Automatically Decomposing Configuration Problems*

Luca Anselma, Diego Magro, and Pietro Torasso

Dipartimento di Informatica
Università di Torino
Corso Svizzera 185; 10149 Torino; Italy
{anselma,magro,torasso}@di.unito.it

Abstract. Configuration was one of the first tasks successfully approached via AI techniques. However, solving configuration problems can be computationally expensive. In this work, we show that the decomposition of a configuration problem into a set of simpler and independent subproblems can decrease the computational cost of solving it. In particular, we describe a novel decomposition technique exploiting the compositional structure of complex objects and we show experimentally that such a decomposition can improve the efficiency of configurators.

1 Introduction

Each time we are given a set of components and we need to put (a subset of) them together in order to build an artifact meeting a set of requirements, we actually have to solve a *configuration problem*. Configuration problems can concern different domains. For instance, we might want to configure a PC, given different kinds of CPUs, memory modules, and so on; or a car, given different kinds of engines, gears, etc. Or we might also want to configure abstract entities in non-technical domains, such as students' curricula, given a set of courses.

In early eighties, configuration was one of the first tasks successfully approached via AI techniques, in particular because of the success of $R1/XCON$ [10]. Since then, various approaches have been proposed for automatically solving configuration problems. In the last decade, instead of heuristic methods, research efforts were devoted to single out formalisms able to capture the system models and to develop reasoning mechanisms for configuration. In particular, configuration paradigms based on Constraint Satisfaction Problems (CSP) and its extensions [12, 13, 1, 18] or on logics [11, 3, 16] have emerged.

In the rich representation formalisms able to capture the complex constraints needed in modeling technical domains, the configuration problem is theoretically intractable (at least NP-hard, in the worst case) [5, 15, 16]. Despite the theoretical complexity, many real configuration problems are rather easy to solve [17]. However, in some cases the intractability does appear also in practice and solving some configuration problems can require a huge amount of CPU time. These

* This work has been partially supported by ASI (Italian Space Agency).

A. Cappelli and F. Turini (Eds.): AI*IA 2003, LNAI 2829, pp. 39–52, 2003.

ones are rather problematic situations in those tasks in which low response time is required. E.g. in interactive configuration the response time should not exceed a few seconds and on-line configuration on the Web imposes even stricter requirements on this configurator feature.

There are several ways that can be explored to control computational complexity in practice: among them, making use of off-line knowledge compilation techniques [14]; providing the configurator with a set of domain-specific heuristics, with general focusing mechanisms [6] or with the capability of re-using past solutions [4]; defining techniques for automatically decomposing a problem into a set of simpler subproblems [9, 8]. These approaches are not in alternative and configurators can make use of different combinations of them. However it makes sense to investigate to what extent each one of them can contribute to the improvement of the efficiency of configurators. In the present work, we focus on automatic problem decomposition, since to the best of our knowledge this issue has not received much attention in the configuration community.

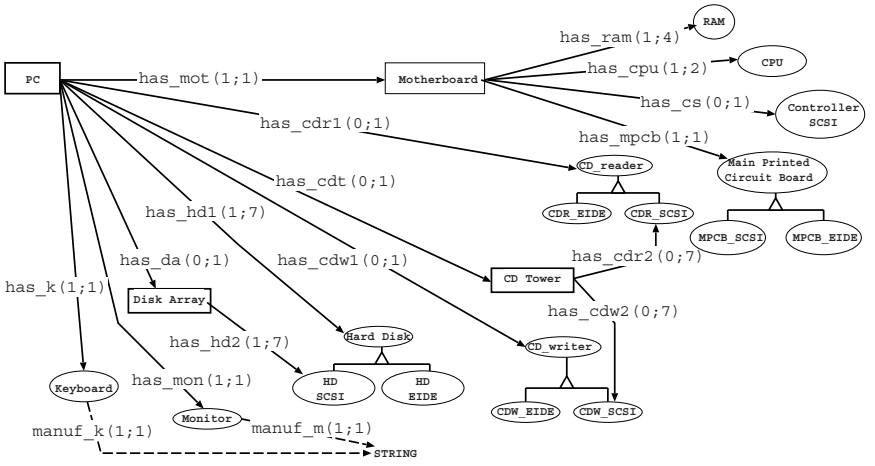
In [7] a structured logical approach to configuration is presented. Here we commit to the same framework as that described there and we present a novel problem decomposition mechanism that exploits the knowledge on the compositional structure (i.e. the knowledge relevant to parts and subparts) of the complex entities that are configured. We also report some experimental results showing its effectiveness.

Section 2 contains an overview of the conceptual language, while Section 3 defines configuration problems and their solutions. In Section 4 a formal definition of the *bound* relation, which problem decomposition is based on, is given; moreover, in that same section, a configuration algorithm making use of decomposition is reported and illustrated by means of an example. Section 5 reports the experimental results, while Section 6 contains some conclusions and a brief discussion.

2 Conceptual Language

In the present paper the $\mathcal{FPC}$ (Frames, Parts and Constraints) [7] language is adopted to model the configuration domains. Basically, $\mathcal{FPC}$ is a frame-based KL-One like formalism augmented with a constraint language.

In $\mathcal{FPC}$, there is a basic distinction between *atomic* and *complex* components. *Atomic components* are the basic building blocks of configurations and they are described by means of properties, while *complex components* are structured entities whose characterization is given in terms of subparts which can be complex components in their turn or atomic ones. $\mathcal{FPC}$ offers the possibility of organizing classes of (both atomic and complex) components in *taxonomies* as well as the facility of building *partonomies* that (recursively) express the *whole-part relations* between each complex component and its (sub)components. A set of *constraints* restricts the set of valid combinations of components and subcomponents in configurations. These constraints can be either specific to the modeled domain or derived from the user's requirements.



CONSTRAINTS

Associated with *PC* class

```
"In any PC, if there is a EIDE main printed circuit board and at least one SCSI device,
 then there must be a controller SCSI"
[co1](<has_mot,has_mpcb>)(in MPCB_EIDE) AND
      ((<has_hd1>)(in HD_SCSI(1;7)) OR (<has_cdr1>)(in CDR_SCSI(1;1)) OR
       (<has_cdw1>)(in CDW_SCSI(1;1))
     )==>(<has_mot,has_cs>)(1;1)
```

Associated with *Motherboard* class

```
"In any motherboard, if there is a SCSI main printed circuit board,
 then there should be no controller SCSI"
[co2](<has_mpcb>)(in MPCB_SCSI)==>(<has_cs>)(0;0)
```

Associated with *CD Tower* class

```
"In any CD tower, there must be at least one CD reader or CD writer"
[co3](<has_cdr2>,<has_cdw2>)(1;14)
```

Fig. 1. A simplified PC conceptual model (CM_{PC})

We illustrate $\mathcal{FPC}$ by means of an example; for a formal description, refer to [7]. In fig. 1 a portion of a simplified conceptual model relevant to PC configuration is represented. The classes of complex components (e.g. *PC*, *Motherboard*, ...) are represented as rectangles, while classes of atomic components (e.g. *Main Printed Circuit Board*, *CD_reader*, ...) are represented as ellipses. *Partonomic roles* represent whole-part relations and are drawn as solid arrows. For instance, the *PC* class has the partonomic role *has_mot*, with minimum and maximum cardinalities 1, meaning that each PC has exactly one motherboard; partonomic role *has_cdr*1, whose minimum and maximum cardinalities are 0 and 1, respectively, expresses the fact that each PC can optionally have one CD reader, and so on. It is worth noting that the motherboard is a complex component having 1 to 4 RAM modules (see the *has_ram* partonomic role), one main printed circuit board (*has_mpcb* role), that can be either the SCSI or the EIDE type, etc.

Descriptive roles represent properties of components and they are drawn as dashed arrows. For example, the $Monitor$ component has a *string* descriptive role $manuf\_m$, representing the manufacturer.

Each *constraint* is associated with a class of complex components and is composed by $\mathcal{FPC}$ predicates combined by means of the boolean connectives $\wedge,\vee,\neg,\rightarrow$. A *predicate* can refer to cardinalities, types or property values of (sub)components. The reference to (sub)components is either direct through partonomic roles or indirect through *chains of partonomic roles*. For example, in fig. 1 [$co2$] is associated with the $Motherboard$ class and states that, if has_mpcb role takes values in $MPCB\_SCSI$ (i.e. the main printed circuit board is the SCSI type), then has_cs relation must have cardinality 0 (i.e. there must be no SCSI controller). An example of a chain of partonomic roles can be found in [$co1$]: the consequent of the constraint [$co1$] (associated with PC class) states that the role chain $\langle has\_mot, has\_cs\rangle$ has cardinality 1, i.e. the PC component has one Motherboard with one SCSI Controller. [$co3$] shows an example of a *union of role chains*: a component of type $CD\ Tower$ must have 1 to 14 CD_readers *or* CD_writers.

3 Configuration Problems

A *configuration problem* is a tuple $CP = \langle CM, T, c, C, V\rangle$, where CM is a conceptual model, T is a partial description of the complex object to be configured (the *target object*), c is a complex component occurring in T (either the target object itself or one of its complex (sub)components) whose type is C (which is a class of complex objects in CM) and V is a set of constraints involving component c. In particular, V can contain the user's requirements that component c must fulfill.

Given a configuration problem CP, the task of the configurator is to refine the description T by providing a complete description of the component c satisfying both the conceptual description of C in CM and the constraints V, or to detect that the problem does not admit any solution.

Configuration Process We assume that the configurator is given a main configuration problem $CP_0 = \langle CM, (c), c, C, REQS\rangle$, where c represents the target object, whose initial partial description $T \equiv (c)$ contains only the component c; $REQS$ is the set of requirements for c (expressed in the same language as the constraints in CM[1]). Therefore, the goal of the configurator is to provide a complete description of the target object (i.e. of an individual of the class C)

[1] It is worth pointing out that the user actually specifies her requirements in a higher level language (through a graphic interface) and the system performs an automatic translation into the representation language. This translation process may also perform some inferences, e.g. if the user requires a PC with a CD tower containing at least one CD reader and at least one CD writer, the system infers also an upper bound for the number of components of these two kinds, as in requirements $req3$ and $req4$ in fig. 2, where the upper bound 7 is inferred for both the number of CD readers and of CD writers that the CD tower can contain.

```
The manufacturer of the monitor must be the same as that of the keyboard
[req1](<has_mon,manuf_m>)=(<has_k,manuf_k>)

It must have a disk array
[req2](<has_da>)(1;1)

It must have a CD tower with at least one CD reader and at least one CD writer
[req3](<has_cdt,has_cdr2>)(1;7)
[req4](<has_cdt,has_cdw2>)(1;7)

It must have no more than 4 SCSI devices
[req5](<has_cdr1>,<has_cdw1>,<has_hd1>,<has_cdt,has_cdr2>,<has_cdt,has_cdw2>,
       <has_da,has_hd2>)(in CDR_SCSI ∪ CDW_SCSI ∪ HD_SCSI(0;4))
```

Fig. 2. User's Requirements for a PC ($REQS_{PC}$)

satisfying the model CM and fulfilling the requirements $REQS$ (such a description is a *solution* of the configuration problem) or to detect that the problem does not admit any solution (i.e. that such an individual does not exist). Since CM is assumed to be consistent, this last case happens only when the requirements $REQS$ are inconsistent w.r.t. CM. A sample description of an individual PC satisfying the conceptual model CM_{PC} in fig. 1 and fulfilling the requirements listed in fig. 2 is reported in fig. 4.f.

The configuration is accomplished by means of a search process that progressively refines the description of c. At each step the configuration process selects a complex component in T (starting from the target object), it refines the description T by inserting a set of direct components of the selected component (by choosing both the number of these components and their type) and then it configures all the direct complex components possibly introduced in the previous step. If, after a choice, any constraint (either in CM or in $REQS$) is violated, then the process backtracks. The process stops as soon as a solution has been found or when the backtracking mechanism cannot find any open choice. In the last case, CP does not admit any solution.

4 Decomposing Configuration Problems

Because of the inter-role constraints, both those in CM and those in $REQS$, a choice made by the configurator for a component can influence the valid choices for other components. In [9, 8] it is shown that the compositional knowledge (i.e. the way the complex product is made of simpler (sub)components) can be exploited to partition the constraints that hold for a given component into sets in such a way that the components involved in constraints of two different sets can be configured independently. While such a decomposition has been proved useful in reducing the actual computational effort in many configuration problems, here we present an enhancement of such a decomposition mechanism that considers constraints as dynamic entities instead of static ones.

4.1 Bound and Unbound Constraints

The decomposition capability is based on a *bound* relation among constraints. We assume that, in any configuration, each individual component cannot be a direct part of two different (complex) components, neither a direct part of a same component through two different whole-part relations (*exclusiveness assumption on parts*).

Let $CP = \langle CM, T, c, C, V \rangle$ and $CONSTRS(C)$ be a configuration problem and the set of constraints associated with C in CM, respectively and let $u, v, w \in V \cup CONSTRS(C)$. The *bound* relation $\mathcal{B}_c$ is defined as follows: **if** P_u and P_v are two predicates occurring in u and in v, respectively, that mention both a *same* partonomic role p of C **then** $u\mathcal{B}_c v$ (i.e. if u and v refer, through their predicates, to a same part of c, then they are ***directly*** bound in c); **if** $u\mathcal{B}_c v$ **and** $v\mathcal{B}_c w$ **then** $u\mathcal{B}_c w$ (i.e. u and w are bound by ***transitivity*** in c). It is easy to see that $\mathcal{B}_c$ is an equivalence relation.

To solve $CP = \langle CM, T, c, C, V \rangle$, the configurator must refine the description of c by specifying the set $COMPS(c)$ of its components and subcomponents. In particular, it specifies the type of each element in $COMPS(c)$ and, for each partonomic role occurring in the conceptual description of type C (the type of component c) in CM, it specifies which elements in $COMPS(c)$ play that partonomic role.

If S_1 and S_2 are two different equivalence classes of constraints induced by the relation $\mathcal{B}_c$, let $COMPS_{S_1}(c)$ and $COMPS_{S_2}(c)$ be the sets of components in $COMPS(c)$ referred to by constraints in S_1 and in S_2, respectively. Given the exclusiveness assumption on parts, these two sets are disjoint and, for every pair of components $c_1 \in COMPS_{S_1}(c)$ and $c_2 \in COMPS_{S_2}(c)$, there is no constraint in $V \cup CONSTRS(C)$ linking them together. It follows that the choices of the configurator relevant to the components in $COMPS_{S_1}(c)$ do not interact with those relevant to the components in $COMPS_{S_2}(c)$. In other words, S_1 and S_2 represent two mutually independent configuration subproblems.

4.2 Decomposition Mechanisms

In fig. 3 a configuration algorithm making use of decomposition is sketched. For lack of space, let us illustrate the algorithm just by means of an example. Let's suppose that the user wants to configure a PC (described by the conceptual model CM_{PC} in fig. 1) meeting the set $REQS_{PC}$ of requirements stated in fig. 2.

At the beginning, the configurator is given the problem $CP_0 = \langle CM_{PC}, (pc1), pc1, PC, REQS_{PC} \rangle$. Besides the requirements $REQS_{PC}$, the set of constraints associated with PC in CM_{PC} are also considered to fully specify the problem (statement in row 3 of the algorithm in fig. 3). This initial situation is represented in fig. 4.a.

Initial Decomposition Step (statements in rows 5 and 6). Before starting the actual configuration process, the configurator attempts to decompose the

```
(1) configure(CM,T,c,C,V){
(2)   SUBPROBLEMS = <>;
(3)   - add to V the constraints associated with C in CM;
(4)   currentSP=V;
(5)   S=decompose(CM,T,c,currentSP);
(6)   for each s in S push(s, SUBPROBLEMS);
(7)   while(SUBPROBLEMS ≠ <>){
(8)     currentSP=pop(SUBPROBLEMS);
(9)     if(no choice made for the direct components of c involved in currentSP){
(10)       T = insertDirectComponents(CM,T,c,currentSP);
(11)       if(T== FAILURE) return FAILURE;
(12)    }else{
(13)       - choose a direct complex component d of c that has not been configured
             yet and that is involved in currentSP (let D be the type of d);
(14)       T=configure(CM,T,d,D,currentSP);
(15)       if(T==FAILURE) BACKTRACK;
(16)      }
(17)     - remove satisfied constraints from currentSP;
(18)     if(not solved currentSP){
(19)        currentSP=reviseConstraints(CM,c,currentSP);
(20)        S=decompose(CM,T,c,currentSP);
(21)        for each s in S push(s,SUBPROBLEMS);}
(22)   }//while
(23)   - complete T by inserting all the components and subcomponents of c not
         involved in the constraints in V
(24)   return T;
(25) }//configure
```

Fig. 3. Configuration algorithm overview

constraints that hold for the target object $pc1$. To do this, it partitions the constraints
$currentSP = [req1, \ldots, req5, co1]$ into a set of equivalence classes by computing the *bound* relation $\mathcal{B}_{pc1}$ in this set: it is easy to see that the constraints $req2, \ldots, req5, co1$ are bound in $pc1$ according to the definition of the *bound* relation. Instead, $req1$ is not bound with any other constraint belonging to $currentSP$. It follows that $currentSP$ can be partitioned into the two equivalence classes of contraints $S_1 = [req2, \ldots, req5, co1]$ and $S_2 = [req1]$, each one entailing a configuration subproblem.

Resolution of Subproblems (*while* statement in rows 7 to 22). These subproblems are mutually independent. One subproblem is chosen as the current one (in this example that one relevant to the constraints $S_1 = [req2, \ldots, req5, co1]$) and the other ones (in this example only that one relevant to $S_2 = [req1]$) are pushed into the $SUBPROBLEMS$ stack (see fig. 4.b).

Insertion of Direct Components (statement in row 10). To solve S_1 the configurator refines the description of the target object by inserting in it only those direct components of $pc1$ involved in the constraints relevant to the current subproblem. More precisely, the configurator considers each partonomic role p of PC class occurring in the constraints belonging to S_1 and makes for p *two basic choices*: it chooses the number of direct components, playing the partonomic role p, to insert into the configuration and, for each one of them, it chooses its type. In this example, let's suppose that a CD reader, a CD writer, a hard disk (all of the SCSI type), a motherboard, a CD tower and a disk array are inserted

Fig. 4. A configuration example

into the current configuration (fig. 4.c). Since configuration is accomplished by means of a search process, it is worth pointing out that all the open choices (for instance, the alternative EIDE type for the CD reader, the CD writer and the hard disk, or the possibility of inserting more than one hard disk) have to be remembered as they may be explored as a consequence of a backtracking.

Removal of Satisfied Constraints (statement in row 17). The current tentative configuration T_2 does not contradict any constraint relevant to the current subproblem, moreover requirement $req2$ (imposing the existence of a disk array in the configured PC) is now satisfied and it can be removed from $currentSP$. The truth values of the other constraints belonging to $currentSP$ cannot be computed yet, since the configurator has not yet configured all the parts of the target object which these constraints refer to. For instance, a CD tower has been inserted into the current tentative configuration T_2, but it has not been configured yet; therefore, up to this point, it is impossible to know how many CD readers the CD tower will contain and thus the truth value of $req3$ is still unknown. Since $currentSP$ still contains some constraints (whose truth values

are unknown) referring to parts of some direct components of $pc1$ not yet considered by the configurator, the subproblem relevant to $currentSP$ is not solved yet.

Further Decomposition Step (rows 18 to 21). After having refined the description of $pc1$ with the insertion of some of its direct components, the configurator attempts a further decomposition of the current subproblem.

Revision of Constraints and Re-computation of Bound Relation. To perform this decomposition step, the configurator dynamically updates the form of the constraints in $currentSP$ (i.e. the constraints are treated as dynamic entities). In this sample case, even if the truth value of constraint $co1$ cannot be determined in the tentative configuration T_2, for some predicates occurring in $co1$ it is possible to say whether they are true or false. In particular, the predicates $(\langle has\_hd1\rangle)(in\ HD\_SCSI(1;7))$, $(\langle has\_cdr1\rangle)(in\ CDR\_SCSI(1;1))$ and $(\langle has\_cdw1\rangle)(in\ CDW\_SCSI(1;1))$ are all true in T_2. Therefore, in the context of the choices made by the configurator and that leaded to T_2, these predicates can be substituted by their truth values in $co1$ and $co1$ can be simplified in the following way:

$$[co1'](\langle has\_mot, has\_mpcb\rangle)(in\ MPCB\_EIDE) \rightarrow (\langle has\_mot, has\_cs\rangle)(1;1).$$

Since the revision of the constraints relevant to the current subproblem may remove some predicates from the constraints (as it happens for $co1$ in this example), it may happen that some constraints that were previously bound have now become unbound, therefore it makes sense to compute the *bound* relation again, in this revised set of constraints. In our example, the relation $\mathcal{B}_{pc1}$ induces a partitioning of the revised set of constraints $currentSP = [req3, req4, req5, co1']$ into the two classes $S_{11} = [co1']$ and $S_{12} = [req3, req4, req5]$ of bound constraints. This means that *in the context of tentative configuration* T_2 (fig. 4.c), the current subproblem has been further decomposed into a set of independent subproblems.

Resolution of Subproblems (*while* statement in rows 7 to 22). As in the previous execution of the body of the *while*, the configurator chooses one subproblem as the current one (in this case, $currentSP = S_{11}$) while the other ones (in this case only that one relevant to S_{12}) have been pushed into the $SUBPROBLEMS$ stack. All the direct components of $pc1$ involved in the set $currentSP$ of constraints have already been inserted into the tentative configuration. To solve S_{11}, the motherboard $mb1$ needs to be configured: indeed, $co1'$ refers both to the main printed circuit board and to the optional SCSI controller, which are $mb1$ components (rows 13 to 15). This means solving the configuration problem $CP_{mb1} = \langle CM_{PC}, T2, mb1, Motherboard, \{co1'\}\rangle$.

The configuration of $mb1$ has to take into account both the set S_{11} of constraints and constraint $co2$ associated with $Motherboard$ class in CM_{PC} (fig. 1).

In this example, a SCSI main printed circuit board $mpcb\_scsi1$ is inserted into the tentative configuration, therefore no SCSI controller is inserted (because of $co2$). To complete the configuration of $mb1$, the configurator inserts also a CPU ($cpu1$) and four memory modules (fig. 4.d). Constraint $co1'$ is now satisfied, thus it is removed from $currentSP$. Since $currentSP$ does not contain any

other constraint, the configuration of $mb1$ represents a solution to the current subproblem. The subproblem entailed by $S_{12} = [req3, req4, req5]$ becomes the current one.

This subproblem involves the $pc1$ direct complex components $cdt1$ and $da1$. It should be clear that there is no way of extending the tentative configuration $T3$ by configuring these two components while satisfying the constraints in S_{12}.

Indeed, $req3$ and $req4$ require that at least one CD reader and at least one CD writer are inserted into $cdt1$ and, given the conceptual model CM_{PC}, these two devices must be the SCSI type. The conceptual model states also that all the hard disks in the disk array are the SCSI type (and that there is at least one hard disk in a disk array). However, $T3$ already contains 3 SCSI devices; it follows that $pc1$ would have at least 6 SCSI devices and this is in contradiction with requirement $req5$. Therefore the configuration process has to backtrack and to revise some choices. It is worth noting that it would be useless to find an alternative configuration for the motherboard, since $mb1$ was configured while considering the subproblem relevant to S_{11}, which was independent from the one entailed by S_{12} (for which the failure occurred). Therefore, let's suppose that the backtracking mechanism changes from SCSI to EIDE the types of the CD reader and of the CD writer playing the partonomic roles has_cdr1 and has_cdw1, respectively. After that, the tentative configuration $T4$ is produced (fig. 4.e). It is easy to see that $T4$ satisfies all the constraints in $S_1 = [req2, \ldots, req5, co1]$, therefore it represents a solution to the first of the two subproblems the main configuration problem CP_0 was decomposed into (see above).

To solve the main problem, the tentative configuration $T4$ must be extended in order to solve the subproblem entailed by $S_2 = [req1]$ too. $T5$ in fig. 4.f is a global solution.

This simple example illustrates a situation in which the configurator succeeds in further decomposing the current subproblem, after having inserted the direct components of the target object which the current set $currentSP$ of constraints refer to. However, it is worth noting that, in general, the configurator attempts to further decompose the current subproblem also after having completely configured each direct complex component of the target object (see the algorithm in fig. 3). Moreover, for the sake of simplicity, the example focuses only on the problem decomposition performed by partitioning the constraints relevant to the target object: it should be noticed that the decomposition is not limited to the target object, but, on the opposite, it is recursively performed also when configuring its complex (sub)components (by the execution of the recursive invocation in row 14) .

5 Experimental Results

The algorithm described in the previous section has been implemented in a configuration system written in Java (JDK 1.3). In this section we report some results from tests conducted with a computer system configuration domain. The experiments are aimed at testing the performance of the configuration algorithm

described in this paper and at comparing it (w.r.t. the computational effort) with a configuration strategy without decomposition and with the most performant decomposition strategy previously defined is this framework, the one called in [9] "strategy 3" (see [8, 9]). We call the algorithm in [9] *static decomposition* algorithm and the algorithm in Section 4.2 *dynamic decomposition* algorithm. All experiments were performed on a Mobile Pentium III 933 MHz 256 MB Windows 2000 notebook.

Using the computer system model, we generated *a test set of* 200 *configuration problems*; for each of them we specified the type of the target object (e.g. a PC for graphical applications) and some requirements that must be satisfied (e.g. it must have a CD writer of a certain kind, it must be *fast enough* and so on). In 83 problems we intentionally imposed a set of requirements inconsistent with the conceptual model (in average, these problems are quite hard). A problem is considered solved iff the configurator provides a solution or it detects that the problem does not admit any solution. For each problem the CPU time and the number of backtrackings that it required have been measured. The configuration algorithms include some random choices: e.g. after decomposing a problem, the selection of the current subproblem (see Section 4.2) is performed randomly. To reduce the bias due to "lucky" or "unlucky" random choices, every experiment was performed *ten times* and the average values of measured parameters were considered.

The strategy with dynamic decomposition proves to be effective in reducing the time and the number of backtrackings required by a problem to be solved w.r.t. both the algorithm without decomposition and the algorithm with static decomposition. Figure 5 shows the frequency histograms of the CPU times. On the X axis is reported the time interval taken in consideration and on Y axis is reported the number of problems solved within the given interval. The chart shows that the dynamic decomposition is rather effective in "moving" CPU times to low values, particularly to values less than 3 seconds. Figure 6 reports the relative cumulative distribution graphs for CPU times. In this case the Y axis reports

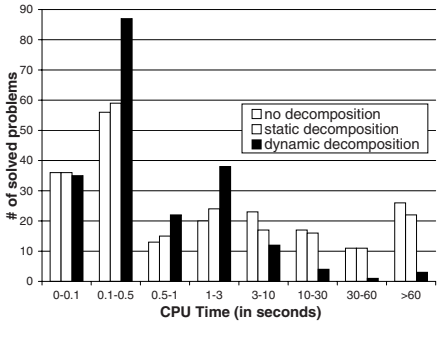


Fig. 5. Frequency histogram of CPU time

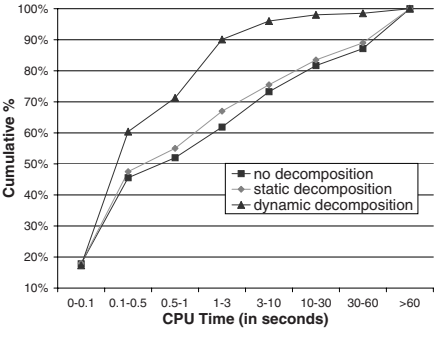


Fig. 6. Relative cumulative frequency graph of CPU Time

the cumulative frequencies of problems solved within the given interval. It may be worth to notice that the 90^{th} percentile for strategy without decomposition is 164 s, for static decomposition it is 68 s, while it is 2.5 s for strategy with dynamic decomposition. Results regarding CPU times are reflected by those regarding the number of backtrackings. Histograms and graphs are similar to those reported for CPU times (because of space constraints it is not possible to show them here). The 90^{th} percentile for the number of backtrackings is 14293 for no decomposition, 8056 for static decomposition and 323 for dynamic decomposition, resulting in a significative reduction of the number of backtrackings, too.

6 Conclusion and Discussion

In some configuration domains the theoretical intractability of configuration problems can appear also in practice since a few configuration problems can require a huge amount of CPU time to be solved. Some tasks, such as interactive configuration and on-line configuration on the Web, need low response times by the configurator, therefore the issue of controlling in practice the computational complexity of configuration problems should be dealt with.

In this paper we have investigated the role of problem decomposition in improving the efficiency of configurators.

Other researchers have recognized the importance of decomposition in solving difficult configuration problems. In particular, in [2], the authors stress the need of designing modular configuration models with low interaction among modules in such a way that the modules can be solved one by one. However, little attention has been paid to provide the configurator with mechanisms to automatically decompose configuration problems.

We have defined a decomposition technique, in a structured logical approach to configuration, that exploits compositional knowledge in order to partition configuration problems into a set of simpler (and independent) subproblems.

In [9, 8] some decomposition mechanisms were presented. Although these decomposition techniques have proved to be useful in reducing CPU times, still they do not allow to solve the large majority of the problems in a time acceptable for interactive and on-line configuration, i.e. in less than few seconds. In this work we have extended both the one called in [8] *constraints-splitting decomposition* and those defined in [9]. Differently from *constraints-splitting decomposition*, the mechanism presented here allows the configurator to perform decomposition recursively by partitioning both the constraints directly associated with the target object and those associated with its components and subcomponents. Moreover, in the decomposition techniques defined in [9, 8], the constraints are treated as static entities, while here we have proposed an improved mechanism that is able to perform more decompositions by dynamically simplifying the constraints during the configuration process.

Some experimental results conducted in a computer system configuration domain are reported which show the effectiveness of the decomposition technique presented here.

Few cases of the test set still required a huge amount of CPU time (more than 60 s), therefore we do not claim that decomposition is the "silver bullet" for difficult configuration problems. However, the experimental results suggest that it can play an important role in increasing the efficiency of configurators, therefore it is worth investigating various integrations of decomposition and other techniques (off-line knowledge compilation, re-using past solutions and so on).

References

[1] G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, (July/August 1998):59–68, 1998. 39

[2] G. Fleischanderl and A. Haselböck. Thoughts on partitioning large-scale configuration problems. In *AAAI 1996 Fall Symposium Series*, pages 1–10, 1996. 50

[3] G. Friedrich and M. Stumptner. Consistency-based configuration. In *AAAI-99, Workshop on Configuration*, 1999. 39

[4] L. Geneste and M. Ruet. Fuzzy case based configuration. In *Proc. ECAI 2002 Configuration WS*, pages 71–76, 2002. 40

[5] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977. 39

[6] D. Magro and P. Torasso. Interactive configuration capability in a sale support system: Laziness and focusing mechanisms. In *Proc. IJCAI-01 Configuration WS*, pages 57–63, 2001. 40

[7] D. Magro and P. Torasso. Supporting product configuration in a virtual store. *LNAI*, 2175:176–188, 2001. 40, 41

[8] D. Magro and P. Torasso. Decomposition strategies for configuration problems. *AIEDAM, Special Issue on Configuration*, 17(1), 2003. 40, 43, 49, 50

[9] D. Magro, P. Torasso, and Luca Anselma. Problem decomposition in configuration. In *Proc. ECAI 2002 Configuration WS*, pages 50–55, 2002. 40, 43, 49, 50

[10] J. McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, (19):39–88, 1982. 39

[11] D. L. McGuinness and J. R. Wright. An industrial-strength description logic-based configurator platform. *IEEE Intelligent Systems*, (July/August 1998):69–77, 1998. 39

[12] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. of the AAAI 90*, pages 25–32, 1990. 39

[13] D. Sabin and E. C. Freuder. Configuration as composite constraint satisfaction. In *Proc. Artificial Intelligence and Manufacturing. Research Planning Workshop*, pages 153–161, 1996. 39

[14] C. Sinz. Knowledge compilation for product configuration. In *Proc. ECAI 2002 Configuration WS*, pages 23–26, 2002. 40

[15] T. Soininen, E. Gelle, and I. Niemelä. A fixpoint definition of dynamic constraint satisfaction. In *LNCS 1713*, pages 419–433, 1999. 39

[16] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with weight constraint rules. In *Proc. of the AAAI Spring 2001 Symposium on Answer Set Programming*, 2001. 39

[17] J. Tiihonen, T. Soininen, I. Niemelä, and R. Sulonen. Empirical testing of a weight constraint rule based configurator. In *Proc. ECAI 2002 Configuration WS*, pages 17–22, 2002. 39

[18] M. Veron and M. Aldanondo. Yet another approach to ccsp for configuration problem. In *Proc. ECAI 2000 Configuration WS*, pages 59–62, 2000. 39