \mathbf{Q}$.

Question: Does there exist a solution $C = \langle I, Q \rangle$ of Π such that $\sum_{(k,o) \in I} k c(o) \leq c^*$?

Note that each of the above problems is essentially a combinatorial problem.

3 Model M2

We will now extend our model M1 in that way that we introduce *rules*. These rules may be interpreted as installation restrictions. For example, one would like to formulate the rule “If harddisk A is used, then either controller B or controller C must be used.”

Definition: 1. Let O be a set of objects and $N = \{1, \dots, k\}$. Let $\Gamma(N, O) = \{[n, o] \mid n \in N, o \in O\}$ denote the set of Boolean variables over N and O . A *configuration restriction rule* r is an implication $[n, o] \rightarrow \psi$, where $[n, o] \in \Gamma(N, O)$ and ψ is a logical formula over $\Gamma(N, O)$ using parentheses, ‘ \neg ’, ‘ \wedge ’, and ‘ \vee ’ in the standard way. A *rule set* R is a finite set of configuration restriction rules over $\Gamma(N, O)$.

2. Let O be given as in 1. Let $C = \langle I, Q \rangle$ be a configuration, where $I \subseteq N \times O$. A *configuration assignment* α_I is a function $\alpha_I : \Gamma(N, O) \rightarrow \{\text{TRUE}, \text{FALSE}\}$ such that for every $[n, o] \in \Gamma(N, O)$:

$$\alpha_I([n, o]) = \begin{cases} \text{TRUE}, & \text{if } (n, o) \in I; \\ \text{FALSE}, & \text{otherwise.} \end{cases}$$

3. A configuration $C = \langle I, Q \rangle$ is called *satisfying for a rule set* R if and only if every rule $r \in R$ is true under the assignment α_I using the known semantics of propositional logic.

Remarks: The semantics of a restriction rule is perhaps best explained by the following example: Let $r = [1, A] \rightarrow ([2, B] \wedge \neg[1, C]) \vee [3, D]$. The meaning of r is: “If a configuration contains exactly one object A , then the configuration must contain either two B ’s and not one C or three D ’s.”

Definition: A configuration problem under model M2 is a tuple $\Pi_R = \langle O, F, V, P, A, T, D, N, R \rangle$ where all elements but N and R are defined as in model M1, and R is a set of configuration restriction rules over $\Gamma(N, O)$. A configuration $C = \langle I, Q \rangle$ is a solution of Π_R if and only if for every demand $(f, x) \in D$ there exists a quality $(g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \text{TRUE}$ and C is satisfying for the rule set R .

The above problems CONF, etc. can be formulated in a similar way for model M2.

4 Is model M2 more powerful than model M1?

Model M1 is suitable to model a *resource-oriented* configuration problem, while model M2 is suitable to model a *skeleton-oriented* configuration problem. Many systems are built upon the skeleton model. If Π is a problem under model M2, then the skeleton of the configured system can be derived from the rules of Π . The skeleton is the digraph $G = (V, E)$ with $V = O$ and $(o_i, o_j) \in E$ if there is a rule which contains o_i in its left-hand side and o_j in its right-hand side. In typical applications, the digraph is a tree, and a configuration problem is then solved by a top-down strategy.

The model M2, which additionally contains a mechanism to express structural knowledge, seems to be more powerful than the pure resource-oriented model. However, as the following results shows, this is not the case. This is quite surprising since one expects that the rule language enables us to formulate more sophisticated configuration problems.

A central theorem of this work is the following.

Theorem A: Let Π be any instance of problem CONF under model M1 (M2). Then there exists an equivalent instance Π' of problem CONF under model M2 (M1) which can be obtained in polynomial time in the size of Π .

Corollary: Theorem A is also valid for the problems FINDCONF and COSTCONF.

We only prove the theorem, the corollary then follows immediately from the proof. (Readers not interested in the proof may continue with the example of the next section.)

Proof: Part I: Let $\Pi = \langle O, F, V, P, A, T, D \rangle$ be an instance of problem CONF under model M1. Trivially, let $R := \emptyset$, $N := \{1\}$ and let $\Pi' := \langle O, F, V, P, A, T, D, N, R \rangle$.

Part II: Let $\Pi = \langle O, F, V, P, A, T, D, N, R \rangle$ with $N = \{1, \dots, k\}$ be an instance of problem CONF under model M2. We have to show that there exists an instance Π' of problem CONF under model M1 such that Π has a solution if and only if Π' has a solution.

We construct Π' as follows. The basic idea of the proof is that the rules are replaced by new functionalities and new tests whose behavior is equivalent to these rules.

First, R is transformed into a logically equivalent set \tilde{R} which contains only 3CNF formulas (i.e., propositional formulas in conjunctive normal form where each clause has at most 3 literals).

This transformation is performed in two steps. First, a transformation technique due to Tseitin [1983] is used to transform each rule into a logically equivalent propositional formula in conjunctive normal form. This step requires the introduction of new variables, $\bar{\Gamma} = \{[1, \bar{o}_1], \dots, [1, \bar{o}_T]\}$. Second, every formula obtained from step one is transformed into an equivalent 3CNF formula by introducing the new variables $\hat{\Gamma} = \{[1, \hat{o}_1], \dots, [1, \hat{o}_S]\}$. Note that both transformations can be made in quadratic time and linear space. Let $\Gamma = \Gamma(N, O) \cup \bar{\Gamma} \cup \hat{\Gamma}$.

Thus, every rule $r \in R$ is transformed into a set $3\text{CNF}(r) = \{r_1, \dots, r_S\}$ such that r is satisfiable if and only if every formula $r_i \in 3\text{CNF}(r)$ is satisfiable. Let $\tilde{R} = \bigcup_{r \in R} 3\text{CNF}(r)$.

The introduction of the new variables implies that the new object set O' is defined as $O' := O \cup \bar{O} \cup \hat{O}$, with $\bar{O} = \{\bar{o}_1, \dots, \bar{o}_T\}$ and $\hat{O} = \{\hat{o}_1, \dots, \hat{o}_S\}$.

1. For every $r \in \tilde{R}$ we construct a new functionality g_r whose values are sets(!). For each $o \in O'$ which occurs in a rule $r \in \tilde{R}$, we define the property set of o with respect to g_r as $g_r(o) := \{(1, o)\}$.

2. We define the following “union” of an item set X and a singleton $\{(1, o)\}$, $o \in O'$, as follows:

$$X \uplus \{(1, o)\} = \begin{cases} X \setminus \{(n, o)\} \cup \{(n+1, o)\}, & \text{if } o \in O \text{ and } o \text{ occurs in } X \text{ and } n \leq k; \\ X \setminus \{(n, o)\} \cup \{(k+1, o)\}, & \text{if } o \in O \text{ and } o \text{ occurs in } X \text{ and } n > k; \\ X, & \text{if } o \in \bar{O} \cup \hat{O} \text{ and } o \text{ occurs in } X; \\ X \cup \{(1, o)\}, & \text{otherwise.} \end{cases}$$

This “union” function will subsequently be used to construct the value sets of the new functionalities.

3. The value set v_{g_r} is inductively defined as follows. Note that Δ is a help variable.

- (i) If o occurs in r , then $g_r(o) \in \Delta$.
- (ii) If $X \in \Delta$ and o occurs in r , then $X \uplus \{(1, o)\} \in \Delta$.
- (iii) Nothing else is in Δ .
- (iv) $v_{g_r} := \Delta \cup \{r\}$.

Note that v_{g_r} contains both sets of number/object pairs and the rule r itself. Also note that the computation of v_{g_r} can be made in a finite number of steps since $k+1$ bounds the number n that can occur in a pair (n, o) which itself occurs in a set $X \in v_{g_r}$.

4. As a demand d_r for r we define $d_r := (g_r, r)$.

5. For g_r we define the test t_{g_r} as follows, where $t_{g_r}(X, Y)$ is only defined for $X = r$ and $Y \in v_{g_r} \setminus \{r\}$:

$$t_{g_r}(r, Y) = \begin{cases} \text{TRUE,} & \text{if } r \text{ is true under } \alpha_Y; \\ \text{FALSE,} & \text{otherwise;} \end{cases}$$

where α_Y is restricted to the variable set $\Gamma_r = \{[n, o] \mid n \leq k+1, \text{ and } o \text{ occurs in } r\}$. Note that other variables than those in Γ_r cannot occur because $k+1$ bounds the number n .

6. Next we define the addition operator a_{g_r} for functionality g_r as follows: $a_{g_r}(X, Y)$ is only defined for $X \in v_{g_r} \setminus \{r\}$ and $Y \in \{g_r(o) \mid o \in r\}$: $a_{g_r}(X, Y) := X \uplus Y$.

7. Henceforth, let $\rho(o) = \{(g_r, g_r(o)) \mid r \in \{r \in \tilde{R} \mid o \text{ occurs in } r\}\}$

8. The elements of $\Pi' := \langle O', F', V', P', A', T', D' \rangle$ are now defined as follows:

$$\begin{aligned} O' &:= O \cup \bar{O} \cup \hat{O}, \\ F' &:= F \cup F_R, \text{ where } F_R := \{g_r \mid r \in \tilde{R}\}, \\ V' &:= V \cup V_R, \text{ where } V_R := \{v_{g_r} \mid r \in \tilde{R}\}, \\ P' &:= \{p_o \cup \rho(o) \mid o \in O\} \cup \{\rho(o) \mid o \in \bar{O} \cup \hat{O}\} \\ A' &:= A \cup A_R, \text{ where } A_R := \{a_{g_r} \mid r \in \tilde{R}\}, \\ T' &:= T \cup T_R, \text{ where } T_R := \{t_{g_r} \mid r \in \tilde{R}\}, \\ D' &:= D \cup D_R, \text{ where } D_R := \{(g_r, r) \mid r \in \tilde{R}\}. \end{aligned}$$

Case A. We have to show: If $C = \langle I, Q \rangle$ is a solution of Π , then there exists a solution $C' = \langle I', Q' \rangle$ of Π' . We show that there exist an item set ΔI and a quality set ΔQ such

that $I' = I \cup \Delta I$, $Q' = Q \cup \Delta Q$, and $C' = \langle I \cup \Delta I, Q \cup \Delta Q \rangle$ is a solution of Π' . Due to this construction of I' and since $D' = D \cup D_R$, we need only consider the “difference” demands D_R . (The original demands D are satisfied by I .)

We have to construct a ΔI in such a way that I' induces a quality set ΔQ with the following characteristic: For each $d = (g_r, r) \in D_R$ there exists a property $(g_r, Y) \in \Delta Q$ with $t_{g_r}(r, Y) = \text{TRUE}$.

Let $d = (g_r, r)$ be any demand of D_R where $r \in \tilde{R}$ is a 3CNF rule of the form $r = l_1 \vee l_2 \vee l_3$ with $l_i \in \{[n, o] \mid [n, o] \in \Gamma'\} \cup \{\neg[n, o] \mid [n, o] \in \Gamma'\}$. Note that r is satisfied if some l_i is satisfied. Since C is a solution of Π , it follows that all rules $r \in R$ are satisfied. Hence, all 3CNF rules in \tilde{R} are satisfiable by some truth assignment $\alpha_{I'}$. Note that $\alpha_I \subseteq \alpha_{I'}$; this guarantees that an object $o \in O$ occurs with frequency n in I if and only if object o occurs with frequency n in I' . Without loss of generality, we can assume that $l_1 = [n_1, o_1]$ is satisfied under $\alpha_{I'}$.

Case A1: Let $o_1 \in O$. If $l_1 = [n_1, o_1]$ then $\alpha_{I'}([n_1, o_1]) = \text{TRUE}$, hence (n_1, o_1) must occur in I . The definition of a_{g_r} (cf. the “ \oplus -operator”) guarantees that a $Y \in v_{g_r}$ with $(n_1, o_1) \in Y$ is inevitably constructed as the quality value of g_r . If $l_1 = \neg[n_1, o_1]$ then $\alpha_{I'}([n_1, o_1]) = \text{FALSE}$ (hence $(n_1, o_1) \notin I$). Now, either $(m_1, o_1) \in I$ with $m_1 < n_1$ or $m_1 > n_1$, then $(m_1, o_1) \in Y$, or o_1 does not occur in I at all and $(m_1, o_1) \notin Y$. As before, an appropriate $Y \in v_{g_r}$ is constructed as the quality value of g_r .

Case A2: Let $o_1 \in \bar{O} \cup \hat{O}$. If $l_1 = [n_1, o_1]$ then $\alpha_{I'}([1, o_1]) = \text{TRUE}$ and we put $(1, o_1)$ in ΔI . The quality value Y of g_r will contain $(1, o_1)$. If $l_1 = \neg[1, o_1]$ then $\alpha_{I'}([1, o_1]) = \text{FALSE}$ and o_1 is not allowed to be in ΔI . Hence, the quality value Y of g_r cannot contain $(1, o_1)$. So, ΔI is defined as the collection of all tuples $(1, o_1)$ found in case A2. Furthermore, let the g_r (with $r \in \tilde{R}$) and their corresponding Y form the set ΔQ . As seen above, with these definitions of ΔQ and ΔI it is guaranteed that for each $d = (g_r, r) \in D_R$ there exists the quality $(d_r, Y) \in \Delta Q$ such that $t_{g_r}(r, Y) = \text{TRUE}$.

Case B. We have to show: If $C' = \langle I', Q' \rangle$ is a solution of Π' , then there exists a solution $C = \langle I, Q \rangle$ of Π . Since $C' = \langle I', Q' \rangle$ is a solution of Π' , all demands in $D' = D \cup D_R$ are satisfied. Let $I = \{[n, o] \mid o \text{ occurs in } O\}$ and let $\Delta Q = \{(f, x) \in Q \mid f \in p_o, o \in \bar{O} \cup \hat{O}\}$.

1. Clearly, $C = \langle I, Q' \setminus \Delta Q \rangle$ satisfies all demands $d \in D$ since objects which occur in $I' \setminus I$ have no properties for which a demand $d \in D$ exists.

2. To see that $C = \langle I, Q' \setminus \Delta Q \rangle$ satisfies all rules $r \in R$, one need only consider the above transformation which guarantees that α_I is a satisfying truth assignment since $\alpha_I \subseteq \alpha_{I'}$ and $R \iff \tilde{R}$. \diamond

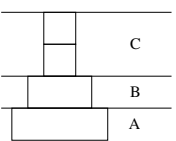
5 Skeleton-oriented configuration

In this section we give an example of a skeleton-oriented configuration problem formulated under model M2. Although a configuration problem and its solution strongly depend on particular aspects of the application, this example gives an idea how a hierarchical organized configuration problem can be described.

One characteristic of the skeleton-oriented configuration is that the solution space can be described by a hierarchical graph with two kinds of nodes: AND-nodes and OR-nodes (cf. Puppe [1990]). An AND-node indicates that each direct successor of this node must be selected in the configuration process (more general: to solve the whole problem, each subproblem has to be solved); an OR-node describes mutually exclusive alternatives. The skeleton-oriented configuration approach is appropriate, if we want to configure a system which has always the same basic structure.

In the following example, the task is to configure a tower which has always three planes: An A-plane, a B-plane and a C-plane. For each plane there exists a particular kind of building blocks (A-blocks, B-blocks and C-blocks). Furthermore, the building blocks of the tower have to fulfill the following restrictions: For both plane A and plane B exactly one block of the appropriate kind must be selected. Plane C has to be constructed with at least one C-block where C3 cannot be combined with any of the other C-blocks. If block C2 is used once, block B1 is not allowed to occur once in a configuration. The goal is to build a tower with a given height and minimum cost. The following figure describes the building blocks which can be used to construct a tower.

Block	Height	Cost
A1	1	2
A2	2	4
B1	1	3
B2	3	5
C1	1	2
C2	2	3
C3	4	6



To describe this problem as a hierarchical configuration problem, we introduce particular “dummy blocks” S, A, B, C which have no properties. With the new building blocks, the configuration restrictions are described by the following rules:

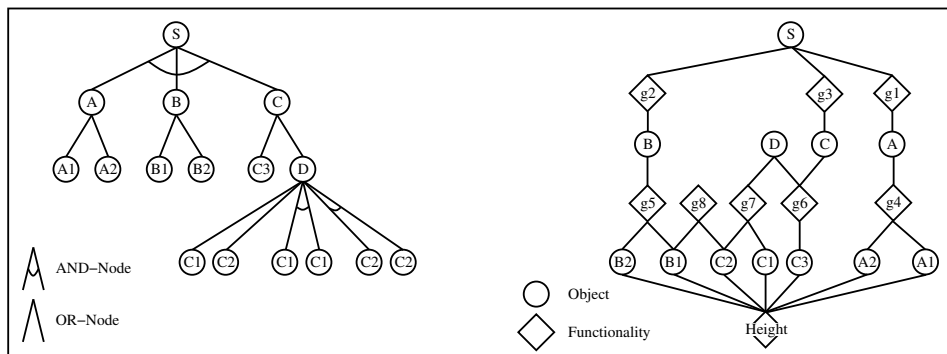
$$[1,S] \rightarrow [1,A] \wedge [1,B] \wedge [1,C], [1,A] \rightarrow [1,A1] \vee [1,A2], [1,B] \rightarrow [1,B1] \vee [1,B2], [1,C] \rightarrow [1,D] \vee [1,C3], [1,D] \rightarrow [1,C1] \vee [2,C1] \vee [1,C2] \vee [2,C2], [1,C2] \rightarrow \neg [1,B1]$$

The set of objects is $O = \{S,A,B,C,D,A1,A2,B1,B2,C1,C2,C3\}$, and the set of functionalities is $F = \{\text{height}\}$

With the specification of the transformation of model M2 into model M1 (see Section 4), we are now able to reformulate this configuration problem as a problem which solely bases on functionalities and their computation. We do not want to perform this transformation explicitly since we gave examples for single transformation steps in Section 4.

In the reformulated problem, the dependencies between the objects must be derived from their properties. There is an edge between those objects which share at least one functionality.

The following figure shows the original configuration problem and its concrete reformulation. It illustrates in which way the transformation of M1 into M2 comes to effect. The transformation of the above rules yields eight new functionalities g_1, \dots, g_8 . In the picture, both functionalities and configuration objects are vertices; an edge (o_i, g_j) indicates that the object o_i has the functionality g_j in its property set.



6 A complexity consideration

We present in this section a result regarding the computational complexity of CONF. Note that all other problems are at least as hard as CONF.

Theorem B: Problem CONF is NP-complete.

Lemma C: Problem CONF with restriction rules (CONF_R) is NP-complete.

The theorem and the lemma are proven in [Najmann & Stein 1992]. The basic idea of the proof is to transform 3SAT (which is NP-complete, [Cook, 1971]) to CONF_R and to apply the equivalence results of section 4.

Summary: We have developed two models of configuration, M1 and M2. Model M1 allows to formulate typical resource-oriented configuration problems, while model M2 allows to formulate skeleton-oriented configuration problems. Model M2 was obtained by augmenting model M1 by a restriction rule language. Although model M2 seems to be more powerful than M1, it was shown that both models can be transformed into each other in polynomial time. That means that there is principally no difference between the classical configuration approach and the resource-oriented approach. Under both models, we have formulated a number of typical configuration problems like the problem of finding a cost-minimal configuration. We have also noted that the fundamental problem, namely to decide whether there exists a configuration for a given problem specification, is NP-complete.

References

- Brown, D. C.; Chandrasekaran, B. [1989] *Design Problem Solving*, Pitman.
- Cook, S. [1971] "The complexity of theorem-proving procedures", *Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery*, New York, pp. 151–158
- Cunis et al. [1991] *Das PLAKON-Buch – Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen*, Springer Verlag, Berlin.
- Doerner, H. [1991] "Ein Modell des Konfigurierens" *Beiträge zum 5. Workshop "Planen und Konfigurieren"*, LKI-M-1/91
- Heinrich, M. [1991] "Ressourcenorientierte Modellierung als Basis modularer technischer Systeme," *Beiträge zum 5. Workshop "Planen und Konfigurieren"*, LKI-M-1/91
- Najmann, O., Stein, B. [1992] "Modeling resource-oriented configuration problems." *Internal report, University of Paderborn*.
- Puppe, F. [1990] *Problemlösungsmethoden in Expertensystemen*, Springer Verlag.
- Stein, B.; Weiner, J. [1990] *MOKON*, Interner Report, Universität-Gesamthochschule-Duisburg, SM-DU-178
- Tank et al. [1989] "AMOR: Eine Beschreibungssprache für technische Systeme zur Unterstützung der Wissensakquisition für Konfigurationsprobleme", *Beiträge zum 3. Workshop "Planen und Konfigurieren"*, Arbeitspapiere der GMD 388
- Tseitin, G. [1983] "On the complexity of derivations in propositional calculus", *Automation of Reasoning 2: Classical Papers on Computational Logic*, pp. 466-483. Springer Verlag, Berlin.

*A reprint of this paper can be found in the Report Series LKI-M of the Laboratory for Artificial Intelligence, Hamburg, number 93/3, pp. 41-47, September, 1993.