Intelligent Support for Interactive Configuration of Mass-Customized Products

A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker

Computer Science and Manufacturing, University Klagenfurt, A 9020 Klagenfurt
{felfernig,friedrich,jannach,zanker}@ifit.uni-klu.ac.at

Abstract. Mass customization of configurable products made knowledge-based (and web-based) product configuration systems an important tool to support the sales engineer or end user when configuring systems according to the customer's needs. Configuration problems are often modeled as Constraint Satisfaction Problems, where the configuration process is an interactive search process. During this search process, the user may encounter situations, where a path in the search tree is reached, where no solution can be found. In this paper we show how model-based diagnosis techniques can be employed to assist the user to recover from such situations by calculating adequate (or optimal) reconfiguration or recovery actions. [1]

Keywords: Intelligent interfaces, Automated Problem Solving.

1 Introduction

Following the paradigm of mass-customization [4], products are offered to the customer in many variants. Knowledge-based configuration systems (configurators) are a successful application of AI technology and can assist the technical engineer, the sales representative, or even the customer to configure the products according both to technical restrictions and the customer's needs. An interactive configuration application has therefore to provide adequate support to guide the user through the problem solving process to come to a working solution. Modeling configuration problems as Constraint Satisfaction Problems (CSP) - as well as extensions like Dynamic CSPs [16] and Generative CSPs [9] - has shown to be applicable to a broad range of configuration problems due to the advantages of comprehensible declarative knowledge representation and effective computability. In a Constraint Satisfaction Problem, the configurable features of the product correspond to constrained problem variables with a predefined range of selectable values. In addition, constraints restrict allowed combinations of value assignments, i.e., legal product constellations. A solution (configuration) is found, if all the problem variables are assigned a value and no constraint is violated. In an interactive configuration session, the user is presented the choices and incrementally enters selections according to his/her specific requirements. During the interactive search for solutions there may be situations where after

[1] This work was partly funded by the EU commission under contract IST-1999-10688.

a sequence of selections no more solution can be found or a certain value is no longer selectable. The question arises which of these choices have to be taken back to recover from this situation. In another situation, after having already made some selections, the user decides to revise some of the prior choices. The problem then is to reconfigure the system in a way, that most of the other choices can be reused and do not have to be entered again. Note, that the calculations for reconfiguration may be done according to some optimality criterion, i.e., where the number of problem variables to be changed is minimized or some overall costs are minimized.
In this paper we present an approach to interactive reconfiguration where configuration problems are represented as CSPs with n-ary constraints and a model-based diagnostic algorithm is utilized to calculate recovery actions. The search for *suitable* alternative configurations given complete or partial configurations can be done without additional modeling efforts for repair actions. Furthermore the search for repairs can be guided by a domain dependent heuristic which takes change costs for the prior user inputs into account.
After giving an example we define the notion of reconfiguration (repair) of interactive configuration problems. After that we sketch an algorithm for the computation of such reconfiguration alternatives (with repair instructions). In the final sections we compare our work to previous work and end with conclusions.

2 Motivating Example

For demonstration purposes we show a small example from the domain of configurable personal computers, where the user can interactively make selections on certain features. The user selections for the *CSP* are unary constraints on the problem variables. During the session the user wants to revise one of the prior choices and select a feature which is inconsistent with the other choices.
Let us assume, our PC has the following configurable features.
model = {*basic, pro, luxury*}. *case*={*desktop, minitower, tower*}.
cd-drive = {*none, 36-speed,48-speed*}. *dvd-drive* = {*no, yes*}.
cd-writer = {*no, yes*}. *scsi-unit* = {*none,10GB, 20GB*}.
ide-unit = {*none, 5GB, 12GB*}. *cpu* = {*PII, PIII, AMD*}.

In addition, certain constraints have to hold on correct configurations, e.g.,
$(\textit{cd-writer} = \textit{yes}) \Rightarrow (\textit{scsi-unit} \neq \textit{none})$.
$(\textit{cd-writer} = \textit{yes}) \wedge (\textit{dvd-drive} = \textit{yes}) \wedge (\textit{cd-drive} = \textit{yes}) \Rightarrow (\textit{case} \neq \textit{desktop})$.
$(\textit{ide-unit} \neq \textit{none}) \Rightarrow (\textit{scsi-unit} = \textit{none})$.
$(\textit{scsi-unit} = \textit{none}) \Rightarrow (\textit{ide-unit} \neq \textit{none})$.
$(\textit{model} = \textit{basic}) \iff (\textit{case} = \textit{desktop})$.

Let us assume the user has already consistently configured some parts of the PC interactively and stated some requirements (R1 ... R7):
(R1) model = basic, (R2) case = desktop, (R3) cd-drive = 36-speed,
(R4) dvd-drive = yes, (R5) cd-writer = no, (R6) cpu = PII, (R7) ide-unit = 5GB.

At this moment, the user decides that he/she wants to have a CD-writer within the configuration. The configurator detects that this choice is not compatible with the prior choices, e.g., the CD-writer will require an SCSI-unit. In addition, not all of the three CD devices will fit into the small desktop case. Therefore, the selection of the CD-writer will not be possible. In addition, typical commercial systems (and online configurators) do not provide adequate support to explain the situation or to help the user to recover from it. In another situation the user may try to revise the decision on the case, where in commercial systems often all decision that were made afterwards will have to be re-entered. Finally, it may be better for the customer to revise his/her decision on the model and select the "pro" model where a mini-tower case is already included and will be cheaper than taking the "base" model where any case other than the desktop causes additional costs. This notion of optimality is also not present in nowadays systems.
It is the goal of our approach to calculate suitable configurations which retain most of the prior decisions or follows some other objective function like a minimal prize. Typically not all user decisions have to be revised and there will be several alternatives. In this context we assume that some part of the user requirements, is fixed and is regarded as a "must". In the example, the requirement "need CD-writer" should definitely not be revised. Possible "good" alternative solutions (S1 ... S4) and repair instructions are therefore, e.g.,

("change cd-drive and IDE-Unit selections" {R3,R7})
S1:(model=basic, case=desktop, cd-drive=none, cd-writer=yes, dvd-writer=yes, scsi-unit=5GB, ide-unit=none, cpu=PII).

("change dvd-drive and IDE-Unit selections" {R4,R7})
S2:(model=basic, case=desktop, cd-drive=36-speed, cd-writer= yes, dvd-writer=none, scsi-unit=5GB, ide-unit=none, cpu = PII).

("change case and IDE-Unit selections" {R2,R7})
S3:(model=basic, case=minitower, cd-drive=36-speed, cd-writer=yes, dvd-writer=yes, scsi-unit=5GB, ide-unit=none, cpu = PII).

("change model, case and IDE-Unit selections" {R1,R2,R7})
S4:(model=pro, case=minitower, cd-drive=36-speed, cd-writer= yes, dvd-writer=yes, scsi-unit=5GB, ide-unit=none, cpu = PII).

In all reconfiguration solutions, an SCSI-unit has to be installed. In addition, in reconfiguration solutions S1 and S2 we give up one of the other CD devices. In S3 we change the case and in S4 the case and the basic model (which already includes the minitower) are changed. The user decision on the CPU type does not necessarily be changed (although it would be possible to configure a completely different system). As a result, the user is prompted a set of possible repair actions (reconfigurations) where the user can select the alternative which matches his preferences best (assuming that we do not want the system to change values without notifying the user). In the following sections we will show how we can calculate a suitable set of reconfiguration solutions using a consistency-based diagnosis approach ([12], [17]) with extensions for our domain. Note, that in general we are not interested in the calculation of all possible other solutions and explanations as there may be too many for larger configuration problems. Our aim is to focus on alternatives where the suggested solutions are minimal with respect to the number of affected variables (user decisions) or some cost function (e.g., price of the components). Note, that if the number and size of these alternatives is too large, this will not help the user too much. The outcome of the diagnostic calculation will be one or more sets of variables (i.e., user decisions) that have to be changed. In addition, possible values for those variables are computed during the search. The affected variables and the value proposals correspond to actions that have to be taken for repair.

In the following section we will treat this model of interactive reconfiguration of constraint-based configuration problems more formally.

3 Defining Reconfiguration for Constraint-Based Configuration Problems

In terms of model-based diagnosis (MBD) a system is a pair $(SD, COMPS)$ where SD is the system description (describing the correct behavior of the components and $COMPS$ are the components of an actual system. Given a set OBS of observations that conflicts with SD and $COMPS$ the goal is to find an explanation (diagnosis), i.e., to find a subset $D \subseteq COMPS$ of components which - if considered faulty - explain the unexpected behavior, i.e. make $SD \cup OBS \cup \{COMPS - D\}$ satisfiable. Both the system and the observations are expressed in terms of logical sentences. We will now match the concepts of MBD with the problem sketched in Section 2.

Definition 1 *Constraint Satisfaction Problem: A Constraint Satisfaction Problem (CSP) is defined as a triplet $< X, D, C >$, where*

- *$X = \{X1, ..., Xn\}$ is a finite set of variables,*
- *each X_i can take its value from a finite domain D_{Xi}, where $D = \{D_{X1} \cup ... \cup D_{Xn}\}$, and*
- *a set C of constraints restricting the combination of values that variables can take.*

A solution to a CSP is an assignment of a value from its domain to every variable from X, in such a way that every constraint from C is satisfied. □

The configuration problem (expressed as CSP) corresponds to SD and any solution to the CSP is a valid configuration. During the interactive session, the *user requirements (UR)* are added to the CSP as unary constraints, e.g., *cd-writer=yes* from the example. If there are user constraints on all problem variables, we call UR *complete* and *partial* otherwise.

Definition 2 *User Requirements (UR): UR is a set of unary constraints for $CSP < X, D, C >$ of the form $(X_i = V)$, where $X_i \in X$, V is an element of the*

Domain D_{Xi} and each X_i appears at most once in a constraint from UR. UR is called "complete" if for every $x \in X$ there is an unary constraint $(x = V)$ in UR; else UR is called "partial". □

The user requirements are divided in two categories: *revisable* requirements RR (corresponding to $COMPS$) and *non-revisable* ones which must not be changed (corresponding to OBS) denoted as NR.

Definition 3 *Reconfiguration Problem: A Reconfiguration Problem (RCP) is a triple ($< X, D, C >$,NR,RR) where $< X, D, C >$ is a CSP and $UR = RR \cup NR$ are user requirements.* □

A reconfiguration problem arises, if the $CSP < X, D, C \cup UR >$ is unsatisfiable. Our aim is to calculate a (minimal) subset S (diagnosis) of elements from RR, which have to be relaxed, such that the remaining problem is satisfiable.

Definition 4 *Reconfiguration Diagnosis: A Reconfiguration Diagnosis (S) for a Reconfiguration Problem $(< X, D, C >, NR, RR)$ is a set $S \subseteq RR$ such that the $CSP < X, D, C \cup NR \cup (RR - S) >$ is satisfiable.* □

Definition 5 *Minimal Reconfiguration Diagnosis: A Reconfiguration Diagnosis S for a RCP $(< X, D, C >, NR, RR)$ is said to be minimal iff there exists no $S' \subset S$ such that S' is a diagnosis for $(< X, D, C >, NR, RR)$.* □

Following a diagnose-and-repair approach we can calculate reconfiguration solutions in the context of a diagnosis:

Definition 6 *Reconfiguration Solution: Given a Reconfiguration Diagnosis S for $RCP(< X, D, C >, NR, RR)$, every solution to the $CSP < X, D, C \cup NR \cup (RR - S) >$ is a reconfiguration solution for RCP.* □

Note that, if the $CSP < X, D, C \cup NR >$ is satisfiable, a reconfiguration solution will always exist, since in the worst case the Reconfiguration Diagnosis S will contain all elements of RR.

With the definitions so far, all solutions to the original CSP joined with the unrevisable user requirements are therefore legal reconfiguration alternatives. As a result, it is important to have a discrimination criterion for these alternatives, i.e., a preference function describing the "fit" or optimality has to be defined. One possibility to do this, is to assign each constraint from RR a corresponding value (weight), expressing e.g., the costs of changing the value or some user preference. In our example, changing the case of the PC could be undesirable for the user (expressed through preferences/weights). Consequently, a "good" reconfiguration solution would avoid relaxing this requirement. The fit of a reconfiguration solution could therefore be defined as the sum of the weights of the constraints from RR. If the constant value "1" is assigned to each of this constraints, the best reconfiguration solution will be the one, which minimizes the number of constraints in S.

Definition 7 *Optimal Reconfiguration Diagnosis:*
Let $RCP(< X, D, C >, NR, RR)$ be a Reconfiguration Problem and Π be a function assigning each constraint from RR a value w, $w > 0$. Given a Reconfiguration Diagnosis S for RCP, let $\Phi(S) = \Sigma_{e \in S} \Pi(e)$. An Optimal Reconfiguration Diagnosis for RCP is a reconfiguration diagnosis S such that there is no other reconfiguration diagnosis S' such that $\Phi(S') < \Phi(S)$. □

Note, that other optimality functions are possible, which take preferences for individual values for the variables into account.

4 Computing Reconfigurations

Having mapped the reconfiguration problem to a consistency-based diagnosis problem, we can use the standard hitting set algorithm ([17],[12]) for the calculation of diagnoses.

The approach is based on the notion of *conflict sets* and on the algorithm to solve overconstrained CSPs from [3]. In addition we adapt the algorithm for the reconfiguration problem and prune the search tree through the usage of an evaluation function for optimal reconfiguration diagnoses.

Definition 8 *Conflict Set: Given a reconfiguration problem $RCP(< X, D, C >, NR, RR)$, a conflict set is defined as a set $CS \subseteq RR$ such that the $CSP(< X, D, C \cup NR \cup CS >)$ is unsatisfiable.* □

Informally speaking, a conflict set is a subset of the constraints of the revisable user requirements which definitely violate a constraint from the original constraint satisfaction problem with the unrevisable requirements (or makes it at least impossible to get to a solution). We construct a directed acyclic graph (DAG) for the computation of minimal hitting sets according to [17] for the reconfiguration problem $RCP(< X, D, C >, NR, RR)$. The DAG is generated in breadth-first manner, since we are interested in diagnoses of minimal cardinality. The nodes are labeled with conflict sets for the reconfiguration problem, edges leading away are labeled with elements from these conflicts. The path from the root node to a node is denoted as $H(n)$. During the construction of the HS-DAG we make a call to the "theorem prover" (which is a constraint solver in our case) $TP(< X, D, C \cup NR >, RR - H(n) >)$ at each node which returns true and a solution if the $CSP(< X, D, C \cup NR \cup (RR - H(n)) >)$ can be solved. Otherwise, a conflict set for RCP is returned. When calculating solutions we can either compute all possible remaining solutions for that diagnosis, can optimize the solution according to some criterion, or can simply return an arbitrary solution (whereby this solution search can be guided by an additional heuristic).

Additional Closing of Nodes

In addition to the techniques for pruning and closing of nodes from the basic algorithm we can introduce an additional rule to reduce the size of the HS-DAG: If a new node n with $H(n)$ is to be generated, we can calculate $\Phi(H(n))$ which will be the reconfiguration costs for that new node. If there is already another node n' containing a diagnosis S in the tree where $\Phi(S) < \Phi(H(n))$ then node n can be closed, because there cannot be a better reconfiguration (according to the optimality criterion) under that node.

5 Experimental Results

We implemented a prototype environment for the proposed approach, where we support interactive reconfiguration. To allow for web-based interactive configuration we used a Java-based constraint solver library (JSolver [5]). With this extensible set of Java-classes one can solve n-ary constraint satisfaction problems using a forward propagation and backtracking search algorithm efficiently. The implementation of the HS-DAG calculation and the extensions for reconfiguration was also done as a set of Java classes, whereby the HS-DAG algorithm can also be employed for other application scenarios of model-based diagnosis [7]. When building an interactive configuration system, the constraint satisfaction problem has to be defined using the JSolver libraries. This configurator is then incorporated into a graphical user interface, where the user interactively makes his selections. In a situation when no more solutions can be found or some values are no longer selectable, the choices so far are handed over to the diagnostic engine, which calculates reconfiguration alternatives. Finally, the system can present a number n of best reconfiguration alternatives to the user to choose from in order to recover from the current situation. Given our simplified example from the domain of configurable PCs, these results can be computed instantaneously on a standard P-II computer. In our application scenario, both the configurator with the knowledge base, i.e., the selectable features and the allowed combinations of values, and the reconfiguration libraries, are integrated into an applet and can be viewed within the user's browser. So the communication costs with constraint checking/solving at the server side can be avoided. The reconfiguration alternatives can not only be presented to the user to choose from but the values of the selected alternative can also be changed automatically. Our experiments showed the applicability of our approach to real-world sized problems for interactive (online-)configuration with a number of about 30 user inputs. Note, that the coniguration problem itself contains possibly hundreds of variables, but these variables will not affect the computation time for diagnosis, because only the user inputs are revisable. However we found that diagnoses of higher cardinality (more than 7) will not help the user much because the alternatives are too complex. Therefore, the search depth can be limited to a certain level.
For diagnosis of larger problems the **exploitation of structural abstractions** in the system (knowledge base) has shown to be a promising approach ([2],[8]). The idea is to diagnose the system on different levels of abstraction, where several components of the system are grouped together and treated as one. This reduces the number of diagnosable components on the more abstract levels and the search space. The results of diagnosis on this abstract levels can be used to focus the diagnostic process on the next level. For our application domain this notion

of grouping of components (in our case the user inputs) is intuitive, because typically the configurable system is decomposed into several parts and for each part there may be several user inputs which can be grouped. When searching for diagnoses we test the system on the more abstract level where always these groups of user inputs are relaxed for one consistency check. Note, that nearly no additional modeling effort for the different abstraction levels is needed (as e.g., described in [2]) because no abstract "behavior" is needed. (For detailed information on the hierarchical approach in another application domain which is implemented in the prototype, see [8].)

6 Related Work

Rich and Sidner [18] present an approach using a collaborative agent based on shared plans supporting the user to cooperatively solve a (e.g., travel planning) problem. The agent guides the user through the interactive process to find a solution. The system records previous user actions and supports the user to recover from situations where the user is lost or stuck in some stage of the process. The system then presents a suggestion to reduce some constraints to come to a solution. This is where our approach can be incorporated to efficiently find a suitable set of recovery actions for configuration problems. The approach of using a collaborative agent can be an interesting for the configuration domain. However, the generation of discourses (maybe using natural language) is beyond the scope of this paper, although the configuration domain can be an application domain for intelligent collaborative agents.

Mannistö et al. [14] describe a general framework for reconfiguration based on a given configuration and reconfiguration language. They assume the explicit existence of reconfiguration rules/operations which can be applied to an existing configuration. The optimality criterion can be expressed through a valuation function for the individual operations. They present a study on current practices of reconfiguration in industry showing the need for more automated support for reconfiguration. Our framework however does not require the existence of explicit reconfiguration rules, whereby the formulation of this knowledge causes additional knowledge acquisition and maintenance efforts. Using their notion of reconfiguration, the main focus lies in reconfiguration of an already installed system, whereas in our approach the interactive setting is of importance (although our approach can also be applied to reconfiguration of existing systems).

Crow and Rushby [6] extend Reiter's [17] framework for model-based diagnosis for fault detection, identification and reconfiguration. They introduce an additional predicate $rcfg$ (similarly to the ab predicate in model-based Diagnosis) on components of the system to be diagnosed. They describe a framework for reconfiguration merging model-based diagnosis with repair, and claiming that automated repair should be the consequence of the diagnostic process. However, in their approach, the reconfiguration knowledge (which comprises a model of "spare" components) has to be modeled explicitly. In addition, integration of optimality criteria is only considered to a small amount, whereas we think that the notion of "good" or "optimal" reconfigurations is an intrinsic feature of reconfiguration.

Bakker et al. [3] show an algorithm for diagnosing over-constrained Constraint Satisfaction Problems using a model-based diagnosis algorithm. This work strongly correlates with ours from the viewpoint of computation and algorithms. However, they focus on general CSPs and not on (interactive) configuration problems with their specific characteristics of constrainedness. In addition, in their algorithm, any of the given constraints of the knowledge base can be revised, whereas we only relax user decisions, i.e., the unary constraints. Therefore the size of the knowledge base, i.e., the number of constraints, does not affect the size of the HS-DAG, which is a limiting factor for efficient diagnosis. In our approach, only the user-selectable variables (resp. only the already given inputs) are taken into account making the resulting HS-DAG smaller in size.

The notion that solving a configuration problem by repairing an existing (and inconsistent) configuration can be more effective than solving the problem from scratch has been brought up already in one of the first configuration design problem solvers [13]. In this system, based on the VT elevator domain, a propose-and-revise strategy problem solving method was described using fixes when parts of the configuration (design) are inconsistent. A heuristic method of repairing a special inconsistent Constraint Satisfaction Problem (n-queens) was presented in [15]. They argue, that for this problem it is more effective to repair an existing setting (configuration) based on the min-conflicts heuristic than building a solution from scratch. Their approach uses a hill-climbing algorithm (with or without backtracking) where one starts with a situation which is "near" to a solution and incrementally changes parts of the configuration. At each step, a slight improvement of the overall situation is achieved. This can be related to our approach, where we start from a "good" situation (the user requirements which can be a complete configuration) and try to reduce conflicts and do not calculate solutions from scratch. Compared to our approach, this repair approach and other (local repair algorithms) only return one single reconfiguration without regarding any optimality criterions and without showing different alternatives.

Freuder et al. [11] describe an approach to solve overconstrained Constraint Satisfaction Problems, where we have a situation, where no solution exists (which can be compared to our reconfiguration situation where the unary user constraints make the problem overconstrained). In their approach, the goal is therefore not to find a "maximal" solution, where all constraints are satisfied (which is not possible) but rather find a solution where a maximal number of constraints is satisfied. During the search process, a search path does not fail when a single inconsistency is encountered but rather when enough inconsistencies (at a certain cut-off number) have been encountered, which can be compared to some extent with the additional tree pruning in our diagnostic algorithm. However, in the reconfiguration setting only the unary user constraints can be relaxed but not the constraints from the original CSP like in Freuder's approach.

Amilhastre et al. [1] present an approach to handle interactivity during the search for solutions for a constraint based configuration problem. They deal with situations during an interactive configuration session, when no solution can be found or features are no more selectable at a certain stage. They propose the construction of a finite automaton based on all possible solutions of the CSP in a preprocessing step before the configuration system is employed. At run time, corresponding answers or explanations (i.e., repair possibilities) for the user questions mentioned above can be calculated. This interactive setting can also be handled with our approach for small-sized problems, whereby immediate response times cannot be guaranteed. In a larger reconfiguration setting, the applicability of constructing an automaton describing all possible solutions to the original CSP may be limited since the graph operations are still exponential in the size of revisable user choices.

7 Conclusions

More and more configurable products are offered to the customer and made interactive configuration systems a valuable tool to support the customer in assembling a functioning system according to his specific needs. The complexity of the interactive configuration process creates new demands on the user interface, because the configuration process is an interactive search process in constraint-based systems. In nowadays systems, only little support is given to guide the user in cases no adequate solution can be found after the user entered some choices. In this paper we present an approach for an intelligent support for interactive configuration of constraint-based configuration problems. Reconfiguration (repair) alternatives can be presented to the user in cases the search process has reached a dead end. We described a framework for computing (optimal) reconfiguration solutions based on a standard algorithm for model-based diagnosis using conflicts for focusing purposes and extensions for our domain. In our approach, the reconfiguration knowledge (repair actions) does not have to be modeled explicitly but is inherently given in the basic underlying CSP model. Our approach is capable of providing reconfiguration solutions for CSPs (and also Dynamic Constraint Satisfaction problems) with n-ary constraints given partial or complete initial configurations. An extensible prototype environment was built to test the applicability of the approach in an interactive setting, whereby the implemented libraries can be incorporated into a system with additional user support.

References

1. J. Amilhastre, H. Fargier: Handling interactivity in a constraint-based approach of configuration, ECAI'2000 Workshop on Configuration, Berlin, 2000.
2. K. Autio, R. Reiter: Structural abstraction in Model-based Diagnosis, Proc. ECAI'98, Brighton, UK, 1998.
3. R.R. Bakker and F. Dikker and F. Tempelman and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. Proc: IJCAI'93, p. 276-281, Chambery, Morgan Kaufmann, 1993.
4. D. Brady, K. Kerwin, D. Welch et al.: Customizing for the masses, Business Week, No. 3673, March 2000.

5. Hon Wai Chun , Constraint Programming in Java with JSolver, Proc. PACLP'99, London, UK, 1999.
6. J. Crow, J. M. Rushby: Model-Based Reconfiguration: Toward an Integration with Diagnosis, AAAI'91, California, 1991.
7. A. Felfernig, G.Friedrich, D. Jannach, and M. Stumptner, Consistency based diagnosis of configuration knowledge-bases. Proceedings ECAI 2000, Berlin, 2000.
8. A. Felfernig, G.Friedrich, D. Jannach, and M. Stumptner: Exploiting structural abstraction for consistency-based diagnosis of configurator knowlegde bases. ECAI Workshop on Configuration, Berlin, 2000.
9. G. Fleischanderl, G. Friedrich, A. Haselboeck, H. Schreiner and M. Stumptner, Configuring Large Systems Using Generative Constraint Satisfaction, IEEE Intelligent Systems, July/August, 1998.
10. G. Friedrich, W. Nejdl: Choosing Observations and Actions in Model-Based Diagnosis/Repair Systems, Proc. KR'92, Massachusetts, 1992.
11. E. Freuder, R. J. Wallace: Partial Constraint Satisfaction, Artificial Intelligence (58), 1992.
12. R. Greiner, B.A. Smith, and R.W. Wilkerson: A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence, 41(1), 1989.
13. S. Marco, J. Stout, and J. McDermott: VT: An expert elevator designer that uses knowledge-based backtracking. AI Magazine, 9(2), 1988.
14. T. Mannistö., T. Soininen, J. Tiihonen and R. Sulonen. Framework and Conceptual Model for Reconfiguration. AAAI'99 Workshop on Configuration, Orlando, Florida, 1999.
15. S. Minton, M. D. Johnston, A. Philips, P. Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction problems, Artificial Intelligence 58, p. 161-205, 1992.
16. S. Mittal, B. Falkenhainer: Dynamic Constraint Satisfaction Problems, In Proc. AAAI'90, August 1990.
17. R. Reiter: A theory of diagnosis from first principles. Artificial Intelligence, 32(1), 1987.
18. C. Rich and C.L. Sidner, Adding a collaborative agent to graphical user interfaces, Proc. ACM Symposium on User Interface Software and Technology, Seattle, 1996.