Multi-site product configuration of telecommunication switches

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Christian Russ, and Markus Zanker
Universität Klagenfurt
Universitätsstraße 65, 9020 Klagenfurt, Austria
{felfernig, friedrich, jannach, russ, zanker}@ifit.uni-klu.ac.at

ABSTRACT

Knowledge-based product configurators support their users in tailoring configurable products according to their specific demands and these systems have been successfully applied in many industrial sectors over the last decades. However, within today's networked economy, the complex solutions offered to the customers are in many cases assembled from configurable sub-products themselves. Within this paper we describe a business case where due to organisational and confidentiality reasons a single-configurator approach is not applicable and several configurators along the supply chain must cooperate in finding correct product configurations and in presenting them to an online customer. We present an algorithm based on Constraint Satisfaction that takes the specific characteristics of the problem domain into account and compare our approach to other work in the field of Distributed Problem Solving.
The implementation framework for distributed configuration which is currently developed in the EU-funded project CAWICOMS[1] is discussed in the final sections.

KEY WORDS: Product Configuration, Distributed Artificial Intelligence, eCommerce.

1. INTRODUCTION

Mass-customization can be defined as a business strategy where customers can tailor a product or service according to their specific needs, i.e., typically the customer can decide which features and options should be included in his/her configuration and set parameters for the personalized variant of the product. Typically, there are several technical or organisational restrictions on legal combinations of the parameters and options such that intelligent support is required to check the consistency of user choices and compute the set of components that have to be included in the solution. Due to the complexity of the task, this support has to be provided by intelligent product configuration systems (configurators).
Over the last decades, product configuration has been a successful application field of AI technology [1] and is applied both in simple sales-configuration scenarios as well as for the configuration of complex technical systems like telecommunication switches [2]. As a result, powerful tools (e.g., ILOG Configurator [3]) are available on the market, whereby the most successful systems rely on Constraint Satisfaction as underlying reasoning mechanism.
However, two related aspects of nowadays' digital economy pose new requirements on the software systems that support the (online-) selling and the configuration processes: Complex products and services are not provisioned or manufactured by one single company but are assembled from components stemming from highly specialized providers which altogether constitute a supply chain. On the other hand, from the customer's perspective there has to be one single point of interaction, where s/he can rather purchase complete *solutions* from an integrator/reseller than individual *products*.
Our application domain is from the area of telecommunication solutions for small and medium-sized companies that include e.g., advanced in-house telephony, voice messaging, billing services as well as end-user telephone devices. Such an advanced solution (which is sold by one of our project partners) consists - besides a main switch – of several additional hardware and software components, e.g., network routers for IP-based services, that are provided by external suppliers or other organizational units (Figure 1). The main challenge in that setting is that both the main switch as well as the subcomponents may be configurable and there are interdependencies between these components; due to confidentiality reasons or due to the fact that the knowledge on the configurable product itself is distributed and maintained at different sites, it is not possible to integrate this knowledge into one centralized knowledge base. Therefore, the configuration of the complete solution must

[1] *Customer-Adaptive Web Interface for the Configuration of products and services with Multiple Suppliers.*
The work takes place with the financial support of the IST Programme of the European Union under contract IST-1999-10688. (See http://www.cawicoms.org)

take place in a decentralized and distributed manner. Finally, already existing legacy configuration systems for the add-on products have to be integrated in order to support the joint configuration of the overall solution.

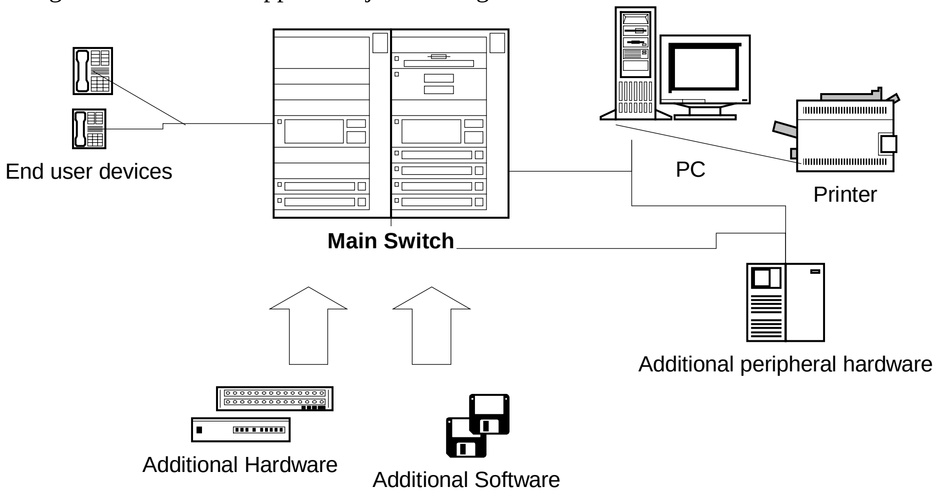

Figure 1 Telecommunication switch with add-ons

In the following sections we will describe the rationale of distributed configuration using a constraint-based approach and provide a sound and complete algorithm for distributed computation of solutions. We will relate this approach to other techniques from distributed problem solving and sketch the implementation in the CAWICOMS configuration workbench.

2. DISTRIBUTED PROBLEM SOLVING CHARACTERISTICS

In the light of the growth of Internet-enabled communication and commerce, the field of Distributed Artificial Intelligence is rapidly growing and is an active research area. On the one hand multi-agent systems (MAS) with loosely coupled, autonomous and negotiating agents are capable of performing cooperative tasks, whereas on the other hand originally centralized problem solving mechanisms, e.g., Constraint Satisfaction [4], are extended to cope with the distributedness of the task. Altogether, both approaches require some sort of common understanding and vocabulary of the problem domain e.g., an ontology of the domain, as well as a protocol for interaction to solve the overall problem.

In addition, in our application domain some special characteristics of the interaction between the involved configuration systems have to be taken into account:

- The configurators do not necessarily work in parallel like agents do, i.e., we are given a sort of a multi-tier client-server interaction, whereby during the configuration of the overall solution - including the main switch - the responsible configurators of the add-on products are only contacted on demand. For instance, the detailed configuration of some optional router hardware is only done if the router is required in the overall solution.
- Consequently, the involved configurators do not have equal priorities, because of the hierarchical order between these systems given by the supply chain. Such an prioritization is also required in approaches like Distributed CSPs [5], whereby in our application this ordering is solely determined by the supply chain and is tree-structured with respect to the subassemblies that build the overall solution[2]. In order to communicate during the search for solutions, neighbouring nodes in the tree of interacting configurators have to be integrated, i.e., they have to share parts of their product model.

[2] Note that the supply chain itself is not necessarily tree-structured with respect to the involved companies.

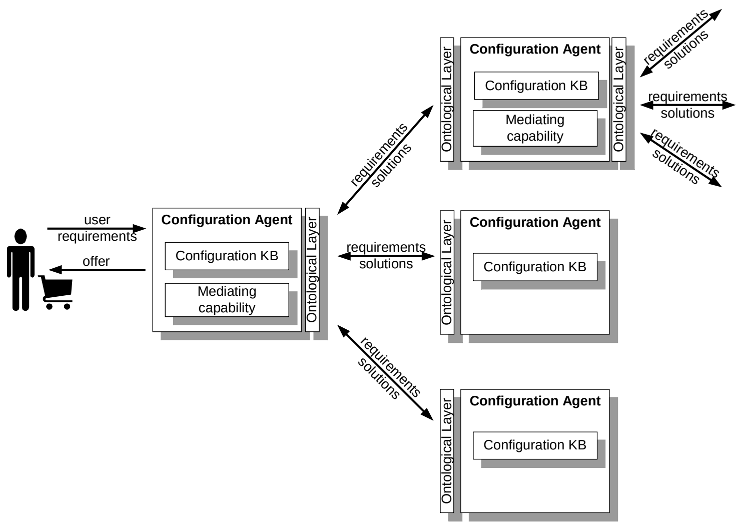

Figure 2 Tree-structured supply chain

- The involved configuration systems at the leaf level of the tree may not all implement the same problem solving mechanisms; therefore, an intermediate layer providing minimal communication facilities as well as a mapping between ontological concepts has to be implemented for the participating configurators.

3. BASIC PROBLEM SOLVING ALGORITHM

Distributed configuration in the CAWICOMS framework [6] is based on Constraint Satisfaction (CS) techniques. This is a natural choice because Constraint Satisfaction has shown to be an adequate mechanism to represent and solve configuration problems ([2],[7]) and because of the availability of industrial-strength constraint solvers. We will now describe the basic distributed algorithm of our approach based on a standard CSP mechanism with a specific form of dynamic variable activation (DCSP, see [8]). Note, that in the implementation we extend the basic forward-checking and backtracking constraint solving algorithm implemented in ILOGs[3] JConfigurator library; however the approach can generally be applied to any solver relying on this search technique.

The overall problem is divided into a set of extended CSPs (*Dynamic Component - CSP*) sharing variables, whereby the assignment of a value for one individual variable lies in the responsibility of exactly one DC-CSP solver. Furthermore, each DC-CSP is assigned a possibly empty set *S* of *supplying* DC-CSPs with which it shares some variables, whereby sharing can be done in two ways in a client-server like relationship:

a) a supplying DC-CSP can "publish" some variables to the configurator at the next higher level of the supply chain; these variables are integrated into the DC-CSP at the next level where additional constraints may be defined. The values for these variables are computed by the supplier configurator and communicated to its client who checks the consistency with its local problem.

b) some variables defined in a client DC-CSP may be *relevant* for the configurator at the next lower level of the supply chain, i.e, changes in these variables have to be communicated to the supplier.

Within our approach we require that the graph defined by these dependencies between DC-CSPs form a strict tree structure, i.e., sharing of variables can only be done between two neighboring DC-CSPs.

[3] see http://www.ilog.com

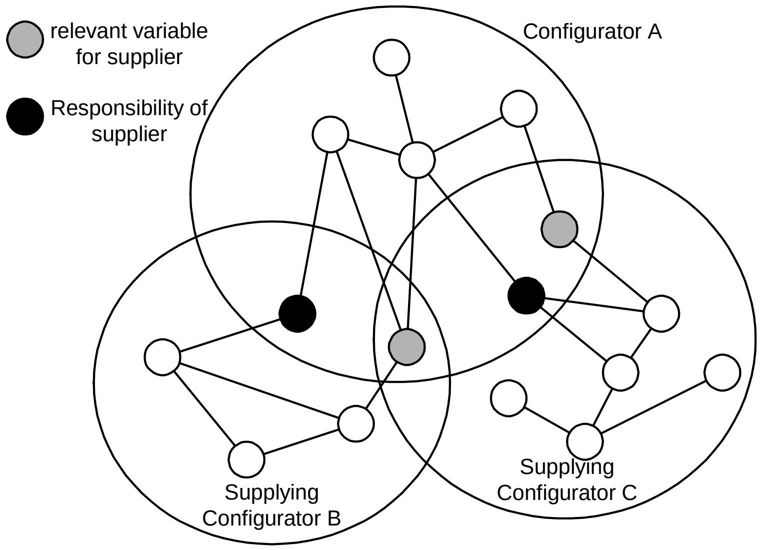

Figure 3 Sharing variables

We will first describe the "local" extended Constraint Satisfaction Problems that a participating configurator has to solve.

Definition (DC-CSP): *A Dynamic-Component CSP consists of*

- *a finite set of variables $V = \{v_1, \dots v_n\}$,*
- *each assigned a finite domain $D = \{d_1, \dots d_n\}$,*
- *a set C of constraints restricting the combinations of values the variables can simultaneously take, and*
- *a set S of suppliers involved in solving the DC-CSP and a distinguished activation constraint AC for each $s \in S$ (denoted as s.AC).*

Each variable $v \in V$ can be marked with one $s \in S$ (denoted as v.supplier) which is responsible for assigning a value for it and a set $r \subseteq S$ (denoted as v.relevant) for which an assignment to v is relevant.

$V = V_{init} \cup V_{dyn}$, whereby V_{init} is the set of initially active variables and V_{dyn} contains the set of dynamically activated variables.

An activation constraint s.AC describes the condition under which the variables of supplier s have to be activated.

Compared with the classical Dynamic CSP, activation is not done on the level of variables but rather complete subassemblies can be activated. The activation of the variables of a supplier is based on a arbitrarily complex constraint[4]. The overall distributed problem consists of a strict tree of such *DC-CSP*s.

Definition (DC-CSP-Tree): *A DC-CSP-Tree T consists of a set of DC-CSPs forming a tree structure according to the supplier sets S of the involved DC-CSPs , where*

- *one DC-CSP is the root of the tree, and*
- *the set of suppliers S for leaf-nodes is Ø .*

Informally, two DC-CSPs can only *share* variables if they are in a predecessor-successor relationship in the DC-CSP-tree.

Definition (Solution to a DC-CSP): *A solution to a DC-CSP is an assignment of a value from its domain to every variable in V_{init}, and to every variable v in V_{dyn} where v.supplier = s and s.AC is satisfied in a way that all constraints in C are satisfied.*

For each $s \in S$ where s.AC is satisfied, the set of assignments Φ to the set $SV = \{v \in V \mid v.supplier = s \vee s \in v.relevant\}$ must be part of a solution of the DC-CSP of supplier s.

The above definition states that we have to ensure that all variables of the suppliers that are *active* are assigned a value and that the assignments of values to these subsets of shared variables are consistent with the supplier's *DC-CSP*. In this supply-chain setting, the goal is then to find a solution to the *DC-CSP* which is the root of tree. Finding such a solution involves searching for solutions to other DC-CSPs depending on the activation of direct successors of DC-CSPs in the tree. We will now describe the extended forward-checking backtracking algorithm that is implemented in an inner node of the tree based on the description of [4] (FC-1, p. 127).

[4] Note that we assume that each subassembly in the local problem is provided by a different supplier.

```
procedure SDFC-Start($V_{init}$, $V_{dyn}$,D,C)
        SDFC($V_{init}$,{}, D,C)
end;
procedure SDFC(Unlabelled,Labels,D,C)
1   S = set of suppliers, where S.AC is satisfied
    // compute set of currently active variables which are not assigned a value
2   Unlabelled = Unlabelled $\cup \{v \in V_{dyn} \mid v.supplier \in S \wedge \langle v,\_\rangle \notin Labels\}$
3   if Unlabelled = {} then return Labels;
4   select x from Unlabelled; supplier = x.supplier;
5   if supplier $\neq$ {} then   // found remote var; collect all variables for that supplier
6     svars = $\{v \in Unlabelled \mid v.supplier = supplier\}$// collect set of relevant variables
7     rvars = $\{v \in Unlabelled \mid supplier \in v.relevant\}$
8     domains = current domains of svars and rvars
9     supplier.reset();
10    repeat      // try to retrieve adequate solution from supplier
11      sLables = supplier.nextSolution(svars,rvars, domains)
12      if (sLabels $\neq$ NIL) then // test supplier's solution for consistency
13          D' = integrateSolution(svars, sLabels)
14          if no constraint violated and no domain in D' empty then
15            result = SDFC(Unlabelled – {svars},  Labels + sLabels, D')
16            if result$\neq$ NIL then return (result)
17    until sLabels = NIL;
18    return NIL  // no solution from supplier
19  else      // local var
20    repeat
21      select d from $D_x$ ; delete d from $D_x$;
22      if (Labels + <x,d>) violates no constraints then
23          rSuppliers = v.relevant  // check, if value selection ok for suppliers.
24          if chkRel(rSuppliers,Labels + <x,d>) = ok
25          D' = Update(Unlabelled – {x}, D, C, <x,d>)
26          if (no domain in D' is empty) then
27            result = SDFC(Unlabelled – {x}, Labels +   {x,d},D')
28            if result $\neq$ NIL then return (result);
29    until $D_x$ = {}
30    return NIL// no solution found
end
```

Listing 1. Procedure SDFC

The procedure SDFC starts with computing the set of currently active variables (*Unlabelled*) that still have to be assigned a value by evaluating the activation constraints. Next, it selects an unassigned variable and tests whether it is to be assigned locally or by a supplier configurator (5). In the latter case, we compute the sets of all variables of that supplier (6), the set of all variables that are relevant (7) as well as the current domains of these variables (8). Before retrieving the first or another solution for this set of variables (11), we call the *reset* procedure of the supplying configurator (9) – see Listing 3: This is done because we assume that a supplier can provide more than one solution given some inputs; in the loop from (10) to (17) we try to find a suitable solution that can be integrated into the problem at the higher level. Therefore the supplier has to maintain the list of solutions (*oldLabels*) it provides in the current context. *reset()* causes this information to be discarded.
Note, that the inner nodes in the tree of suppliers will use procedure *SDFC* (with the extension of storing prior solutions) to assign labels; however, we do not want to enforce the usage of this procedure at the leaf nodes in order to support integration of other configuration systems. If the supplier is able to provide labels for the variables (12), we try to integrate these into the problem on the higher level (13) using procedure *integrateSolution* in Listing 2 and propagate domain reductions – (compare procedure *Update* from [4]).
In cases where this integration is successful we continue solving the CSP with the remaining unlabelled variables (15). Note that when calling a supplier configurator, *all* variables of the respective supplier are assigned a value at once. In addition, in case of backtracking, the loop from (10) to (17) will try to find another complete assignment for that supplier.
Beginning at line (20), the standard backtracking algorithm from [4] is used with extensions in lines (23,24): When we try to select a value for a variable which is marked to be *relevant* for some suppliers, we have to check both the consistency with the local constraints as well as the consistency of the value with those suppliers for which this variable is relevant and which have already configured their subassembly. In procedure *chkRel* (Listing 2) we therefore collect all the variables from each

supplier together with the relevant variables and their current domains and call procedure *checkConsistency* at the supplier configurator.

```
procedure integrateSolution(svars, sLabels)
  if (Labels ∪ sLabels) violates no constraint then
        for each  <x,v> ∈ sLabels
              D' = Update(Unlabelled – svars,D, C, <x,v>)
end

procedureUpdate(Unlabelled,D,C,Label)
  D' = D;
  /* propagate values and  reduce domains. */
  return (D')
end

procedure chkRel (rSuppliers, Labels)
   for each supplier in rSuppliers
       /* collect information for supplier */
        svars = {v ∈ V | v. supplier = supplier}
       rvars = {v ∈ V | supplier ∈  v.relevant)}
        curdomains = current domains of svars and rvars
       if ∃ x ∈  svars where <x,v> ∈ Labels then
       /* already configured subassembly */
          result=supplier.checkConsistency(svars,rvars, domains)
       if result = false then return false
  return true        /* everything ok */
end
```

Listing 2. Auxiliary procedures

The next listing contains the procedures that have to be implemented by the supplier configurators:

```
procedure checkConsistency(svars, rvars, domains)
   if domains not empty and assignments consistent with local constraints then
              return true
   else return false
end

procedure reset()
          oldLabels = {}
end

procedure nextSolution(svars,rvars,curDomain)
       sLabels = assign labels to svars observing local constraints and reduced domains from curDomain (using SDFC)
                      whereby labels ∉ oldLabels.
       if no consistent assignment possible then
          return NIL
       oldLabels = oldLabels ∪ sLabels
       return sLabels
end
```

Listing 3. Procedures of supplier confiugurator

The minimal requirements for integration of an existing configuration system at leaf nodes is therefore the implementation of three functions: assigning values to variables (*nextSolution*), checking for consistency of assignments and domain reductions as well as resetting the set of previously returned solutions.

*COMPLETENESS***:** The basis of the sequential distributed algorithm is a standard backtracking algorithm. Therefore, the completeness of the algorithm is ensured by exhaustive domain exploration. Consequently, the algorithm will find a solution if one exists. The only modifications to the basic procedure lie in dynamic activation of variables which involves some additional variable ordering (all variables of one supplier are instantiated in one step). More details on the required properties for finding solutions to distributed configuration problems can be found in [9].

4. IMPLEMENTATION FRAMEWORK

One of the main goals of the CAWICOMS project is the development of a framework for the rapid development of distributed configuration applications. The two major issues within this area are mechanisms for distributed configuration problem solving and the development of tools and techniques for knowledge acquisition and knowledge integration between cooperating (heterogeneous) configuration systems. In the current prototype, ILOG's JConfigurator libraries are used for the problem solving task: Based on Constraint Satisfaction, this tool provides a more abstract layer for the configuration domain, i.e., instead of speaking of *variables*, the configuration problem is described in terms of *components*, *properties* and *interconnections* between components (see [3],[10]). We extended this *preference-based* search mechanism according to the presented algorithm in a way that a supplier is contacted in situations when during the search process it is determined that a component or subassembly from a supplier has to be included in the configuration.
In order to allow integration of heterogeneous configuration systems, data exchange in the supply chain is facilitated using an XML-based protocol. This mechanism allows us to exchange complex data structures using XML-Schema documents and also serves as a generic interaction protocol for configuration problem solving over HTTP (for more details, see http://www.cawicoms.org).

Beside adequate algorithms for distributed problem solving, one of the main problems of such systems relates to knowledge acquisition and integration: The participants in the supply chain must have a common understanding and vocabulary of the problem domain. Therefore, during the integration process, a common *ontology* of the domain has to be defined and the different product models of the suppliers have to be partially mapped. Within our framework we base knowledge acquisition and representation for the configuration domain on the usage of the Unified Modeling Language [7]. These graphical models serve as a general notation for modeling configuration problems and can be automatically compiled into the specific representations of existing configuration systems.

Finally, one goal for the design of the implementation framework is to minimize the requirements on the configuration systems at least for the leaf-levels of the supply chain: Basically these systems may not even implement a constraint-based algorithm, since the only requirement according to the algorithm is that – given some variables and values – they can compute at least one solution or report failure of finding a solution.

5. RELATED WORK

In recent years, some major advances in applying constraint technology in distributed environments (Distributed Constraint Satisfaction) have been made ([5],[9],[11]). The main goal of these approaches is to search for solutions in constraint networks that are distributed among several agents. Similar to our approach, the motivation does not lie in a speed-up of the search process through parallelization but is rather justified by the fact that the knowledge itself is distributed and there are e.g., security and confidentiality issues to be considered. Although the underlying algorithms are described in a way that one agent holds exactly one variable, these approaches can theoretically be generalized to cases where each agent can solve more complex local CSPs.
The main difference compared to our approach is that these algorithms work highly parallel (asynchronously), whereas our algorithm is – in the current version – purely synchronous. However, in the application domain of *product configuration* of telecommunication switches, parallel configuration of subassemblies is not desirable or possible because of the given structure of the configurable artifacts and the supply chains: it would not make sense to start configuring some subassembly before it is decided if this subassembly will be needed in the final product at all. Some small degree of parallelism, however, can be simply introduced in our algorithm, when we assume that a supplier will always find a solution given some requirements and that any solution of the supplier will not interfere with the configuration at the higher level.
Although the agents in Distributed CSP approaches work independently in general, some prioritization or ordering among the agents is needed in order to ensure soundness and completeness (of the backtracking mechanism). While this ordering is predefined in [5] or dynamically computed from the constraint graph [12], this hierarchy is given through the product structure in our approach in a natural way. Moreover, techniques like *Asynchronous Backtracking* rely on *no-good* recording which is both costly in terms of memory-consumption as well as problematic in terms of confidentiality because by exchanging *no-goods* knowledge about constraints is implicitly exchanged. While these approaches solve a more sophisticated algorithmic problem (allowing parallel computation), our approach tackles a specific real-world problem using existing commercial tools.
Future work on our approach will be to test the applicability of existing enhancements of the basic backtracking scheme (see e.g., [13]). More recent work [14] on intelligent backtracking takes advantage of the given structure of the constraint graph (bi-connected components) in order to compile partial solutions and jump back and forward during the search process more intelligently thus reducing the search space. These special constraint graph structures (containing loosely connected clusters)

make this approach applicable in our domain: Typically, only a small set of all constraints are defined between variables of different suppliers. Therefore, improvements like "reusing" former solutions provided by suppliers when we had to backtrack and the inputs for the suppliers did not change (compare to [14]) will be incorporated in future versions of the algorithm.

6. CONCLUSIONS

Product configuration of complex products and services that are assembled from parts provisioned by several suppliers in a supply chain are an important issue in today's and future eCommerce environments.
Based on the requirements of a real-world application scenario from the telecommunication domain, we have presented an algorithm for distributed configuration of such products in the supply chain. Relying on sharing of variables (and product models), the proposed algorithm is an extension of a standard forward-checking and backtracking algorithm for Constraint Satisfaction and was implemented by extending a commercially available tool for product configuration. We compared the algorithm with related work in the field and elaborated the specific demands for distributed problem solving in the domain of product configuration.

REFERENCES

[1] M. Stumptner. An overview of knowledge-based configuration, *AI Communications 10(2)*, 1997, 111-126.

[2] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction, *IEEE Intelligent Systems*, July/August, 1998.

[3] D. Mailharro. A Classification and Constraint-based Framework for Configuration, *AI EDAM, Vol. 12*, Cambridge University Press, 1998.

[4] E. Tsang, *Foundations of Constraint Satisfaction.* (Academic Press, 1993).

[5] M. Yokoo. *Distributed constraint satisfaction - foundations of cooperation in multi-agent systems* (Springer, Berlin, Germany, 2001).

[6] L. Ardissono, A. Felfernig, G .Friedrich, D. Jannach, R. Schaefer, M. Zanker. Intelligent Interfaces for Distributed Web-based Product and Service Configuration. *Proc. Web Intelligence (WI-2001),* Maebashi, Japan, *Lecture Notes in Artificial Intelligence*, Springer, October, 2001.

[7] A. Felfernig, G. Friedrich and D. Jannach. UML as domain-specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE), vol. 10 (4)*, 2000, 449-469.

[8] S. Mittal, B. Falkenhainer. Dynamic Constraint Satisfaction Problems. *Proceedings AAAI'90*, Boston, MA, 1990, 25-32

[9] A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, Towards Distributed Configuration. *Proc. KI-2001, Joint German/Austrian Conference on AI*, Vienna, Austria, Springer, September, 2001.

[10] U. Junker. Preference-programming for Configuration, *Proc. IJCAI'01 - Configuration Workshop*, Seattle, Wa, August, 2001.

[11] M. Silaghi, D. Sam-Haroud, B. Faltings. Asynchronous Search with Aggregations. *Proc. AAAI/IAAI 2000*, Austin, TX, 2000, 917-922.

[12] Y. Hamadi, C. Bessiere, J. Quinqueton. Backtracking in Distributed Constraint Networks, *Proc. ECAI'98*, Brighton, UK, John Wiley, 1998, 219-223.

[13] R. Dechter. Enhancement schemes for Constraint Processing: Backjumping, Learning, and Cutset decomposition. *Artificial Intelligence*, 41(3), Elsevier, 1990, 273-312.

[14] J. Baget, Y. Tognetti, Backtracking Through Biconnected Components of a Constraint Graph, *Proc. IJCAI'01*, AAAI Press. Seattle, 2001, 291-296.