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).

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.

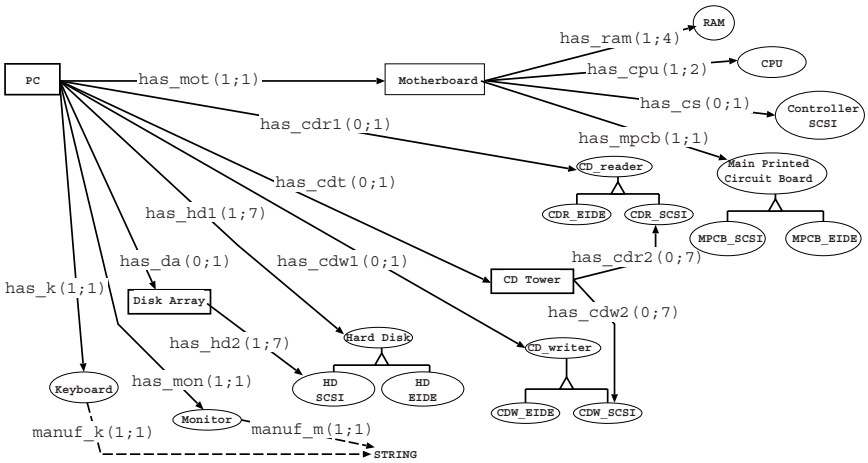
In [7] a structured logical approach to configuration is presented. Here we commit to the same framework as that described there and we present a novel problem decomposition mechanism that exploits the knowledge on the compositional structure (i.e. the knowledge relevant to parts and subparts) of the 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* restrict 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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.