Decomposing and Distributing Configuration Problems

Diego Magro and Pietro Torasso

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

Abstract. In the present work the issue of decomposing and distributing a configuration problem is approached in a framework where the domain knowledge is represented in a structured way by using a KL-One like language, where *whole-part relations* play a major role in defining the structure of the configurable objects. The representation formalism provides also a constraint language for expressing complex relations among components and subcomponents.
The paper presents a notion of *boundness* among constraints which specifies when two components can be independently configured. *Boundness* is the basis for partitioning constraints and such a partitioning induces a decomposition of the configuration problem into independent subproblems that are distributed to a pool of configurators to be solved in parallel.
Preliminary experimental results in the domain of PC configuration showing the effectiveness of the decomposition technique in a sequential approach to configuration are also presented.

1 Introduction

In recent years configuration has attracted a significant amount of attention not only from the application point of view but also from the methodological one [12]. In particular, logical approaches such as [13,4] and approaches based on CSP have emerged [10,3,11, 14]. In CSP approaches, configuration can exploit powerful constraint problem solvers for solving complex problems [5,1]. From the other hand, logical approaches make use of a more explicit and structured representation of the entities to be configured (e.g. [9]). Logical approaches seem to offer significant benefits when interaction with the user (e.g. [7]) and explainability of the result (or failure) are major requirements.

Configuration, as many other tasks, can be computationally expensive; therefore, the idea of problem decomposition looks attractive since, from the early days of AI, problem decomposition has emerged as one of the most powerful mechanisms for controlling complexity. Ideally, the solution of a complex configuration problem should be easily assembled by combining the solutions of the subproblems the initial problem has been decomposed into. Moreover, decomposing a configuration problem into a set of mutually independent subproblems allows the configuration process to be distributed among a set of configurators working in parallel. Unfortunately, in many cases it is not obvious at all how to perform such a decomposition.

In the present work the issue of decomposing a configuration problem is approached in a framework where knowledge about the entities is represented in a structured way by using a KL-One like language augmented with a constraint language for expressing

D. Scott (Ed.): AIMSA 2002, LNAI 2443, pp. 81–90, 2002.

complex inter-role relations (see section 2 for a summary of the representation language). Partonomic relations provide the basic knowledge for decomposing the configuration problem. In fact, two subparts involved into two partonomic relations can not be independently configured if there is at least a constraint that links them together. For this reason we have introduced a notion of boundness among constraints (section 3) which assures that two components not involved in a same set of *bound* constraints can be independently configured.

Section 4 provides a high-level description of the configuration algorithm and of the decomposition strategy, while an example of an application of the algorithm is shown in section 5. Section 6 reports preliminary experimental results concerning the reduction of the computational effort. A discussion of the approach is reported in section 7.

2 The Conceptual Language

In the last few years we have developped a representation formalism called $\mathcal{FPC}$ [6,7] (Frames, Parts and Constraints) for modeling configuration problems. Basically, $\mathcal{FPC}$ is a frame-based KL-One like formalism augmented with a constraint language.

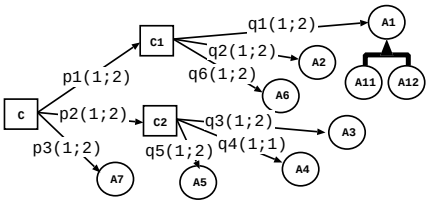
In $\mathcal{FPC}$, there is a basic distinction between *atomic* and *complex* components. *Atomic components* are described by means of a set of features characterizing the component itself, while *complex components* can be viewed as structured entities whose characterization is mainly given in terms of their (sub)components, 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 any one of its (sub)components. Moreover, in any complex component, a set of *constraints* restricts the set of valid combinations of its (sub)components.

Frames and Parts. In $\mathcal{FPC}$, each *frame* represents a *class* of components (either *atomic* or *complex*) and it has a set of *member slots* associated with it. Each slot represents a *property* of the components belonging to the class and it can be of type either *partonomic* or (alternatively) *descriptive*. Any slot p of a class C is described via a value restriction D(that can be another class or a set of values of a predefined kind) and a number restriction (i.e. an integer interval $[m,n]$ with $m \leq n$), as usual in the KL-One like representation formalisms. A slot p with value restriction D and number restriction $[m,n]$ captures the fact that the property p for any component of type C is expressed by a (multi)set of values of type D whose cardinality belongs to the interval $[m,n]$.

In the following we restrict our attention to partonomic slots since in this framework they represent the basic knowledge for problem decomposition.

Partonomic slots are used for capturing the *whole-part relation* among components. In $\mathcal{FPC}$ this relation is assumed to be asymmetric and transitive. Formally, any partonomic slot p of a class C is interpreted as a relation $p : C \rightarrow D$ such that $(\forall c \in C)(m \leq |p(c)| \leq n))$, being D and $[m,n]$, respectively, its value and its number restrictions; the meaning is straightforward: any complex component of type C has from a minimum of m up to a maximum of n direct parts of type D via a whole-part relation named p.

It is worth noting that in each configuration a component can neither be a direct part of two different complex components nor a direct part of the same complex component via two different whole-part relations (*exclusiveness assumption on parts*).



CONSTRAINTS
Associated with C:
[co1]({<p1,q1>})(1;1) ==> ({<p2,q5>})(1;1)
[co2]({<p1,q2>})(1;1) ==> ({<p2,q3>})(1;1)
[co3]({<p2,q3>})(2;2) ==> ({<p2,q4>})(2;2)
[co4]**true** ==> ({<p1,q1>})(**in** A11 (1;4))
[co5]({<p3>})(1;1) ==> ({<p3>})(**in** A71)
Associated with C1:
[co6]({<q1>})(1;1) ==> ({<q6>})(**in** A61)

Fig. 1. A toy conceptual model

Figure 1 contains a toy conceptual model that we use here as a simple example. Each rectangle represents a class of complex components, each oval represents a class of atomic components and any thin solid arrow corresponds to a partonomic slot. In the figure, it is stated, for instance, that C is a class of complex components and the partonomic slot $p1$ specifies that each instance of C has to contain one or two (complex) components of type $C1$; whereas the partonomic slot $p3$ states that any instance of C has to contain one or two (atomic) components of type $A7$.

In any conceptual model, a *slot chain* $\gamma = \langle p_1, \ldots, p_n \rangle$, starting in a class C and ending in a class D is interpreted as the relation composition $p_n \circ p_{n-1} \circ \ldots \circ p_1$ from C to D. The chain represents the subcomponents of a complex component $c \in C$ via the *whole-part* relations named $p_1, \ldots, p_n$. In figure 1, for example, $\langle p1, q1 \rangle$ denotes the subcomponents (of type $A1$) of each instance of C through the partonomic slots $p1$ and $q1$. Similarly, a set of slot chains $R = \{\gamma_1, \ldots, \gamma_m\}$ (where each γ_i starts in C and ends in D_i) is interpreted as the relation union $\bigcup_{i=1}^{m} \gamma_i$ from C to $\bigcup_{i=1}^{m} D_i$.

Besides the partonomies, also the taxonomies are useful in the conceptual models. In figure 1 the subclass links are represented by thick solid arrows. In that toy domain we assume that each class of atomic components Ai is partitioned into two subclasses $Ai1$ and $Ai2$. Only the partitioning of $A1$ into $A11$ and $A12$ is reported in figure.

Constraints. A set (possibly empty) of *constraints* is associated with each class of complex components. These constraints allow one to express those restrictions on the components and the subcomponents of the complex objects that can't be expressed by using only the frame portion of $\mathcal{FPC}$, in particular the inter-slot constraints that cannot be modeled via the number restrictions or the value restrictions.

Each constraint cc associated with 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 component $c \in C$, if c satisfies α then it must satisfy β. It should be clear that if $\alpha = true$, then, for each $c \in C$, β must always hold, while if $\beta = false$, then, for each $c \in C$, α can never hold.

In the following we present only a simplified version of some predicates available in $\mathcal{FPC}$. For a more complete description of them see [6].

Let $R = \{\gamma_1, \ldots, \gamma_m\}$, where each $\gamma_i = \langle p_{i_1}, \ldots p_{i_n} \rangle$ is a slot chain starting in a class of C complex components. For any $c \in C$, $R(c)$ denotes the values of the relation R computed for c.
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 $h \leq k$.
2) $(R)(inI)$. $c \in C$ satisfies the predicate *iff* $R(c) \subseteq I$, where I is a union of classes in the conceptual model.
3) $(R)(inI(h;k))$. $c \in C$ satisfies the predicate *iff* $h \leq |R(c) \cap I| \leq k$, where h, k are non negative integers with $h \leq k$ and I is a union of classes in the conceptual model.

For example, the constraint $co5$ states that if only one component playing the partonomic role $p3$ is present in a configuration of an object of type C, then this component must be of type $A71$. It is worth noting that the user's requirements are automatically translated into the $\mathcal{FPC}$ constraint language. For example, the (user's) requirement of inserting in a configuration of an object of type C from 1 up to 4 subcomponents of type $A11$ is expressed by the constraint $co4$.

3 The Role of Partonomic Knowledge in Problem Decomposition

Given this framework, configuring a complex object of type C means to completely determine an instance c of C in which all the partonomic slots of C are instantiated and each direct component of c is completely configured too. c has to respect both the conceptual model (number and value restrictions imposed by the taxonomy and the partonomy as well as the constraints associated with the classes of components involved in c) and the user's requirements.

Configuring a complex component by taking into consideration only its taxo- partonomic description would be a straightforward activity. In fact, for any well formed model expressed in $\mathcal{FPC}$ in which no constraints are associated with any class, a configuration respecting that model would always exist. A simple algorithm could find it without any search and by simply starting from the class of the target object (i.e. the one for which the configuration has to be built), considering each slot p of that class and, for it, choosing its cardinality, i.e. choosing the number of components playing the partonomic role p to introduce into the configuration, and the type for each such component. This process must be recursively repeated for each complex component introduced in the configuration, until all the atomic ones are reached. In this process the algorithm needs only to respect the number and the value restrictions of the slots. Unfortunately, this is not realistic. The conceptual model usually contains complex constraints that link together different slots. In this more realistic situation a solution can't be generally found by making only a set of local choices and without resorting to search in a large space of alternatives.

Moreover, the requirements usually imposed by the user to the target artifact further restrict the set of legal configurations. This means that the search for a configuration is not guaranteed to be fruitful any more. In fact, even assuming the consistency of the conceptual model, the user's requirements could be inconsistent w.r.t. it and in such a case no configuration respecting the model and satisfying the requirements exists.

Therefore, in general, the task of solving a configuration problem can be rather expensive from a computational point of view. As we have mentioned above, in $\mathcal{FPC}$ framework this is mainly due to the constraints (both those that are part of the conceptual model and those imposed as user's requirements) that link together different components

and subcomponents. In these situations a choice made for a component during the configuration process might restrict the choices actually available for another one, possibly preventing the latter to be fully configured. In such cases the configuration process has to revise some decision that it previously took and to explore a different path in the search space. Usually, in real cases the search space is rather huge and many paths in it don't lead to any solution.

However, in many cases it does not happen that every constraint interacts with each other and the capability of recognizing the sets of (potentially) interacting constraints can constitute the basis for decomposing a problem into independent subproblems.

Once a problem has been decomposed into a set of independent subproblems, these could be solved concurrently and with a certain degree of parallelism, potentially reducing the overall computational time. However, also a sequential configuration process can take advantage of the decomposition. In fact, if two subproblems are recognized to be independent, the configurator is aware that no choice made during the configuration process of the first one needs to be revised if it enters a failure path while solving the second one.

To be effective, the task of recognizing the decomposability of a problem (and of actually decomposing it) should not take too much time w.r.t. the time requested by the whole resolution process.

In our approach, the partonomic knowledge can be straightforwardly used in recognizing the interaction among constraints (with an acceptable precision) and in defining a way of decomposing a configuration problem into independent subproblems. With this aim, we introduce the *bound* relation among constraints. Intuitively, two constraints are *bound* iff the choices made during the configuration process in order to satisfy one of them *can* interact with those actually available for the satisfaction of the second one. If c is a complex component in a (tentative) configuration, the **bound relation** $\mathcal{B}_c$ is defined in the set $CONSTRS(c)$ of the constraints that c must satisfy, as follows: let $u, v, w \in CONSTRS(c)$. **If** u and v contain both a *same* partonomic slot p of $class(c)$ **then** $u\mathcal{B}_c v$ (i.e. if u and v refer to a same part of c, they are bound); **if** $u\mathcal{B}_c v$ **and** $v\mathcal{B}_c w$ **then** $u\mathcal{B}_c w$ (transitivity).

It is easy to see that $\mathcal{B}_c$ is an equivalence relation. If U is an equivalence class in the quotient set $CONSTRS(c)/\mathcal{B}_c$, every constraint in U might interact with any other constraint in the same class during the configuration process of c. On the contrary, given the *exclusiveness assumption on parts*, if $V \in CONSTRS(c)/\mathcal{B}_c$ is different from U, it means that in c the constraints belonging to V don't interact in any way with those in U. This means that the problem of configuring c by taking into consideration $CONSTRS(c)$ can be split into the set of independent subproblems of configuring c by considering the set W of constraints, for each $W \in CONSTRS(c)/\mathcal{B}_c$.

4 Configuration via Decomposition and Distribution

As said in the section above, $bound$ relation can be used for singling out independent subproblems in the configuration process. A sequential algorithm for configuration exploiting decomposition is described in [8]. In the present paper we describe a general configuration technique which exploits the decomposition mechanism for distributing independent subproblems among a set of configurators working in parallel.

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.