Integrating Time Constraints into Constraint-Based Configuration Models

Ulrich John[1] and Ulrich Geske[2]

[1] DaimlerChrysler AG, Research Information & Communication, email: Ulrich.John@daimlerchrysler.com
[2] Fraunhofer FIRST, email: Ulrich.Geske@first.fhg.de

Abstract. Over the last few years, we have been developing the configuration model ConBaCon, which is based on Constraint Programming over finite domains. The model is sound and suitable for building efficient and flexible systems that fulfill all the requirements of advanced configuration systems.
In this paper, we present model extensions that enable time-extended configuration and reconfiguration problems to be solved: Besides "normal" configuration and reconfiguration problems, the extended model can now also solve problems that contain time-dependent resource avaibilities or supply constraints for the ground components included. The general nature of the model extensions introduced, would seen to make them suitable for use other (commercial) constraint-based configuration systems/configurator libraries as well.

1 Introduction

Configuration problems can be found in a huge number of business fields. That is why the development of efficient and flexible configuration systems is still a hot topic in computer science. In the last twenty years, with the introduction of the well-known, rule-based configuration system XCON for configuring DEC computers, different approaches have been proposed and investigated for the knowledge-based configuration of products and technical systems. These include various rule-based, case-based and, recently, more and more constraint-based approaches. Overviews of different approaches and systems are given in [19], [18] and [12]. When both research approaches and commercial systems were considered in the past, the following general shortcomings were often found. The problem specification was nondeclarative and hard to maintain. Often, the sequence of interactions during the configuration process were fixed, making a flexible configuration process such as that supported by *ConBaCon* impossible. The simulation of different effects resulting from alternative, interactive decisions is rare and the support of good reconfigurations, which is needed by industry, is inadequate or nonexistent. Furthermore, finding optimal or near-optimal configurations is impossible, and there are other problems like the occasional failure of the underlying algorithms fail to terminate.

It is generally accepted that high-quality configuration systems can be realized, especially by using *constraint programming* (cf. [8], [2], [17]).

Our configuration system ConBaCon, based on the CLP language CHIP, overcomes the above shortcomings/problems. The ConBaCon model behind the system is theoretically sound and covers – together with several model extensions – a broad range of technical and non-technical configuration problems (cf. [12]).

A rather interesting problem class is that of *time-constrained configuration problems*. Such problems contain, besides the "normal" problem elements, time-constrained resource availabilities and possible supply times for potential result components.

The practical importance of these problems is highlighted by the following example: Often we have a situation where a customer wants to buy a complex product, for instance a car. Besides some constraints relating to the physical configuration of the car like color and equipment, and some optimizing goals like price and consumption, the customer wants a wants the guarantee of delivery by a certain self-appointed date.

Although several software and research companies are working on approaches that tackle these problems, there are no configuration systems available that are able to handle this problem satisfactorily, nor are the authors familiar with any publications that describe adequate solutions. At best, some approaches offer the opportunity to check the earliest delivery date after completion of the configuration process proper.

We have developed some extensions of our constraint-based configuration model that allow highly flexible, efficient configuration and reconfiguration processes driven by the fixed delivery date. After giving the delivery date as a hard constraint, the user can explore the possibilities for his/ her car in a flexible, interactive way. All options that cannot be chosen because of the fixed delivery date are marked automatically throughout the configuration process. This enables the user to configure his/ her "dream car" subject to the restriction of the self-appointed delivery date.

The following section outlines the specification language ConBaConL according to our initial configuration model. Section 3 introduces some key aspects of our configuration model and its realizationConBaCon, which allow the configuration of industrial products/technical systems. The model extensions for solving time-constrained configuration and reconfiguration problems are presented in Section 4, where we also give an example for illustrative purposees. The paper closes with a conclusion and some remarks on possible future extensions.

2 ConBaConL

By analyzing the results of design problems for industrial control systems, we developed a formal problem model and, based on this, the largely declarative specification language *ConBaConL*, which allows the specification of relevant configuration problems. Such specifications are composed of three parts: *object*

hierarchy, *context-independent constraints* and *context constraints*. Every technical object that can play a part in the configuration problem must be specified in terms of its structure in the object hierarchy. An object can consist of several components in the sense of the $consist\_of$-relation, where components may be optional, or the object has some specializations in the sense of the is_a-relation. In addition, all attributes of the technical objects are specified. If the attribute values of a technical object are explicitly known, they are enumerated.

A correct context-independent representation of the configuration problem is created from the object-hierarchy specification by adding the specification of the constraints concerning different attribute value sets on the one hand, and the existence or nonexistence of technical objects in the problem solution on the other. If context constraints exist (e.g. customer-specific demands or resource-oriented constraints), we have to specify them as problem-specific constraints in ConBaConL. The distinction between problem-specific and context-independent constraints is useful because the technical correctness of the problem solution is ensured if all context-independent constraints are fulfilled.

The constraint elements of ConBaConL can be divided into *Simple Constraints*, *Compositional Constraints* and *Conditional Constraints*. Most of them are introduced below.

2.1 Simple Constraints

Attribute Value Constraints and Existence Constraints

[o, Attr, VS]/not([o, Attr, VS]) $\equiv$
the attribute *Attr* of object *o* must/must not take a value from VS,

exist(Objectlist)/noexist(Objectlist) $\equiv$
all objects contained in *Objectlist* must/must not be part of the solution.

Relational Constraints Between Attribute Value Sets & Table Constraints

eq(T1,T2), neq(T1,T2), lt(T1,T2), let(T1,T2), gt(T1,T2), get(T1,T2). Furthermore, it is possible to specify equations over attributes.

In practice, coherences between solution parts are often specified in the form of tables (decision tables). To avoid a manual, error-prone translation of the table into other kinds of ConBaConL constraints, a table constraint (see [10]) was introduced.

2.2 Compositional Constraints

Compositional Constraints are, besides the above-mentioned Simple Constraints, compositions of compositional constraints: *and*($[Cons_1, \ldots, Cons_n]$), *or*($[Cons_1, \ldots, Cons_n]$), *xor*($[Cons_1, \ldots, Cons_n]$), *at_least*($[Cons_1, \ldots, Cons_n]$,*N*)/ *at_most*($[Cons_1, \ldots, Cons_n]$,*N*)/ *exact*($[Cons_1, \ldots, Cons_n]$,*N*) $\equiv$ at_least/at_most/exactly N of the listed constraints are valid[1].

[1] So far, the processing of or-, xor, at_least, at_most, exact-constraints concerning the existence and nonexistence demands of objects has been realized in ConBaCon.

2.3 Conditional Constraints

[if(Comp_Cons$_1$), then(Comp_Cons$_2$)] (*[iff(Comp_Cons$_1$), then(Comp_Cons$_2$)]*)
If (and only if) the compositional constraint $Comp\_Cons_1$ is fulfilled, the compositional constraint $Comp\_Cons_2$ must be fulfilled.

2.4 Preferences

There are two ways of describing and processing preferences. One is to try encoding the existing preferences as preference rules in the labeling heuristics (see below) within the problem-solution model. Another is to specify weak constraints. So far, specifying *weak simple constraints* has been supported (cf. [10]).

2.5 Modifications

In practice, the specification of product taxonomies/component catalogues has high modification rates. For instance, about 40% of the 30,000 component types of DEC computers used in R1/XCON were updated annually (cf. [4]). There are several reasons for the need to change the product taxonomies/problem specifications. The most common is the fact that new technical modules become available or obsolete ones are withdrawn. A special case of this is so-called versioning. Another reason is the changing of context-independent constraints due, for instance, to changes in laws or government policy. In order to allow subsequent reconfigurations, obsolete information should not be deleted in the specification. Instead, obsolete modules and constraints should be labeled with the keyword *obsolete* in the specification/product taxonomy. New modules and constraints can easily be added to the specifications/product taxonomies. There are two cases in which a new module/object *new_o* is integrated as an alternative to an already specified module o_x. In the first case, o_x is a specialization of an object o. In the second case, o_x is a module of o, whereby a notional object must be introduced at the position of o_x, which acquires the specializations *new_o* and o_x.

Other important elements of ConBaConL are *optimization goals* such as minimization or maximization of indicated attribute values.

A typical specification of ground converters for large electric motors is outlined in [10], together with the problem solution using ConBaCon.

3 Problem Solution Model

When transforming a problem specification, our goal is to obtain a problem-solution model that allows efficient problem solution. The model should also support the option of high-quality user interactions. The model of a constraint-logic system over finite domains is taken as a basis for the solution model outlined below. Thus, the model can also be seen as a global constraint for structural configuration.

3.1 Objects

Each specified object (representing a technical module) that is not marked as obsolete is transformed into a *module object* of the problem-solution model[2]. Moreover, each attribute of a specified object is transformed into an *attribute object*, i.e. a specified object with n attributes is represented by $n+1$ objects in the problem-solution model (Fig. 1).

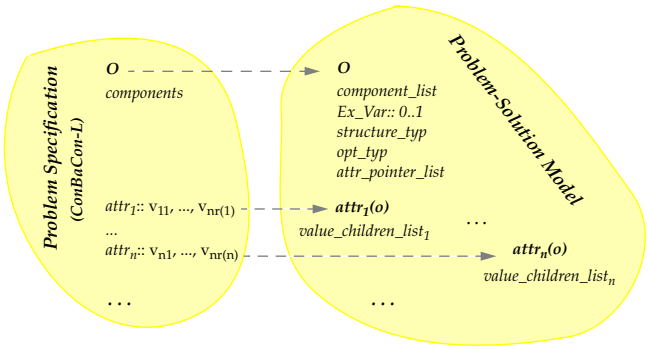

Fig. 1. Transformation of Objects

Objects of the problem-solution space acquire certain model-specific attributes. The attribute *component_list* of object o contains identifiers of the object components (*structure_type* = and-node) and of the specializations (*structure_type* = or-node) of o, respectively. The constraint variable Ex_Var determines whether or not the object is contained in the solution. If the value of Ex_Var is zero, o is not part of the solution. If the value is one, o is part of the solution. *opt_type* contains information about whether o is optional or not. Links to the corresponding attribute objects are given by *attr_pointer_list*. Each attribute object stores possible attribute values in *value_children_lists* and in the domain of a corresponding constraint variable. Moreover, identifiers of the value-related children nodes are stored if the object o contains specializations. In this case, the attribute value sets of o are the set unification of the corresponding attribute value sets of the specialization objects.

Besides the model objects, constraints are needed in the problem-solution model to ensure the coherences between the objects of the model so that the correctness of the solution and the completeness of the solution process are guaranteed with respect to the problem specification. These constraints we call *consistency-ensuring constraints* (CE constraints).

[2] Some constellations require the introduction of auxiliary module objects. These are not considered in the present paper. Details can be found in [12].

3.2 CE Constraints

Consistency-Ensuring Constraints are realized as logical coherences between values of Ex_Var-attributes/attribute value sets of different attribute objects. The most important CE constraints are schematized in Fig. 2. If it becomes obvious

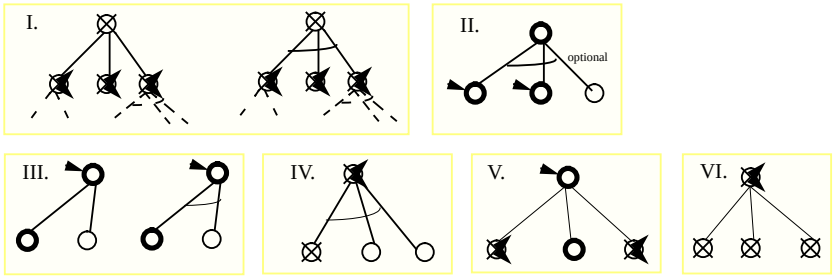

Fig. 2. Consistency-Ensuring Constraints

that an object cannot occur in the solution, it must be inferred that no component/specialization of it can occur in the solution (I). If it becomes obvious that an object is part of the solution ($Ex\_Var = 1$), it must be ensured that all nonoptional components of the object are part of the solution, too (II). The existence of an object in a solution implies in each case the existence of its parent object (III). Furthermore, if a nonoptional component of an object o cannot occur in any solution, the parent object o cannot occur in any solution either (IV). If the specialization of an object o is part of the solution, no other specialization of o can be part of the solution (V). If it becomes obvious that no specializations of an object o can occur in any solution, it must be inferred that o cannot occur in the solution either (VI).

Attribute value sets are kept consistent by a special class of CE constraints. Where a value is deleted in the attribute value set of a specialization of an object o, the value must be deleted in the corresponding attribute value set of o, except if there is another specialization of o that contains the deleted value in the corresponding attribute value set[3]. If an attribute value is deleted in an attribute value set of an object o possessing specializations, the same value must be deleted in all corresponding attribute value sets of the specializations of o. If an attribute value set of an object o becomes empty, the nonexistence of o will be inferred by a special CE constraint.

By integrating the introduced CE constraints in the problem-solution model, the structural coherences between objects of the solution model are ensured with respect to their existence, nonexistence and attribute value sets. Moreover, the constraints formulated in the problem specification must be transformed into constraints of the problem-solution model.

[3] To avoid intensive checking, the attribute *value_children_list* of the corresponding attribute object is checked and updated after each deletion of an attribute value.

3.3 Specified Constraints

Attribute value constraints and *existence constraints* result in the deletion of attribute values in the problem-solution model or in the setting of Ex_Var-attributes. *Relational constraints* between attribute value sets result in the deletion of attribute values, which become invalid because of the specified relation. If there are other value tuples that do not fulfill the relation, some appropriate daemons have to be generated which control the relational constraints after each alteration of the attribute value sets in question. *Table constraints* define connections between the attribute value sets in question and existence information (Ex_Var) on the objects listed in the table head. Altering the attribute value sets or existence values results in the marking as invalid of corresponding table lines. If all table lines are marked as invalid, the table constraint is not satisfied. Conversely, it is ensured that the attribute value sets in question contain only values that are registered in valid table lines[4]. *Compositional Constraints* are normally realized in the solution model by equations and inequations over corresponding Ex_Var-attributes. For each nonexistence statement of an object, the term "$1-Ex\_Var$" is used instead of Ex_Var in the equation/inequation. *Conditional Constraints* are transformed into conditional transitions of the problem-solution model, which ensure the specified logical coherences within the problem-solution model. In order to substantially reduce the problem space within the problem-solution model, the contrapositions of the specified conditional constraints are also transformed into elements of the problem-solution model.

3.4 Configuration Process

Based on the outlined problem-solution model, a flexible and efficient problem-solution process (Figure 3) was realized within the prototypical configuration system ConBaCon using the CLP language CHIP. In particular, the object-based data management and the existence of *Conditional Propagation Rules*[5] in CHIP facilitated the implementation.

The specified configuration problem is transformed into objects of the problem-solution model. This means that the objects of the solution model are generated, corresponding CE constraints are inferred and set, and the specified constraints are transformed into corresponding constraints of the problem-solution model. The value one is assigned to the Ex_Var-attribute of the target object because the target object must exist in each solution.

Thanks to the generated model with the model-specific CE constraints, a substantial reduction of the search space is guaranteed. In [12] mathematical sentences with their proofs are presented which allow explicit identification of whether a given configuration-problem specification will be transformed by the described procedure into a *strong k-consistency* solution model (backtrack-free solution process is ensured) or not. For specified problems that do not fulfill

[4] For implementation details of table constraints, see [12].

[5] Similar language elements exist in other CLP languages, e.g. Constraint-Handling Rules in ECLIPSE.

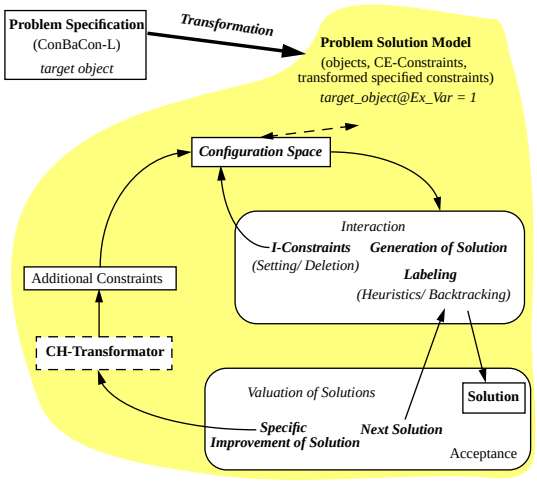

Fig. 3. Problem-Solution Process

the required properties, instructions for possible preprocessing steps and – as an alternative – for generating interaction-control procedures for the configuration process are given.

We call the set of the currently active module objects of the problem-solution model *Configuration Space*. Interactive user constraints now can be given (one by one) relating to the existence or nonexistence of objects of the configuration space or to the shape of the corresponding attribute value sets. The user's freedom to decide which object or attribute value set of the configuration space should be restricted by an interactively given user constraint is an outstanding feature compared with most other configuration models/tools. Governed by the constraints of the problem-solution model, this results in a new configuration space. Thus, a new cycle can start. Users can either give a new interactive constraint or they can delete previously given interactive user constraints. This allows the simulation of several user decisions, which is the prerequisite for a highly flexible configuration process. If no further interactive constraints are required, the generation of a solution can be started. This is done by labeling the Ex_Var-attributes of the (still) active objects of the problem-solution model. Such labeling can be controlled by heuristics. This allows us to take into account preferences in the form of preference rules for controlling the labeling process. If the solution found is not suitable or fails to pass the solution-quality check, further solutions can be created by backtracking. If a partial improvement of the solution suffices, a specific solution improvement can be started by specification and processing of a constraint hierarchy, i.e., the constraints that must be satisfied unconditionally are specified as *hard* constraints, and the solution parts that should, if possible, be in the new solution or desired attribute values are fixed as *weak* constraints. The weak constraints can be marked with several weights. The specified constraint hierarchy is processed in an error-minimization process, which results in the generation of a set of equivalent (hard) constraints

of the problem-solution model. Information about the realization and application of constraint hierarchies in ConBaCon for partial improvement can be found in [12].

At second sight, it becomes obvious that the improvement process using a constraint-hierarchy transformer provides a sound basis for reconfiguration, which is needed by industry. The reconfiguration approach using ConBaCon is described in [13], and in more detail in [12].

Besides model extensions for realizing reconfiguration processes with ConBaCon, we have developed a couple of other model extensions which extend the set of configuration problems that are manageable by our approach. Among them are extensions for tackling large configuration problems (mainly clustering of the model; see [11, 12]), for certain design problems and for optimization-oriented configuration problems (handling of large nets of arithmetic constraints; see [12]).

In the following section, we present the idea of how to integrate time constraints into the solution model in such a way as to enable the solution of time-constrained configuration problems of the sort described in the Introduction.

4 Integrating Time Constraints

Time-constrained configuration problems emerge from classical configuration problems by taking into account availability times of preliminary products, processing times, constraints between these time points/slices and resource constraints regarding the required processing operations. We can distinguish between assembly processes and transformation processes. *Transformation processes* can be connected with all problem objects represented by module objects in the problem-solution model (c.f. Section 3). *Assembly processes* are connected with all aggregated objects which are presented in the solution model by module objects with *structure_type = and-node* and their component elements. To be able to specify time-constrained configuration problems, we must of course add, some suitable specification primitives to the specification language ConBaConL. In the rest of this section we focus on necessary extensions of the problem solution model.

4.1 Availability Variables

For each module object *o* of the solution model, an availability FD-variable *avail(o)* must be introduced, which represents the availability of the module.

The domain of each availability variable must contain the value *zero* as a special element.

If a module is a ground object (supply component), the domain of the associated variable must contain, in addition to *zero*, all the time values at which the module is available. The domains of availability variables of nonground objects

(not supply objects) initially get a proper FD-interval, e.g. $\langle now, end\_of\_planning\_horizon \rangle$[6] in addition to the special domain element *zero*.

The inital domains of the availability variables are normally reduced after generating special constraints into the problem-solution model (see below). In order to get better propagation, it is advantageous to delete, in a preprocessing step, values from $\langle now, end\_of\_planning\_horizon \rangle$ that obviously cannot be valid because of the availabilities of the respective components (see next section).

Each processing step (or atomic chain of processing steps) in production can be explicitly associated with a module object. For each object o, we have to introduce a FD-variable $proctime(o)$.

To simplify the description, let us asume that the processing times are always *zero*. This restriction does not affect the quality of the presented model extensions because processing times greater than zero can be easily introduced into the model by proper additional addends (FD variables) in the model.

4.2 Availability Constraints

We must ensure the generation of proper constraints that specify the problem-dependent relations between the availability variables of a solution model. We call them *availability constraints*. In doing so, we must distinguish between aggregated objects (structure_type = and-node) and *or-objects* (structure_type = or-node).

Aggregated Objects For each aggregated object o with components $c_1, \ldots, c_n$, it is clear that $avail(o) = proctime(o) + max(avail(c_1) + proctime(c_1), \ldots, avail(c_n) + proctime(c_n))$ if we assume that the assembly process can start at the earliest when all components are available after associated transformation processes are finished. In our solution model, we can realize this relation using the global constraint *maximum(CV,CVList)* which ensures that CV is the maximum of the elements of $CVList$. The propagation realized by this constraint works in both directions: from CV to the elements of $CVList$ and vice versa. If the availability variable domain of an aggregated object is restricted top-down, the availability variable domains of all components will be restricted with the same limit. If "no real" availability time point remains, the variable will be instantiated with the remaining value *zero*.

or-Objects In the case of or-nodes o with specializations $s_1, \ldots, s_n$, we know that o can be available at earliest when one of its specializations s_i is available and the transformation operation possibly associated with s_i is finished. We can realize these dependencies between the availability variables of objects in an or-dependency using the procedure (meta constraint) $domain\_aeq\_cb(R, L)$ (see below), which was also used for model extensions relating to large nets of arithmetic constraints in specified configuration problems (for details, see [12]).

[6] *now* should be greater than *zero*.

$domain\_aeq\_cb(R, L) :-$
$\quad domain\_aeq\_cb\_up(R, L), !,$
$\quad touched(domain\_aeq\_cb\_down, R, L, all).$
$domain\_aeq\_cb\_up(R, L) :-$
$\quad domain\_aeq\_cb\_up\_t2(L, [], Nullist),$
$\quad length(L, Le), Z :: 1..Le,$
$\quad element(Z, L, Nullist, R, [all, all, all, all]).$
$domain\_aeq\_cb\_up\_t2([], Nullist, Nullist).$
$domain\_aeq\_cb\_up\_t2([H|T], A, Nullist) :-$
$\quad !, domain\_aeq\_cb\_up\_t2(T, [0|A], Nullist).$

For each specialization dependency of a specified configuration problem, the procedure $domain\_aeq\_cb(R, L)$ is called, R being the availability variable of the or-object and L the availability variable list of the specializations. The call of $touched(domain\_aeq\_cb\_down, R, L, all)$ ensures that values are deleted from the domains of the variables in L (exception: special value *zero*) if the values are deleted in the domain of R. The procedure $domain\_aeq\_cb\_up(R, L)$ ensures by calling an *element* constraint that values are deleted from the domain of R if they are removed from the domains of all availability variables listed in L. The *element* constraint is a global constraint. In short, $element(i, [e_1, \ldots, e_n], CV)$ ensures that $CV = e_i$. In our model, this means that R is, in each generated problem solution, equal to one availability variable of L. This gives us the guarantee that the propagation will be done in the desired quality. Thus, the procedure $domain\_aeq\_cb(R, L)$ is also a global constraint that ensures the consistency between availability value sets of an or-node and its specializations. By doing so, a complete propagation between the availability variables in or-dependencies is guaranteed.

Initial Reduction of Availability Variables After generation and activation of the described availability constraints, the initial availability-variable domains that belong to aggregated objects should be reduced before the configuration process is started. This should be done bottom-up with respect to the specified taxonomy. For each aggregated object o, the smallest value *limit* that is able to fulfill the *maximum* relation (see above) is calculated on the basis of the availability-variable domains of the components of o. All values from $\langle 1, limit \rangle$ are deleted in $avail(o)$. Of course, further domain reductions can follow, caused by propagation due to the availability constraints of the model during the initial reduction process.

Constraints between Availability and Existence It is intuitively clear that object elements of configuration problems can be part of a problem solution if and only if they are available in time. A remaining task is introducing transformations of these constraints into our problem-solution model.

We can do this by extending the set of CE Constraints (see Section 3.2) by constraints between availability variables and their associated existence vari-

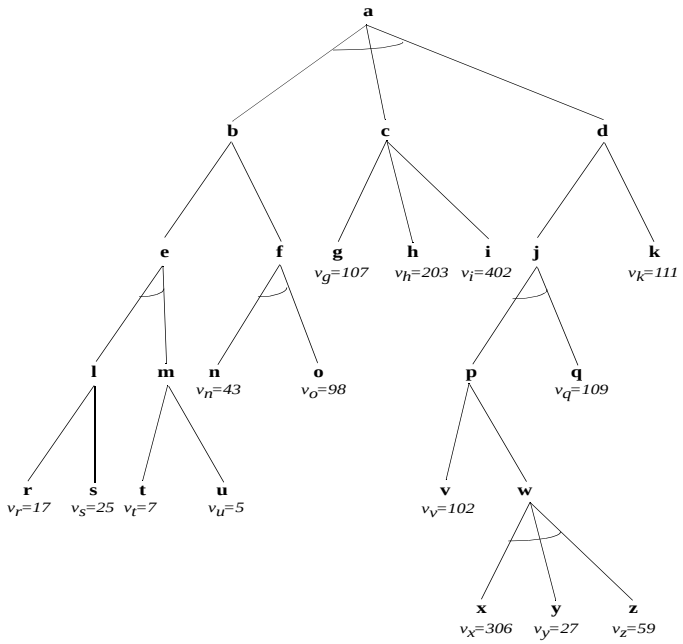

Fig. 4. Taxonomy of Product a with Given Availabilities

ables. To ensure the described dependencies, it is enough to generate, for each module object o, the following conditional constraints:

1. $Ex\_Var(o) = 0 \rightarrow avail(o) = 0$,
2. $avail(o) = 0 \rightarrow Ex\_Var(o) = 0$ and
3. $Ex\_Var(o) = 1 \rightarrow avail(o) > 0$.

4.3 Example of Time-Constrained Configuration

To understand better the way our configuration model works – and the extensions introduced in the previous sections – let us now consider the following example problem shown in Figure 4.

Given is the problem tof configuring the complex product a, which consists of the components b, c and d. The specification elements b, c and d are abstract elements. This means that b can be instantiated in the final product either with e or f, c with g, h or i, and d with j or k, and so on. The specified components $g, h, i, k, n, o, q, r, s, t, u, v, x, y$ and z are ground components (which may be delivered by a supplier). For each of them, the earliest possible delivery time is listed in the figure; v_s is, for instance, the earliest availability time for s.

For the sake simplicity, we assume that all operation durations are *zero*. In principle, however they could be greater than *zero*. On the other hand, resource constraints relating to availability could also be integrated into the problem specification.

Let the specified configuration problem be transformed into a corresponding (constraint-based) problem-solution model following the description given in Section 3. In addition to this transformation process, availability variables and availability constraints are generated as described in Sections 4.1 and 4.2.

After processing the initial reduction of the availability-variable domains of aggregated objects (see above) and following the domain reductions due to the constraints realized in the solution model, we obtain the following variable domains:

dom(avail(g)) = {0, 107}, dom(avail(h)) = {0, 203}, dom(avail(i)) = {0, 402}, dom(avail(k)) = {0, 111},

dom(avail(n)) = {0, 43}, dom(avail(o)) = {0, 98}, dom(avail(q)) = {0, 109},

dom(avail(r)) = {0, 17}, dom(avail(s)) = {0, 25}, dom(avail(t)) = {0, 7}, dom(avail(u)) = {0, 5}, dom(avail(v)) = {0, 102},

dom(avail(x)) = {0, 306}, dom(avail(y)) = {0, 27}, dom(avail(z)) = {0, 59},

dom(avail(l)) = {0, 17, 25}, dom(avail(m)) = {0, 5, 7}, dom(avail(p)) = {0, 102, 306},

dom(avail(e)) = {0, 17, 25}, dom(avail(f)) = {0, 98}, dom(avail(j)) = {0, 109, 306},

dom(avail(b)) = {0, 17, 25, 98}, dom(avail(c)) = {0, 107, 203, 402}, dom(avail(d)) = {0, 109, 111, 306},

dom(avail(a)) = {0, 109, 111, 203, 306, 402},

Now the main configuration process, which is described in Section 3.4 can be started. In addition to the "conventional" questions, which can be answered during the configuration process, we are now able to investigate time-relevant questions as well. For instance, the user can observe that the earliest delivery time for product a is 109.

Also, it is obviously possible to answer questions of the type mentioned in the Introduction using the problem solution model. For instance, system user are able to investigate, which configurations of a are available if they set the latest delivery time to 203. This demand is equivalent with the interactively given constraint $avail(a) \leq 203$. Because of the *maximum constraint* between $avail(a), avail(b), aivail(c)$ and $avail(d)$ (see Section 4.2), the value 402 is deleted from $dom(avail(c))$ and the value 306 is removed from the domain of $avail(d)$. This immediately implies the deletion of 306 in $dom(avail(j))$ due to the *domain_aeq_cb constraints* between $avail(c), avail(g), avail(h)$ and $avail(i)$ and between $avail(d), avail(j)$ and $avail(k)$. The deletion of 402 in the domain of $avail(i)$ results in the implication of the nonexistence of i because of the constraint $avail(i) = 0 \rightarrow Ex\_Var(i) = 0$ (c.f. Section 4.2). And so on.

As a result of the hard demand that the latest possible delivery time be 203, our solution model deduces that neither the component i nor the component x can be included in the final product.

The configuration process can now be continued in the familiar way (see Section 3.4). Of course, reconfiguration processes can also be started after the generation of a solution.

5 Conclusion

We have presented some fundamental information about our constraint-based problem-solution model ConBaCon for the configuration and reconfiguration of technical systems/industrial products. An idea of the complexity of the configuration problems that can be tackled by the solution model was given by describing the main elements of the corresponding specification language ConBaConL.

The problem-solution model – together with several extensions[7] – was realized using the CLP language CHIP. The resulting ConBaCon system was successfully used with several realistic and abstract configuration problems, including the configuration of power-supply systems for large electric motors and the configuration of computer rack systems.

By substantially reducing the search space, the problem-solution model – together with the underlying CLP system – allows an efficient configuration process that can be flexibly controlled by user interactions. It is ensured that each solution obtained is correct with respect to the problem specification and the underlying constraint solver. In addition, the completeness of the solution process is guaranteed.

The main focus of this paper was on novel model extensions that allow the flexible solving of time-constrained configuration problems. Compared with other configuration systems, this is an outstanding feature of the resulting problem-solution model. Given the rather general nature of the extensions, we assume that the key ideas presented here can also be integrated quite easily into other constraint-based configuration systems or configuration libraries like ILOG Configurator.

The new features of the problem-solution model were demonstrated by means of an example.

Our extended configuration model offers a broad range of interesting tasks for future work. For instance, the development of extensions for time-constrained multiproduct configurations as well as investigations on the tighter integration of scheduling systems into the problem-solution model are important areas of future research.

References

1. Axling, T.: Collaborative Interactive Design in Virtual Environments. www.sics.se/~axling/3dobelics.html (1996)
2. Axling, T., Haridi, S.: A Tool for Developing Interactive Configuration Applications. Logic Programming 26 (2) (1996) 147-168
3. Fleischanderl, G. et al.: Configuring Large Systems Using Generative Constraint Satisfaction. IEEE- Intelligent Systems 13 (4) (1998)
4. Freuder, E. C.: The Role of Configuration Knowledge in the Business Process. IEEE Intelligent Systems 13 (4) (1998)

[7] Links to the corresponding publications are given in the paper.

5. Geller, S.: Come, and they will build it. Manufacturing Systems (June 1999)
6. Gupta, L., Chionglo, J. F., Fox, M. S.: A Constraint Based Model of Coordination in Concurrent Design Projects. www.ie.utoronto.ca/EIL/DITL/WET-ICE96/ProjectCoordination/ (1996)
7. Haselböck, A., Stumptner, M.: A Constraint-Based Architecture for Assembling Large-Scale Technical Systems. Proceedings of International Conference on Expert Systems Applications/ AI in Engineering. Edinburgh (1993)
8. Van Hentenryck, P., Saraswat, V.: Constraint Programming: Strategic Directions. J. of Constraints (2) (1997)
9. John, U.: Constraint-Based Design of Reliable Industrial Control Systems. In: Bajic, V.(eds.): Advances in Systems, Signals, Control and Computers. IAAMSAD. Durban, South Africa (1998)
10. John, U.: Model and Implementation for Constraint-Based Configuration. Proceedings of the 11th International Conference on Applications of Prolog, INAP'98. Tokyo (1998)
11. John, U.: Solving Large Configuration Problems Efficiently by Clustering the ConBaCon Model. Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-2000. Lecture Notes in Artificial Intelligence, Vol. 1821, Springer-Verlag, Berlin Heidelberg New York (2000)
12. John, U.: Configuration and Reconfiguration with Constraint-based Modelling (in German). PhD Thesis Technical University of Berlin. DISKI 255, Aka-Verlag, Berlin (2001)
13. John, U., Geske, U.: Reconfiguration of Technical Products Using ConBaCon. Proceedings of the AAAI'99 Workshop on Configuration. Orlando (1999)
14. John, U., Geske, U.: Constraint-Based Configuration of Large Systems. In: Bartenstein, O. et al: Web Knowledge Management and Decision Support. Revised Papers of 14th International Conference on Applications of Prolog, INAP 2001. Lecture Notes in Artificial Intelligence, Vol. 2543, Springer-Verlag, Berlin Heidelberg New York (2003)
15. Van Parunak et al.: Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. Proceedings of the AAAI'97 Workshop on Constraints and Agents. Providence (1997)
16. Pasik, A. J.: The Configuration Invasion. Report, Lazard Frères & Co. LLC, www.selectica.com/html/articles/Lazard1.html. New York (1998)
17. Sabin, D., Freuder, E. C.: Configuration as Composite Constraint Satisfaction. Proceedings of AAAI'96. Portland (1996)
18. Sabin, D., Weigel, R.: Product Configuration Frameworks - A Survey. IEEE- Intelligent Systems 13 (4) (1998)
19. Stumptner, M.: An Overview of Knowledge-Based Configuration. AI Communications 10 (2) (1997)