Reconfiguration of Technical Products Using ConBaCon

U. John and U. Geske

Research Institute for Computer Architecture
and Software Technology, GMD FIRST
Rudower Chaussee 5, D-12489 Berlin, Germany

{john, geske}@first.gmd.de

Abstract

When we consider large/expensive technical products/systems, it is often the case (after some time) that repair is required to replace modules that are not working and no longer commercially available or the functionality of the system needs improving. The problem is to make these changes at minimal cost.

In this paper, we outline a reconfiguration approach within our constraint-based model for the configuration of technical systems, and look at some aspects of its prototypical implementation ConBaCon (*Con*straint *Ba*sed *Con*figuration). The integration of a constraint-hierarchy transformer, allowing the computation of improvement instructions and preferences, provides a sound basis for an inexpensive reconfiguration/reconditioning of existing industrial products, including the problem of versioning of technical modules.

Introduction

The computer-assisted development of industrial products is still under intensive research. The reasons for this are relatively long development times, the costs these entail, the resulting competitive disadvantages and the error-proneness (cf. (Sabin 1996), (Axling 1996) or (Tiihonen et al. 1996)) of the development processes.

Since 1981, 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 (Tiihonen et al. 1996), (Stumptner 1997) and (Sabin and Weigel 1998). If approaches from research as well as commercial systems[1] are considered, the following general deficits are found:

- The problem specification is nondeclarative and often in the shape of a decision tree (hard to maintain).
- The sequence of interactions during the configuration process is fixed in the problem specification. Thus, a flexible configuration process, as supported by *ConBaCon*, is impossible.
- The simulation of different effects, resulting from alternative interactive decisions, is rare.
- *Support of* good *reconfigurations*, which is needed by industry, *is insufficient or nonexistent.*
- It is impossible to find optimal or near-optimal configurations.
- Sometimes the underlying algorithms fail to terminate, etc.

It is a generally accepted fact that high-quality configuration systems can be realized, especially by using of *constraint programming* (cf. (Van Hentenryck and Saraswat 1997), (Axling and Haridi 1996), (Gelle and Weigel 1996), (Faltings and Weigel 1994), (Sabin and Freuder 1996)). Successful commercial configuration systems are always dubbed "constraint-based", but this is mostly misleading because these systems do not use integrated constraint solvers to reduce the search space; they merely process a constraint-based problem specification by simple checking of constraints (cf. (Axling 1996)). Regarding genuine constraint-based configuration systems or research prototypes, it may be said that relevant publications on these approaches are often of a very general nature or exhibit limitations with regard to the quality of search-space reduction and the problem class they can handle. Especially parametric components – such as are needed, for instance, by industrial control systems – are seldom mentioned in configuration-related publications (cf. (Tiihonen et al. 1996)).

Our prototypical configuration system ConBaCon, based on the CLP language[2] CHIP, attempts to overcome the above deficits. For the most part, it was a product of the VERMEIL[3] project, which was concerned with developing concepts to support the knowledge-based development of reliable control systems. Our industrial project partner ELPRO Prozessindustrie GmbH, which is a typical producer of energy-supply and process-control systems, served as a reference enterprise.

The next section outlines the corresponding specifi-

[1] CeBIT'98 fair, Hannover/ Germany

[2] CLP=Constraint Logic Programming

[3] Funded by the German Federal Ministry for Education, Science, Research and Technology

cation language ConBaConL, which was extended by constructs for documenting modifications of the specification/product taxonomy. Section 3 introduces some key aspects of our configuration model and its realization ConBaCon, which allow the configuration of industrial products/technical systems.

The reconfiguration model based on the presented problem-solution model is treated in Section 4. The paper closes with a conclusion and some remarks on possible future extensions.

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. These specifications are composed of three parts:

- specification of the *object hierarchy*,
- specification of *context-independent constraints*,
- specification of *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 will be 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, on the one hand, different attribute value sets, and on the other, the existence or nonexistence of technical objects in the problem solution.

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.

Simple Constraints

Attribute Value Constraints and Existence Constraints *[o, Attr, VS]/not([o, Attr, VS])* ≡ the attribute *Attr* of object *o* must/must not take a value from *VS*,

exist(Objectlist)/noexist(Objectlist) ≡ 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).* Further, it is possible to specify equations over attributes.

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

table(Tablehead, Tablebody)
Tablehead specifies table columns which can be assigned to object attributes or objects. In *Tablebody*, the table lines are collected, while attribute values are assigned to attribute columns and existence/nonexistence demands are assigned to object columns. The table constraint demands that the tabular coherences between attributes and objects of the Tablehead be fulfilled. This means that at least one line of the table must be valid.

Compositional Constraints

Compositional Constraints are, besides the above-mentioned Simple Constraints, compositions of compositional constraints: *and([Cons$_1$,...,Cons$_n$])*, *or([Cons$_1$,...,Cons$_n$])*, *xor([Cons$_1$,...,Cons$_n$])*, *at_least([Cons$_1$,...,Cons$_n$],N)/ at_most([Cons$_1$,..., Cons$_n$],N)/ exact([Cons$_1$,...,Cons$_n$],N)*
≡ at_least/at_most/exactly N of the listed constraints are valid[4].

Conditional Constraints

[if(Comp_Cons$_I$), then(Comp_Cons$_T$)]
If the compositional constraint *Comp_Cons$_I$* is fulfilled, the compositional constraint *Comp_Cons$_T$* must also be fulfilled.

[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.

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. They have the shape *weak(Simp_Cons, Level)*, level being a measure of the weakness of *Simp_Cons*. In the future, it would be useful to extend ConBaConL and the corresponding problem-solution model by adding weak compositional constraints (*weak(Comp_Cons, Level)*) and weak conditional constraints (*[if(Comp_Cons$_I$), then(weak(Comp_Cons$_T$, Level))]*).

[4]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.

Modification of Product Taxonomies

In practice, the specification of product taxonomies/component catalogs 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. (Freuder 1998)). There are several reasons for the necessity of changing the product taxonomies/problem specifications. The most common one is the fact that new technical modules become available or rid ones are withdrawn. A special case of this is so called versioning. A third reason is the changing of context-independent constraints, caused for instance by 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 (Figure 1). 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 (I.), o_x is a specialization of an object o. In the second case (II.), o_x is a module of o, whereby a notional object h must be introduced at the position of o_x, which gets the specializations *new_o* and o_x.

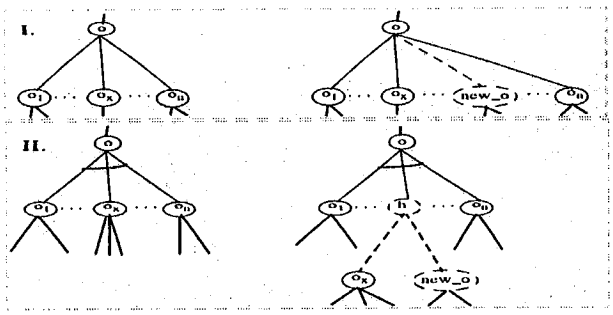

Figure 1: *Integration of New Modules*

A typical specification of ground rectifiers for large electric motors[5] – together with the problem solution using ConBaCon – is outlined in (John 1998b).

Modeling & Implementation

When transforming a problem specification, our goal is to obtain a problem-solution model that allows an efficient problem solution. The model should also support the option of high-quality interactions with the user. 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.

[5]On the basis of data provided by our industrial partner.

Objects

Each specified object (representing a technical module) that is not marked as obsolete will be transformed into a *module object* of the problem-solution model[6]. Moreover, each attribute of a specified object will be transformed into an *attribute object*, i.e., a specified object with n attributes will be represented by $n+1$ objects in the problem-solution model (Figure 2). In contrast to

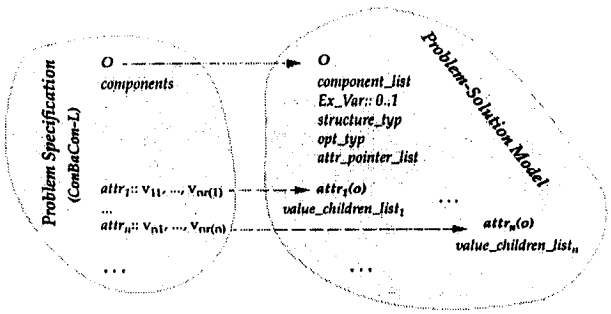

Figure 2: *Transformation of Objects*

our approach, parameterized modules are only rarely considered in configuration-related literature (cf. (Tiihonen et al. 1996)).

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_typ* = and-node) and of the specializations (*structure_typ* = 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[7]. *opt_typ* 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*

[6]Some constellations require the introduction of auxiliary module objects. These are not considered in the present paper.

[7]Restriction to the values zero and one is a simplification of the realized model. There are actually more values that reflect the existence of different technical identifiers of one technical object, these depending, for instance, on the fixed parameter values.

constraints (CE-constraints).

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 Figure 3. If it becomes obvious that an object can-

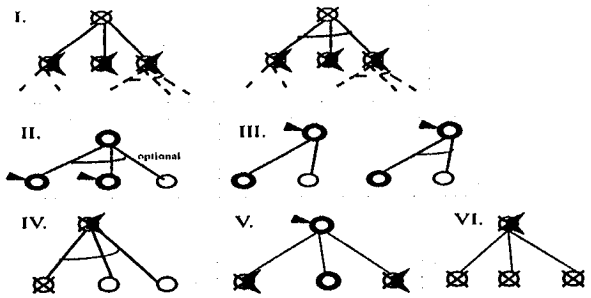

Figure 3: *Consistency-Ensuring Constraints*

not occur in the solution, it must be inferred that all components/specializations of it cannot 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). Further, 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 all specializations of an object o cannot 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. In the case that a value is deleted in the attribute value set of a specialization of an object o, the value has to 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[8]. If an attribute value is deleted in an attribute value set of an object o possessing specializations, the same value has to 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 the existence, nonexistence and attribute value sets. Moreover, the constraints formulated in the problem specification must be transformed into constraints of the problem-solution model.

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 altering 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 invalidity-marking 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. *Compositional Constraints* are normally realized in the solution model by equations and unequations 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.

Implementation

Based on the outlined problem-solution model, a flexible and efficient problem-solution process 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*[9] 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. We call the set

[8]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.

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

of the currently active module objects of the problem-solution model *Configuration Space*. Now, interactive user constraints 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 freedom of the user 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 precondition for a highly flexible configuration process. The simulation aspect of configuration processes is discussed in (John 1997). 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 will be specified as *hard* constraints, and the solution parts that should, if possible, be in the new solution or desired attribute values will be fixed as *weak* constraints. The weak constraints can be marked with several weights. The specified constraint hierarchy will be processed in an error-minimization process, which results in the generation of a set of equivalent (hard) constraints of the problem-solution model (CH Transformer). Information about the realization and application of constraint hierarchies in ConBaCon for partial improvement can be found in (Schiemann et al. 1997). At second sight, it becomes obvious that the improvement process using a constraint-hierarchy transformer provides a sound basis for reconfiguration. This is discussed in the next section.

Reconfiguration

There are two possible reasons for reconfiguring existing systems/products. On the one hand, parts/components sometimes break down and have to be replaced, repair is not possible without affecting other modules of the system. On the other hand, there may be the wish to realize a new/modified functionality of the existing system, which can result in extentions and replacements. Several additional objectives for the reconfiguration process are conceivable. Thus, it might be useful to keep the number of necessary replacements/changes to a minimum. This can be seen as a substitution goal to minimize costs. Another objective might be to make as few changes to existing parameter values as possible[10]. Owing to space limitations, we will confine ourselves here to the goal of minimal replacements/changes. Extensions – including the treatment of resource-accentuated modules – can be made quite easily. Figure 4 shows the reconfiguration process for a system s. The exact specification of the existing system s has to be merged with the current product model of s in the following way. Starting with the current product model, all technical modules/objects of the system s *which are not out of order* and *which do not need replacing to meet explicit reconfiguration demands* should be now marked as valid if they were labeled as obsolete before. They are available because they are already integrated in s. Specified context-independent constraints were processed in a similar way. For each technical module of s that is not out of order and that does not need replacing to meet explicit reconfiguration demands, a *weak existence constraint* is generated. For each optional object of the resulting product model that is not part of s and whose existence is not explicitly demanded by reconfiguration demands a *weak nonexistence constraint* is deduced. The resulting *reconfiguration product model*, which also contains the reconfiguration demands (ConBaConL-specified constraints) and the generated weak constraints, is the *new problem specification*. After transforming the specification into an adequate problem-solution model as described in the previous section, the weak nonexistence constraints and existence constraints are transformed to the sum over the existence variables Ex_Var of the objects in question or the terms "$1 - Ex\_Var$" for the weak existence constraints. The resulting sum is minimized by "restricted forward labeling" of the existence variables in question. After backjumping, the value of the (non labeled) sum term is set to the determined minimum. Thus, a configuration space is given that can be further reduced as described above. For each solution, the minimal number of replacements is ensured with respect to

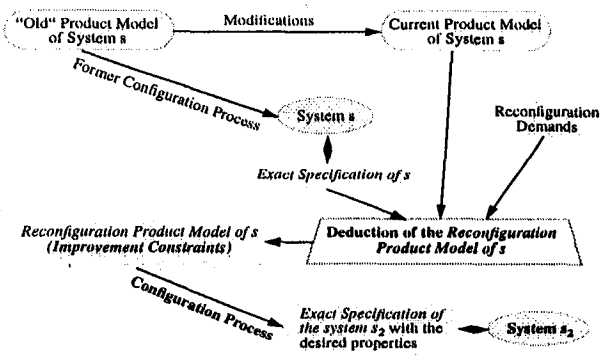

Figure 4: *Reconfiguration of a Technical System*

[10]For instance, in the field of safety control engineering, a large number of parameter value changes would necessitate expensive field simulations.

the system s, the current product model and the reconfiguration demands.

Conclusion / Future Work

We have introduced a reconfiguration approach based on our constraint-based problem-solution model for the configuration of technical systems/industrial products. An idea of the complexity of the configuration problems that can be tackled by the solution model has been given by describing the main elements of the corresponding specification language ConBaConL. So far, the prototypical realization of the presented problem-solution model ConBaCon, in the CLP language CHIP, has proved successful in the field of industrial control systems for the configuration of power-supply systems for large electric motors. 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 found is correct with respect to the problem specification and the underlying constraint solver. In addition, the completeness of the solution process is guaranteed.

The integration of a constraint-hierarchy transformer allows the computation of improvement instructions and preferences and provides a sound basis for the outlined reconfiguration approach.

By integrating a graphical problem editor into ConBaCon, the system supports innovative design processes, which is essential in many practical design problems (see (John 1998a)). One important extension of the existing problem-solution model is distribution, which is motivated by two concerns. On the one hand, it is useful for the near-optimal solution of large complex problems to develop DPS-oriented approaches[11]. More specifically, we have to develop proper problem-decomposition methods and models of corresponding agent systems. On the other hand, it is useful to support existing team structures in configuration related companies. Some work in this direction is documented in (Gupta, Chionglo, and Fox 1996; Van Parunak et al. 1997).

Interesting approaches for solving large configuration problems can be found in (Sabin and Freuder 1996) and (Fleischanderl et al. 1998). These provide ideas for dynamizing our problem-solution model.

References

T. Axling: Collaborative Interactive Design in Virtual Environments. www.sics.se/~axling/3dobelics.html.

T. Axling, S. Haridi: A Tool for Developing Interactive Configuration Applications. J. of Logic Progr., 1996.

Th. Christaller, D. Schuett (Eds.): Foundations and Applications of AI (In German). Springer. 1993.

E. Gelle: Preliminary Bridge Design. liawww.epfl.ch/~gelle/bridge-design.html. 1997.

E. Gelle, R. Weigel: Interactive Configuration using Constraint Satisfaction Techniques. Proc. of PACT'96.

Boi Faltings, Rainer Weigel: Constraint-Based Knowledge Representation for Configuration Systems. Technical Report TR-94/59 des EPFL, 1994.

G. Fleischanderl, G. Friedrich, A. Haselböck, M. Stumptner: Configuring Large Systems Using Generative Constraint Satisfaction. IEEE- Int. Systems 4/ 1998.

Eugene C. Freuder: The Role of Configuration Knowledge in Business Process. IEEE- Int. Systems 4/ 1998.

A. Günter: Models for Configuration (In German). In (Christaller and Schuett 1993).

L. Gupta, J.F. Chionglo, Mark S. Fox: A Constraint Based Model of Coordination in Concurrent Design Projects. www.ie.utoronto.ca/EIL/DITL/WET-ICE96/ProjectCoordination/. 1996.

P. van Hentenryck, V. Saraswat: Constraint Programming: Strategic Directions. J. of Constraints, 2/ 1997.

U. John: Constraint-Based Simulation of Configuration Processes. Proc. of IMACS'97, August 1997.

U. John: Constraint-Based Design of Reliable Industrial Control Systems. In "Advances in Systems, Signals, Control and Computers (V. Bajic, Ed.)". IAAM-SAD. Durban, South Africa. September 1998.

U. John: Model and Implementation for Constraint-Based Configuration. Proc. of INAP'98.

Van Parunak et al: Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. Proc. of WS Constraints and Agents on AAAI'97.

D. Sabin: www.cs.unh.edu/ccc/config. 1996.

D. Sabin, E.C. Freuder: Configuration as Composite Constraint Satisfaction. Proc. of AAAI'96.

D. Sabin, R. Weigel: Product Configuration Frameworks - A Survey. IEEE Int. Systems 4/ 1998.

A. Schiemann, U. John, U. Geske, D. Boulanger: Realization and Application of Constraint Hierarchies for Configuration of Technical Systems with ConBaCon (In German). Proc. of 12th Workshop Logic Programming (WLP'97). Munich 1997.

M. Stumptner: An Overview of Knowledge-Based Conf.. AI Communications. Vol. 10 No. 2, 1997.

J. Tiihonen, T. Soininen, T. Männistö, R. Sulonen: State-of-the-Practice in Product Configuration - A Survey of 10 Cases in the Finnish Industry. In (Tomiyama, Mäntylä, and Finger), 1996.

T. Tomiyama, M. Mäntylä, S. Finger (Eds.): Knowledge Intensive CAD. Capman & Hall, 1996.

[11] DPS = Distributed Problem Solving