Interactive Configuration Capability in a Sale Support System: Laziness and Focusing Mechanisms

Diego Magro and Pietro Torasso
Dipartimento di Informatica, Università di Torino
Corso Svizzera 185; 10149 Torino; Italy
{magro, torasso}@di.unito.it

Abstract

The paper discusses the configuration process in a sale support system aimed at helping a customer of a virtual store to assembly the right complex product containing the subcomponents that he/she chose and meeting her/his requirements. Because of the interaction of the customer with the system during the configuration process, we define a *lazy* configurator able to configure only the portion of the complex product which is directly related to the customer's requirements. We define also some focusing mechanisms able to use the constraints in order to reduce the search space during the configuration.

1 Introduction

In the last decade, much attention has been paid to the formalization of the configuration problem and different approaches have been proposed. Many of these are based on different extensions of the CSP framework (see, for example [Mittal and Falkenhainer, 1990]). A logical formalization of the configuration problem has been proposed by other researchers (e.g., in [Friedrich and Stumptner, 1999], a consistency-based definition of the configuration problem is given; in [Soininen *et al.*, 2000] a formalization of the configuration process as a problem of finding a stable model for a logical theory of *weight constraint rules* is presented).

The basic concepts underlying the configuration problem have been carefully analyzed (e.g. see [Tiihonen *et al.*, 1998] and [Felfernig *et al.*, 2000]) and the importance of representing the structure of the product (i.e. the partonomical knowledge) has been outlined by many researchers (the problem of representing the structure in a CSP framework is treated, for example, in [Sabin and Freuder, 1996] and in [Veron and Aldanondo, 2000]). However, fewer works have analyzed the issue of the interaction between the configurator and the user (among them, [Amilhastre and Fargier, 2000], [Gelle and Weigel, 1996], and [Veron and Aldanondo, 2000]).

In the present paper we are mainly interested in the interactive aspect of the configuration process and, in particular, we consider a scenario where an intelligent agent has to support a customer in selecting a **complex product** which satisfies the a set of requirements by taking into account the **atomic products** present in the catalogue of a virtual store (in the following e-catalogue). Some products are considered *atomic*, since they are sold "as they are" and thus their internal structure (which could be quite complex) is of no interest for this task (but could be relevant for other tasks such as repair or manifacturing). A complex product can be viewed as a structured entity whose subparts can be complex products in their turn or atomic ones. So, eventually, it can be considered as the result of an assembly of atomic products. Since it is impossible to foresee a-priori the huge number of different variants of a complex product needed for matching the different requirements that a customer could impose, any specific variant of a complex product satisfying the customer requirements must be built "on the fly" by assembling a set of atomic components (if possible). The tasks of checking the consistency of a set of requirements and of building a product that meets a (consistent) set of requirements can be very difficult even for domain experts. In a virtual store it is not possible to make strong assumptions on the expertise of the customer, so that an intelligent agent has not only to support the customer in configuring the complex product but also in guiding her/him in the specification of the problem. In this paper we refer to the following scheme of interaction between the customer and the sale support system (the parts written in italic are relevant to the customer, while the other ones refer to the system):

1. *The customer selects a (gross) category of products and states a set of requirements*.
2. The portion of the e-catalogue relevant to the chosen category of products is displayed to the customer.
3. *The customer selects a set of (atomic) components from the e-catalogue*.
4. A subset of (complex) products in the chosen category is presented to the customer, on the basis of the (atomic) components she/he has chosen.
5. *The customer selects one product among those the system has presented her/him.*
6. A consistency check is performed in order to verify if it is possible to build one instance of the chosen product containing (at least) the set of chosen (atomic) components and meeting the desired requirements. If no product can be built with the desired features, the inconsistency is notified to the customer. Otherwise, a partial configuration of the desired product is shown to the customer.
7. *In case of inconsistency, the customer can go back to steps 1, 3 or 5 and revise her/his choices. In case of consis-*

tency, she/he can ask the system for an alternative (partial) configuration (if possible) or she/he can ask the system for a support in extending the proposed partial configuration. The customer can also decide to stop the interaction with the system.

8. The system builds an alternative (partial) configuration or it extends the one that it proposed to the customer (if this is incomplete). In the first case, the interaction goes back to previous step (with a consistent situation). In the second case, the portion of the e-catalogue relevant to an incomplete subcomponent of the current configuration (possibly chosen by the customer) is presented to the customer. Then the interaction goes back to step 3 (and it skips the steps 4 and 5).

As can be seen, in this scenario the intelligent selling agent has a strong interaction with the customer and advises her/him in the different steps. Since a customer could be unaware of the particular product that fits her/his needs, the system does not force the customer to single out a specific complex product, but on the basis of a set of requirements and a generic indication of the category of product the customer is interested in (step 1) and of the selected (atomic) components (steps 2 and 3), the system is able to make a set of hypotheses on the products actually needed by the customer (step 4), among which the customer can choose one (step 5). This functionality is realized via an inference mechanism presented in [Magro and Torasso, 2000] (i.e. the *Hypothesis Formulation Algorithm*) and it will not be further analyzed in this paper. It is worth noting that the system disregards some of these steps (in particular steps 4 and 5) in case a more knowledgeable customer has been able to single out the specific product he/she needs from the real beginning. The step 6 can be seen as a configuration process, but the system should not be too eager to complete the work (configuring the entire complex product), since the customer has to be involved in the configuration process by leaving her/him the possibility of extending a partial configuration or selecting among competing hypotheses. In this way, the configuration module has the only task of verifying that the customer requirements are consistent with the complex product description and, in this case, it should propose a partial configuration satisfying them (and containing the atomic components chosen so far by the customer). In other words, the system should be *lazy* and it should come back to the customer as soon as it can. Moreover, because of the strict interaction between the custumer and the system, the time spent by the system in configuring is critical. For this reason, we have developed a set of focusing mechanisms to reduce the computational effort without loosing any solution.

In the rest of the paper we will analyze the impact of this scenario on the configuration process. Particular attention will be given to the problem of configuring only the portion of the complex product the customer is interested in or has made some explicit choices.

The approach presented here is tested by developing (in Java) a prototype of the sale support system and the domain of PC has been used as a test bed.

2 The Conceptual Model and the Configuration Algorithm: Overview

The configuration algorithm, in its basic structure, is similar to the *hypothesis validation algorithm* presented in [Magro and Torasso, 2000] and it works on a conceptual representation of the domain expressed in the $\mathcal{FPC}$ language described in the same paper. In the following only a brief overview of the conceptual language and the configuration algorithm is provided.

The Conceptual Model. The conceptual model of a domain is made up of two different kinds of entities: atomic products and complex ones. Both atomic and complex products can be organized around taxonomies and are described by a set of properties (inherited along the tassonomic links and represented by means of *descriptive slots* associated to the class) and by a set of *constraints*. The structure of a complex product is modeled via whole-part relationships between it and each of its subparts; in particular each class of complex products has a set of *partonomic slots* to capture these relationships.

Any partonomic slot p of a class C of complex products relating C to another class D (either of complex or of atomic products) can be considered as a link directed from C to D. The cardinality associated to the partonomic slot p is represented by a pair of non-negative integers (indicated with $C_{min}(C, \langle p \rangle)$ and $C_{max}(C, \langle p \rangle)$). Formally, p is interpreted as a relation between C and D and it denotes that $(\forall c \in C)(p(c) \subseteq D \wedge C_{min}(C, \langle p \rangle) \leq |p(c)| \leq C_{max}(C, , \langle p \rangle))$ The meaning is straightforward: each complex product of type C has from a minimum of $C_{min}(C, \langle p \rangle)$ up to a maximum of $C_{max}(C, \langle p \rangle)$ subparts of type D via an has-part relation named p. The codomain D of this relation is indicated with $cod(C, \langle p \rangle)$. The other properties of the product are expressed by means of *descriptive slots* and they are formally interpreted as relations too. A path of links in a partonomy is called *slot chain*. A slot chain $\gamma = \langle p_1, \ldots, p_n \rangle (n \geq 0)$ (the empty chain is indicated with $\langle\rangle$), starting in a class C is formally interpreted as the relation composition $p_n \circ p_{n-1} \circ \ldots \circ p_1$. If p_n is a partonomic slot, the chain represents the subcomponents of a complex product $c \in C$ via the *has-part* relationships named $p_1, \ldots, p_n$. If p_n is a descriptive slot, this chain represents the values of the property p_n for the subcomponents of c via the *has-part* relationships $p_1, \ldots, p_{n-1}$. Similarly, a set of slot chains $R = \{\gamma_1, \ldots, \gamma_m\} (m \geq 1)$ (each one starting in C) is interpreted as the relation union $\bigcup_{i=1}^{m} \gamma_i$. $\{\langle\rangle\}$ is interpreted as the identity function. With $C_{min}(C, R)$ and $C_{max}(C, R)$ we indicate, respectively, the lower bound and the upper bound of the cardinality of the set $R(c)$ (for each $c \in C$) as they can be computed only on the basis of the taxo-partonomical portion of the conceptual model (i.e. the sets of constraints associated to each class of complex products are not considered). We assume $C_{min}(C, \{\langle\rangle\}) = C_{max}(C, \{\langle\rangle\}) = 1$. The codomain of R is indicated with $cod(C, R)$. We have $cod(C, \{\langle\rangle\}) = C$. If $\gamma \neq \langle\rangle$, with $Prefs(\gamma)$ we indicate all the subchains of γ starting with the slot p_1. With $Suffs(\gamma)$ we indicate all the subchains of γ ending with p_n (if $n \geq 1$, we have $\langle\rangle \notin Prefs(\gamma) \cup Suffs(\gamma)$, while $Prefs(\langle\rangle) =$

$Suffs(\langle\rangle) = \{\langle\rangle\}$). If $\delta = \langle q_1, \ldots, q_k\rangle (k \geq 0)$ is a slot chain starting in $cod(C, \{\gamma\})$, we indicate with $\bullet$ the justapposition operator: $\gamma \bullet \delta = \langle p_1, \ldots, p_n, q_1, \ldots, q_k\rangle$ ($\langle\rangle$ is the neutral element w.r.t. this operator).

Any *configuration* is represented as a tree. The root represents the complex product and each node n can represent a subcomponent or a property value (in the latter case, it must be a leaf). In the former case, with $class(n)$ we denote the the most specific class (w.r.t. the taxonomy) to which n belongs. Each arc starts in a component node n. If it ends in another component node m, it represents a *has-part* relationship between n and m and it is labelled with the corresponding partonomic slot. If it ends in a property node it is labelled with the corresponding descriptive slot and it represents that property. Given any node n in a configuration, the path from the root to n defines a slot chain: we denote it with $path(n)$ (obviously, $path(root) = \langle\rangle$).

As we said, a set (possibly empty) of *constraints* is associated to each class of complex products. These constraints restrict the set of valid combinations of components and subcomponents for a given complex product type. Due to their importance both in realizing the *laziness* and in the focusing mechanisms, we define (a simplified version of) the constraint language.

Let $R = \{\gamma_1, \ldots, \gamma_m\} (m \geq 1)$, where each $\gamma_i = \langle p_{i_1}, \ldots p_{i_n}\rangle (i_n \geq 1)$ is a slot chain starting in C.

The basic building blocks of the constraint language are the predicates. In the conceptual language, there are six different kinds of predicates. Here, due to space limitations, we present (in a simplified version) only those three types that are heavily used in the focusing mechanisms:

1) $(R)(h, k)$. $c \in C$ satisfies the predicate *iff* $h \leq |R(c)| \leq k$, where h, k are non negative integers with $h \leq k$ and $R(c)$ is a set of components.

2) $(R)(inI)$. $I = I_1 \cup \ldots \cup I_s (s \geq 1)$ and each $I_j (j = 1, \ldots, s)$ is a class. $c \in C$ satisfies the predicate *iff* $R(c) \subseteq I$. $R(c)$ can be a set of components or a set of property values.

3) $(R)(inI(h, k))$. $I = I_1 \cup \ldots \cup I_s$ and each $I_j (j = 1, \ldots, s)$ is a class. $c \in C$ satisfies the predicate *iff* $h \leq |R(c) \cap I| \leq k$, where h, k are non negative integers with $h \leq k$ and $R(c)$ is a set of components.

Each constraint cc associated to C is of the form $\alpha \Rightarrow \beta$, where α is a conjunction of predicates or the boolean constant **true** and β is a predicate or the boolean constant **false**. The meaning is that for every complex product $c \in C$, if c satisfies α then it must satisfy β. It should be clear that if $\alpha = \mathbf{true}$, then, for each $c \in C$, β must always hold, while if $\beta = \mathbf{false}$, then, for each $c \in C$, α can never hold. The constraint $\mathbf{true} \Rightarrow \mathbf{false}$ is forbidden.

If v is either a predicate or a constraint, with $chains(v)$ we indicate the set of slot chains occurring in v.

The Configuration Algorithm. The requirements stated by the customer (see the step 1 in the scenario described in section 1) and the set of atomic components that she/he has chosen (see the step 3) are translated into a set $CONSTR_I$ of input constraints (expressed in the costraint conceptual language). The configuration algorithm accepts in input the conceptual model of the domain, a class C of complex products and the set $CONSTR_I$. C can represent either the chosen type of complex product (see the steps 1 and 5) or the type of the incomplete subcomponent chosen to expand the current partial configuration (see the step 8). $CONSTR_I$ is added to the set of constraints associated to the class C. The configuration process starts with a configuration (tree) containing only the root c representing a product of type C and it tries to expand this configuration until all the considered constraints are satisfied. If it succeeds, it returns the produced (partial) configuration, otherwise, it returns a failure message representing the inconsistence of the requirements $CONSTR_I$ (see the step 6).

The expansion of the configuration is realized by means of a search process with a backtracking mechanism. The main steps of the algorithm are the following:

a. A subcomponent n that must still be expanded is chosen (starting with the root c);

b. The set $C_c(n)$ of the current constraints for n is computed;

c. On the basis of $C_c(n)$, the set $S_c(n)$ of the current partonomic slots is computed;

The following steps d-g are repeated until $S_c(n)$ becomes empty:

d. A slot $p \in S_c(n)$ is selected;

e. The cardinality of p (i.e. the number of the new subcomponents that should be introduced) is chosen and the new subcomponents are added to the configuration;

f. A type for each new subcomponent of n is chosen;

g. If any constraint in $C_c(n)$ is violated a backtracking occurs. Otherwise, the satisfied constraints are deleted from $C_c(n)$ and $S_c(n)$ is updated.

These steps are repeated until all the considered constraints are satisfied or the algorithm fails in producing any configuration.

Since this algorithm is used in an interactive way (see the scenario in section 1), it shouldn't be "too autonomous" and, given a set of requirements, it should introduce in the configuration only those components related in some way to the requirements (*laziness*). Moreover, some *focusing mechanisms* are needed to speed-up the search in the configuration process.

3 Laziness

If we assume the consistency of the conceptual model, we are guaranteed that at least one product of the considered type C can somehow be built if $CONSTR_I = \emptyset$. However, the interesting case is when $CONSTR_I \neq \emptyset$ (the customer has put some requirements) and therefore the input constraints can interact both with the taxo-partonomical portion of the conceptual model of C and with the constraints associated to C. This interaction can reduce the set of valid configurations. If this set is reduced to the empty one, we say that the input constraints $CONSTR_I$ are inconsistent. The *laziness mechanisms* allow the sale support system to reduce the work needed for determining a (partial) valid configuration or for detecting the inconsistency. More precisely: *the configuration process is* **lazy** *iff each partonomic slot p selected in step d is such that the choices made for it (steps e and f) can be critical for the satisfaction of some input constraint*.

The laziness is mainly realized by means of the following mechanisms.

1. *Bound Relation* among constraints. Two constraints are *bound* if the choices made during the configuration process in order to satisfy one of them *can* interact with the set of possible choices for the second one [1]. The bound relation $\mathcal{B}_C$ is defined as follows: let u, v and w be three constraints for a class C; **if** $(\exists \alpha \in chains(u))(\exists \beta \in chains(v))(\alpha = \langle p, q_1, \ldots, q_n \rangle \wedge \beta = \langle p, r_1, \ldots, r_m \rangle \wedge n \geq 0 \wedge m \geq 0)$ **then** $u \mathcal{B}_C v$ (i.e. if two constraints refer to the same subparts, they are bound); **if** $u \mathcal{B}_C v \wedge v \mathcal{B}_C w$ **then** $u \mathcal{B}_C w$ (transitivity). It is easy to show that $\mathcal{B}_C$ is an equivalence relation. If V_C is a set of constraints for C, we indicate the quotient set with $V_C/\mathcal{B}_C$.

2. Computation of the initial set $C_c(n)$ of current constraints for the selected component n. (step b in the algorithm). The set of current constraints for a component influences the set of partonomic slots that have to be considered for that component (see the item 3) and thus it has a direct impact on the number of subcomponents of n that should be taken into consideration during the configuration process. Usually, the smaller $C_c(n)$ the smaller the set of partonomic slots of n that are considered during the configuration. For this reason we restrict $C_c(n)$ to contain only those constraints related in some way to the input ones. Let $CONSTR$ be the set of constraints associated to $class(n)$ in the conceptual model. $C_c(n)$ is computed as the union of two disjoint sets: that of *local constraints* $LC_c(n)$ and that of *inherited constraints* $IC_c(n)$. n is *the origin* of each constraint in $v \in LC_c(n)$ and this is expressed by the equation $origin(v) = n$ (each predicate occurring in a constraint has the same *origin* as the constraint).

2.a. If n is the root: $IC_c(n) := \emptyset$; $LC_c(n) := \bigcup_{V \in [(CONSTR \cup CONSTR_I)/\mathcal{B}_{class(n)}] \wedge V \cap CONSTR_I \neq \emptyset} V$.
In this case, the input constraints ($CONSTR_I$) are added to the constraints associated to the considered class in the conceptual model ($CONSTR$). $CONSTR \cup CONSTR_I$ is partitioned by means of the *bound relation* $\mathcal{B}_{class(n)}$ and only those equivalence classes of constraints containing at least one input constraint are considered in the evaluation of $C_c(n)$.

2.b. If n is an internal node: let m be the parent node of n in the configuration and S be the set of partonomic slots associated to $class(n)$ in the conceptual model. The considered component n inherits all the current constraints of m that refer to some subcomponents of n: $IC_c(n) := \{v \in C_c(m) : (\exists \gamma \in chains(v))(\exists r \in S)(Prefs(\gamma) \cap Suffs(path(n) \bullet \langle r \rangle) \neq \emptyset)\}$;
If a constraint v in $CONSTR$ refers to the same subcomponents of n as any inherited constraint v_{in} (i.e. a constraint in $IC_c(n)$), then v and all the constraints bound to v have to be considered. Formally, $LC_c(n) := \bigcup_{V \in I} V$, where $I = \{V \in CONSTR/\mathcal{B}_{class(n)} : (\exists v \in V)(\exists v_{in} \in IC_c(n))(\exists \gamma \in chains(v))(\exists \gamma_{in} \in chains(v_{in}))(\gamma = \langle r, p_1, \ldots, p_n \rangle \wedge n \geq 0 \wedge Prefs(\gamma_{in}) \cap Suffs(path(n) \bullet \langle r \rangle) \neq \emptyset))\}$.

3. Computation of the initial set $S_c(n)$ of current slots. (step c in the algorithm).

All and only those partonomic slots mentioned in any constraint either locally (i.e. in $LC_c(n)$) or by inheritance (i.e. in $IC_c(n)$) are taken into consideration. Formally, $S_c(n) := \{r \in S : ((\exists v \in LC_c(n))(\exists \gamma \in chains(v))(\gamma = \langle r, p_1, \ldots, p_n \rangle) \wedge n \geq 0) \vee ((\exists v \in IC_c(n))(\exists \gamma \in chains(v))(Prefs(v) \cap Suffs(path(n) \bullet \langle r \rangle) \neq \emptyset))\}$ where S is the set of partonomic slots associated to $class(n)$ in the conceptual model.

4. Updating of the current set $S_c(n)$ of slots. (step g in the algorithm). Let $\overline{S}$ be the set of slots computed as described in the previous item, on the basis of the *updated* set $C_c(n)$. $S_c(n)$ is computed after updating the set $C_c(n)$. Any slot in $S_c(n)$ has its justification in a current constraint, i.e. it must belong to $\overline{S}$. Moreover, a slot should not be considered more that once (except in case of backtracking), thus, any slot in the updated set of slots must belong to the old set of slot and be different from the last considered slot p: $S_c(n) := (S_c(n) - \{p\}) \cap \overline{S}$, where p is the last considered slot (step d).

The mechanisms described so far allows the system to subdivide the constraints into independent groups on the basis of the equivalence classes and the input constraints. In this way the global configuration problem can be split into a number of independent configuration subproblems and the configuration algorithm exploits such a possibility since it works separately on each class of constraints $V \in (CONSTR \cup CONSTR_I)/\mathcal{B}_C$. In this way, some useless backtrackings can be easily avoided. In fact, let's suppose that during the configuration process a constraint v is violated. If the introduction of v in any initial set $C_c(n)$ had its justification (see the items 2.a and 2.b) - either direct or indirect - in a constraint belonging to $V \in (CONSTR \cup CONSTR_I)/\mathcal{B}_C$, it should be clear that it would be useless to revise any choice made for any partonomic slot p whose introduction in any initial set $S_c(m)$ (item 3) had its justification in a constraint belonging to another class $U \in (CONSTR \cup CONSTR_I)/\mathcal{B}_C$: We have the guarantee that this does not happen since the classes in $(CONSTR \cup CONSTR_I)/\mathcal{B}_C$ are considered separately by the configuration algorithm. The subdivision of constraints into independent classes has also the advantage of providing the customer with much more information when an inconsistency is detected. Let us suppose that such an inconsistency is detected while the configuration algorithm is considering a class $V \in (CONSTR \cup CONSTR_I)/\mathcal{B}_C$: the system can notify the customer that he/she has to revise his/her choices and/or requirements by pointing out only the subset of input constraints that are involved in the the class V. This mechanism allows the customer to focus his/her attention on a specific portion of the domain knowledge. Moreover, the subdivision of constraints into independent classes allows the configuration system to re-use part of work already done: in fact, in case of change of some requirement and/or choice (see steps 6 and 7 in the scenario) only some of the classes of constraints are interested in the change. Therefore the portion of the configuration built while considering the(consistent) classes of constraints not involved in the

[1] Here we are referring to the two main choices made during the search process by the configuration algorithm: that of the cardinality of a partonomic slot p (step e) and that of the type for each new subcomponent (step f).

change can be re-used without any additional computation or check.

4 Focusing Mechanisms

Let n be the selected component in a partial configuration $\mathcal{CONF}$ (step a in the configuration algorithm) and p be the current slot (step d). $ANT\_SAT(n)$ is the set of current constraints for n whose antecedents are satisfied in $\mathcal{CONF}$. $CONS_i(n)$ ($i = 1, 2, 3.$) is the set of current constraints for n whose consequent is a predicate of type i (see the three types of predicates presented in Section 2). $FOR\_p(n) = \{\alpha \Rightarrow \beta \in C_c(n) : (\exists \gamma \in chains(\beta))(Prefs(\gamma) \cap Suffs(path(n) \bullet \langle p \rangle) \neq \emptyset)\}$ is the set of current constraints for n whose consequent mentions the slot p of n.

Focusing the Choice of a Cardinality for the Current Slot. (step e in the configuration algorithm). Initially, the possible choices for the cardinality of p are those specified in the taxo-partonomical description of $class(n)$, namely $CARDs = [C_{min}(class(n), \langle p \rangle), C_{max}(class(n), \langle p \rangle)]$. In order to reduce (hopefully) this set, the constraints in $ANT\_SAT(n) \cap CONS_1(n)$ are used.[2] In fact, for each constraint in $ANT\_SAT(n) \cap CONS_1(n)$, its consequent must hold, thus any cardinality in $CARDs$ that would prevent this consequent to become true can be discarded without loosing any solution. Obviously, a choice made for p can only influence the truth value of a consequent that mentions the slot p of the chosen component n. Thus, the set of predicates $FOCUS1 = \{\beta : \exists(\alpha \Rightarrow \beta) \in ANT\_SAT(n) \cap CONS_1(n) \cap FOR\_p(n)\}$ is used to (try to) reduce the set $CARDs$ (i.e. to focus the search process). $CARDs$ is iteratively updated *for each* $\beta \equiv (R)(h, k) \in FOCUS1$ according to the following criterion: $CARDs := CARDs \cap [cm_p, CM_p]$ where cm_p is a lower bound for the cardinalities that can be assigned to p without preventing the inequality $|R(origin(\beta))| \geq h$ to hold (in at least one extension of $\mathcal{CONF}$). cm_p is computed on the basis of the current configuration $\mathcal{CONF}$ and of the taxo-partonomical portion of the conceptual model without resorting to the constraints. CM_p is an upper bound for the cardinalties which takes into consideration the inequality $|R(origin(\beta))| \leq k$.

If, after the update, $CARDs$ becomes empty, a backtracking occurs.

Focusing the Choice of the Type for each New Subcomponent. (step f of the configuration algorithm). Let $SUBS\_np$ be the set of subcomponents of n (in $\mathcal{CONF}$) through the has-part relation expressed by p and $NEW\_S(\subseteq SUBS\_np)$ be the set of these subcomponents whose type (i.e. class) has not been chosen yet. At the beginning of step f, we have $SUBS\_np = NEW\_S$) and the possible types $CLASSES$ for each subcomponent in $SUBS\_np$ are those specified in the taxo-partonomical description of $class(n)$, namely all the most specific subclasses of $cod(class(n), \langle p \rangle)$ in the taxonomy [3].

The constraints in $ANT\_SAT(n) \cap CONS_2(n)$ and in $ANT\_SAT(n) \cap CONS_3(n)$ are used to (hopefully) reduce this set. In the following we provide some detail how predicates of Type 2 are used (predicates of type 3 are used in a similar way).

Let $F = \{\beta : \exists(\alpha \Rightarrow \beta) \in ANT\_SAT(n) \cap CONS_2(n)\}$.

Each predicate $\beta \equiv (R)(inI)$ in F entails the set of predicates (still of type 2) $One\_chain(\beta) = \{(\gamma)(inJ) : \gamma \in R \wedge J = cod(class(origin(\beta)), \{\gamma\}) \cap I\}$ (possibly $J = \emptyset$). The set of predicates $FOCUS2 = \{\delta \equiv (\gamma)(inJ) : (\exists \beta \in F)(\delta \in One\_chain(\beta) \wedge Prefs(\gamma) \cap Suffs(path(n) \bullet \langle p \rangle) \neq \emptyset)\}$ are used as focusing predicates. $FOCUS2$ is the set of predicates entailed by the predicates in F and containing only one slot chain in which the slot p occurs. It should be clear that, given $\mathcal{CONF}$, each predicate in $FOCUS2$ must hold and that its truth value may be influenced by the choices made for the type of the subcomponents in $SUBS\_np$. Therefore, each class in $CLASS$ that, if chosen as a type for any subcomponent in $SUBS\_np$, would prevent one or more predicate in $FOCUS2$ to hold can be discarded without loosing any solution. Thus, at the beginning of step f (i.e. when no class has been assigned to any subcomponent in $SUBS\_np$ yet), each predicate in $FOCUS2$ is considered and $CLASSES$ is iteratively updated in the following way:

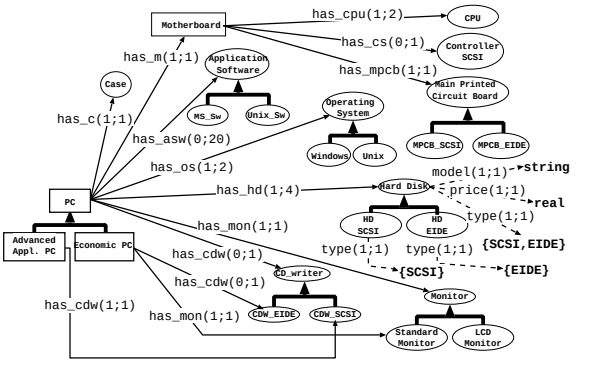
for each $\delta \equiv (\langle p_1, \ldots, p_n, p, p_{i_1}, \ldots, p_{i_m} \rangle)(inJ) \in FOCUS2$, the set $CLS1 = \{C \in CLASSES : C_{min}(C, \{\langle p_{i_1}, \ldots, p_{i_m} \rangle\}) = 0 \vee cod(C, \langle p_{i_1}, \ldots, p_{i_m} \rangle) \cap J \neq \emptyset\}$ is computed and it it assigned to $CLASSES$ (if $CLS1 = \emptyset$, a backtracking occurs). Let's suppose that there is a $D \in CLASSES$ such that $C_{min}(D, \{\langle p_{i_1}, \ldots, p_{i_m} \rangle\}) \neq 0 \wedge cod(D, \langle p_{i_1}, \ldots, p_{i_m} \rangle) \cap J = \emptyset$. This means that for each $d \in D$, the set $(p_{i_m} \circ \ldots \circ p_{i_1})(d)$ has at least one element *that can't belong to J*. The choice of D as a class for a subcomponent $s \in SUBS\_np$ would prevent the considered predicate $(\langle p_1, \ldots, p_n, p, p_{i_1}, \ldots, p_{i_m} \rangle)(inJ)$ to hold no matter what classes are chosen for the other subcomponents in $SUBS\_np - \{s\}$. D can thus be discarded and this is exactly what the focusing mechanism does.

In order to test the effectiveness of the focusing mechanisms we performed some preliminary experiments over a conceptual model relevant to a (simplified) PC domain. We tested the configuration capability of the system by means of 50 different test cases. Each case represented an interaction with the system in which the customer selected a specific complex product C and a set of atomic components. For each case, we measured the effectiveness of the focusing mechanisms by comparing the number of bactrackings occurred when building a (partial) configuration of the product C (see the step 6 of the scenario) without using the focusing mechanisms with the same task performed by using such mechanisms.

It resulted that, by exploiting the focusing mechanisms, the number of backtracking is about 25% of that one occurred with the strategy that didn't make use of the focusing mecha-

[2] Since any predicate of type 3 $(R)(inI(h, k))$ entails the predicate of type 1 $(R)(h, C_{max}(class(n), R))$, also the predicates of type 3 are actually used by this focusing mechanism.

[3] For the sake of simplicity, we assume that the set of direct subclasses of each class in the conceptual model is a partition of that class and that the type of each (sub)component in a configuration always is a most specific class w.r.t. the taxonomy.

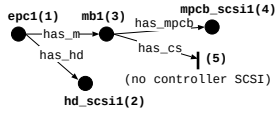



```
[co1] ({<has_asw>})(in Unix_Sw (1;20)) ==> ({<has_os>})(in Unix (1;2))
      (In any PC, if there is a Unix application software,
       then there has to be a Unix operating system)
[co2] ({<has_hd>})(in HD_SCSI (1;4))∧({<has_m,has_mpcb>})(in MPCB_EIDE)==>
      ({<has_m,has_cs>})(1;1) (In any PC, if there is a SCSI hard disk and
       an EIDE main printed circuit board, then a controller SCSI is needed)
[co3] (...) (Analogous to the previous one, for CD writer)
[co4] ({<has_mpcb>})(in MPCB_SCSI) ==> ({<has_cs>})(0;0) (In any motherboard,
       if there is a SCSI main printed circuit board,
       then there can't be any additional controller SCSI)
```

Figure 1: A sample (portion of a) conceptual model

nisms. These results are encouraging, even if a more accurate statistical analysis has still to be performed.

5 An Example




```
[co_ch1]true ==> ({<has_m,has_mpcb>})(in MPCB_SCSI(1;1))
[co_ch2]true ==> ({<has_hd>})(in HD_SCSI(1;4))
```

Figure 2: A partial configuration

Figure 1 contains a small simplified fragment of a conceptual model relevant to the PC domain. Each rectangle represents a class of complex products and each oval represents a class of atomic products; any thin solid arrow corresponds to a partonomic slot whilst each thin dashed arrow corresponds to a descriptive slot. The subclass links are represented by thick solid arrows. In the figure, it is stated, for example, that each PC contains exactly one motherboard (which is a complex product), from one to four hard disks, an optional CD writer and so on; that any hard disk is described by its model (denoted by a string), its price (expressed as a real number), its type (either SCSI or EIDE), etc; that an Advanced Appl. PC is a particular PC having exacly one SCSI CD writer and so on. Moreover, let's suppose that in the model there are only the constraints from $co1$ to $co4$ shown in figure 1 (those from $co1$ to $co3$ are associated to the PC class, while $co4$ is associated to the Motherboard class).

Let's suppose that the customer wants an Economic PC (in the following EPC) with a SCSI Hard Disk and a SCSI Main Printed Circuit Board and that no other requirements are imposed to the final product. We want to see how the configuration algorithm works in order to check the consistency of the requirements (step 6 of the scenario in section 1).

First of all, we note that the EPC class inherits all the PC class constraints, but $co3$ (let V_{EPC} be this set of constraints), since this one is trivially entailed by the taxo-partonomical description of the EPC class (the constraint inheritance is described in [Magro and Torasso, 2000]). It is easy to see that $V_{EPC}/\mathcal{B}_{EPC} = \{\{co1\}, \{co2\}\}$ (i.e. $co1$ and $co2$ are unbound). The set of chosen atomic products is translated into the set of constraints $CONSTR_I = \{co\_ch1, co\_ch2\}$ for the EPC class (figure 2. See [Magro and Torasso, 2000] for a description of such a translation).

In figure 2 the numbers next to the component nodes express the order in which the components have been introduced into the configuration (we selected one order among the set of possible ones). The process starts with the root (component 1), representing an instance $epc1$ of the EPC class (i.e. the target product). We have $(V_{EPC} \cup CONSTR_I)/\mathcal{B}_{EPC} = \{\{co2, co\_ch1, co\_ch2\}, \{co1\}\}$, thus $C_c(1) = LC_c(1) = \{co2, co\_ch1, co\_ch2\}$; $S_c(1) = \{has\_hd, has\_m\}$. Let's suppose that the slot has_hd is firstly considered. The possible cardinalities for has_hd are all those specified in the taxo-partonomy (i.e. from 1 to 4), since the current constraints doesn't produce any restriction of this interval. Let's suppose that the minimum cardinality is chosen, producing the introduction of the component 2, for which the only admissible type is HD_SCSI (this choice has been focused by means of the constraint co_ch2 – whose consequent is of type 3 – in a way similar to that presented in section 4). After introducing the motherboard $mb1$, we have $C_c(1) = \{co2, co\_ch1\}$ and $S_c(1) = \emptyset$. At this point, the component 3 (i.e. the motherboard) is selected for expansion. Since the constraints $co2$ and co_ch1 both mention some slots of the Motherboard class (i.e. has_mpcb and has_cs), they are inherited by the component 3: $IC_c(3) = \{co2, co\_ch1\}$. Since $co4$ mentions some slots that are mentioned also in some inherited constraints (e.g. has_mpcb), it has to be taken into consideration: $LC_c(3) = \{co4\}$. Thus, we have $C_c(3) = \{co2, co\_ch1, co4\}$ and, consequently, $S_c(3) = \{has\_mpcb, has\_cs\}$. Let's suppose that the slot has_mpcb is firstly considered and that a Main Printed Circuit Board is introduced into the configuration (component 3). The consequent of the constraint co_ch1 (which is a predicate of type 3) is used to focus the choice of the type for the component 3: on the basis of this predicate, the only admissible type is $MPCB\_SCSI$. At this point, we have $C_c(3) = \{co4\}$ and $S_c(3) = \{has\_cs\}$. The choice of the cardinality for has_cs is focused by means of the consequent of the constraint $co4$ and the only admissible cardinality is zero (thus no additional controller SCSI is introduced). Finally, $C_c(3) = \emptyset$ and $S_c(3) = \emptyset$. All the considered constraints are satisfied (no other complex component has to be expanded), thus the configuration process halts returning the partial configuration shown in figure 2 (this demonstrates the consistency of the customer's requirements).

It should be clear that only the components involved in the satisfaction of the input constraints have been introduced into the configuration and any choice involving, say, the software or the CPU can be taken in co-operation with the customer. Moreover, it is worth noting that if the customer had selected also a software product, the configuration problem

would have been split into two independent subproblems: the one presented in this example and another one involving only the new input constraint and the constraint $co1$.

6 Discussion

In this paper we have described some aspects of a sale support system aimed at helping a customer of a virtual store in selecting a complex product (or only a set of atomic components for a complex product). In particular, we analyzed the configuration capability, that plays a central role in such systems. We pointed out the advantages of using, in this context, a *lazy* configuration mechanism. In fact, the system should propose to the customer a partial configuration satisfying the current requirements (if they are consistent) and containing only those subcomponents related in some way to these requirements. In this way the customer can interact with the configuration process by stating a new set of requirements referring to those subparts of the product for which the system didn't make any proposal.

Some mechanisms that realize the laziness and other ones that aim at reducing the problem solving time by focusing the search during the configuration process have been described.

There are some differences between our application domain of configuration and more technical ones. First of all, we can't assume that the user is an expert of the domain. This fact means also that the customer could not be aware of all the requirements from the beginning of her/his interaction with the system

The main aim of the sale support system is not to inform the customer on how to actually connect the components that she/he bought, but to support her/him in buying the correct set of components (and to make a meaningful order to the virtual store owner).

Despite these differences, some similarities still exist. The need of having a structured representation of the products, and, in particular, to express both the taxonomical and the partonomical knowledge, proved to be fundamental also for our task. In [McGuinness and Wright, 1998] the domain knowledge is represented by means of description logic. Formally, the semantics of our representation language is strongly inspired by description logic languages.

In many cases, the product models built by the modelers are translated into a lower level representation language used by the configurator, e.g. in a logical theory or in some kind of CSP (for an automated translation of a conceptual model expressed in UML into a lower level logical representation language, see [Felfernig *et al.*, 2000]). This approach has the advantage that the well-known algorithms developed for those lower level languages can be used and, in this way, a formal analysis of the configuration process is made possible.

Instead, we defined the configuration mechanisms that work directly at the conceptual level. In this way, there is no need of translating the conceptual model into a lower level representation language and the results of the configuration can be easily presented to the customer.

In [Amilhastre and Fargier, 2000] an off-line precomputation of all the solutions of a configuration problem (represented as a CSP) is suggested. These solutions are represented by means of an automaton which is used at runtime to interactively solve the configuration problem in an efficient way.

Differently from this approach, our configuration mechanisms solve each problem at runtime and try to reduce the computational effort by subdividing the main problem into a set of independent sub-problems (through the *bound relation* presented in the section 3), by adopting a *lazy policy* and by making use of some *focusing mechanisms*.

References

[Amilhastre and Fargier, 2000] J. Amilhastre and H. Fargier. Handling interactivity in a constraint-based approach of configuration. In *Proc. of the Configuration Workshop held at the ECAI 2000*, pages 7–12, 2000.

[Felfernig *et al.*, 2000] A. Felfernig, G. E. Friedrich, and D. Jannach. Uml as domain specific language for the construction of knowledge-based configuration systems. *Int. J. of Software Engineering and Knowledge Engineering*, 10(4):449–469, 2000.

[Friedrich and Stumptner, 1999] G. Friedrich and M. Stumptner. Consistency-based configuration. In *AAAI-99, Workshop on Configuration*, 1999.

[Gelle and Weigel, 1996] E. Gelle and R. Weigel. Interactive configuration using constraint satisfaction techniques. In *Second International Conference on Practical Application of Constraint Technology, PACT-96*, pages 57–72, 1996.

[Magro and Torasso, 2000] D. Magro and P. Torasso. Description and configuration of complex technical products in a virtual store. In *Proc. of the Configuration Workshop held at the ECAI 2000*, pages 50–55, 2000.

[McGuinness and Wright, 1998] D. L. McGuinness and J. R. Wright. An industrial-strength description logic-based configurator platform. *IEEE Intelligent Systems*, (July/August 1998):69–77, 1998.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. of the Eigth National Conference on Artificial Intelligence*, pages 25–32, 1990.

[Sabin and Freuder, 1996] D. Sabin and E.C. Freuder. Configuration as composite constraint satisfaction. In *Proc. Artificial Intelligence and Manufacturing. Research Planning Workshop*, pages 153–161, 1996.

[Soininen *et al.*, 2000] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Unified configuration knowledge representation using weight constraint rules. In *Proc. of the Configuration Workshop held at the ECAI 2000*, pages 79–84, 2000.

[Tiihonen *et al.*, 1998] J. Tiihonen, T. Lehtonen, T. Soininen, A. Pulkkinen, R. Sulonen, and A. Riitahuhta. Modeling configurable product families. In *Proc. fourth WDK Workshop on Product Structuring*, 1998.

[Veron and Aldanondo, 2000] M. Veron and M. Aldanondo. Yet another approach to ccsp for configuration problem. In *Proc. of the Configuration Workshop held at the ECAI 2000*, pages 59–62, 2000.