Cooperating Configuration Agents Supporting Supply Chain Integration of Customizable Products

A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker
Institut für Wirtschaftsinformatik und Anwendungssysteme,
Universität Klagenfurt, A-9020, Austria,
{felfernig, friedrich, jannach, zanker}@ifi.uni-klu.ac.at

Abstract

The integration of configuration systems for supporting supply chain management of configurable products and services is still an open research issue. Current configurator approaches are designed for solving local configuration problems, but there is still no support for the integration of different configuration systems. In order to facilitate cooperative configuration we employ configuration agents capable of managing requests and posting configuration subtasks to remote agents. For integrating different knowledge representation formalisms of configuration agents we construct common ontologies by employing broadly used configuration domain specific modeling concepts.

1 Introduction

Knowledge-based configuration systems have a long history as a successful AI application area and today form the foundation for a thriving industry (e.g. telecommunication, automotive industry, computer industry, or financial services). Current configurator approaches are designed for solving local configuration problems, but there is still no support for the integration of different configuration systems. Our framework for distributed configuration consists of configuration agents along the value chain of products (see *Figure* 1). Each configuration agent can be seen as an autonomous acting entity, which receives requests for customizable products from its communication interface and demands customizable parts from supplier configuration agents. In such a distributed environment each configuration agent has its knowledge base to provide consistent (partial) configurations for the final product or service. In order to integrate different knowledge representation formalisms we employ common ontologies consisting of *shared configuration knowledge* and a *communication protocol*. For example, in order to integrate a hardware and a software configuration agent both must agree upon a communication protocol and a shared set of component types (e.g. operating system, cpu, other shared knowledge) and constraints between those component types (e.g. *operating system X is incompatible with cpu Y*). For designing a shared configuration model we employ domain specific modeling concepts broadly used in the configuration domain. In [1] we have defined such concepts in terms of UML stereotypes (Unified Modeling Language [6]) and provide translation rules for transforming these concepts into an executable logic representation. UML is widely used in the software development domain thus supporting acceptance of these concepts for a large application community. The mentioned modeling concepts are also employed for designing local configuration knowledge bases (see [1]).

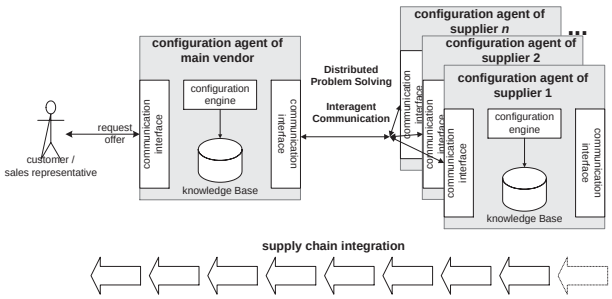

Figure 1. Distributed configuration

2 Knowledge Interchange between Configuration Agents

In practice, configurations are built from a predefined catalog of component types for a given application domain [3]. These component types (*types*) are described through a set of properties (*attributes*), and connection points *(ports)* representing logical or physical connections to other components. This information and additional constraints on

legal configurations are contained in the domain description (*DD*) of a configuration problem. The actual configuration problem has to be solved according to a set *SRS* (system requirements specification). In order to represent distributed configuration problems we define *DD* as $\bigcup DD_i$, *SRS* as $\bigcup SRS_i$, where $i \in \{1..n\}$ and n is the number of configuration agents. A configuration is described through a set of components *(COMPS)*, connections between components *(CONNS)*, and corresponding attribute valuations *{(ATTRS)}*. In the case of a distributed configuration *COMPS*= $\bigcup COMPS_i$, *CONNS*= $\bigcup CONNS_i$, *ATTRS*= $\bigcup ATTRS_i$. Based on this characterization of a configuration task we give a definition of a *Consistent Distributed Configuration* as follows:

If (DD, SRS) is a configuration problem, and COMPS, CONNS, and ATTRS represent a configuration result, the distributed configuration is consistent iff DD $\cup$ *SRS* $\cup$ *COMPS* $\cup$ *CONNS* $\cup$ *ATTRS can be satisfied.*

Additionally we have to specify that COMPS includes all required components, CONNS describes all required connections, and ATTRS includes a complete value assignment to all variables in order to achieve a *Complete Distributed Configuration*[1]. In order to assure completeness and correctness of the distributed configuration w.r.t. the overall configuration task the following axiom must hold:

DD $\cup$ *SRS* $\cup$ *COMPS* $\cup$ *CONNS* $\cup$ *ATTRS* $\cup$ AX_{comp} *is consistent iff* $\forall i$*:* $DD_i \cup SRS_i \cup COMPS \cup CONNS \cup ATTRS \cup AX_{comp}$ *is consistent.*

A consistent and complete Distributed Configuration is called a *Valid Distributed Configuration*. In order to calculate solutions for a given distributed configuration task we employ *asynchronous backtracking* [8], which offers the basis for bounded learning strategies supporting the reduction of search efforts. This is of particular interest for integrating configurators. Configurators send configuration requests to their solution providing partners. These partners eventually discover conflicting requirements *(nogoods)* which are communicated back to the requesting configurator, thus supporting the efficient revision of requirements and design decisions. Beside the *content part* of a common ontology a *communication protocol* must be provided in order to guide the distributed search process.

Asynchronous backtracking provides a simple protocol based on the exchange of conflicting requirements (*nogood* messages) and changed shared knowledge (*ok?* messages). Some agents are responsible for the update of the shared knowledge (value sending agents), other agents are responsible for adapting their knowledge bases regarding the new requirements if possible (constraint evaluating agents).

[1]This is accomplished by additional logical sentences AX_{comp} (see [3]) which can be generated using DD.

3 Prototype Environment

The concepts sketched in this paper are implemented in a prototype environment for the construction of cooperative configuration agents. For the design of product configuration models we employ the CASE tool Rational Rose. Our translation tool uses a XMI (XML Metadata Interchange [5]) representation (generated from Rational Rose models) as input and either generates C++ code which includes the ILOG configuration libraries or an intermediate representation which can be imported into the SAP configurator using the standard data transportation mechanisms of SAP. We have evaluated our approach on real world problems from the domains of private telephone switching systems and automotive industry and noticed a significant reduction of development efforts. An extended version of a distributed PC configuration problem is implemented using ILOG configurators which are implemented as CORBA objects. The communication between the configurators is realized using simple KQML [4] performatives. The message content is represented as an XML ([7]) document containing a set of requirements (conforming the shared configuration knowledge) or a set of incompatible requirements (incompatible components). In order to validate the knowledge base generated from the UML model we employ consistency-based diagnosis [2], where (in)complete configurations serve as test cases for the configurator.

References

[1] A. Felfernig, G. Friedrich, and D. Jannach. UML as domain specific language for the construction of knowledge-based configuration systems. In *11th International Conference on Software Engineering and Knowledge Engineering*, pages 337–345, Kaiserslautern, Germany, 1999.

[2] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-Based Diagnosis of Configuration Knowledge Bases. In *AAAI Workshop on Configuration, Technical Report WS-99-05*, pages 41–47, Orlando, Florida, 1999.

[3] G. Friedrich and M. Stumptner. Consistency-based Configuration. In *AAAI Workshop on Configuration, Technical Report WS-99-05*, pages 35–40, Orlando, Florida, 1999.

[4] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12,3:36–56, 1991.

[5] OMG. XMI Specification. *www.omg.org*, 1999.

[6] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[7] W3C. Extensible Markup Language (XML). *www.w3.org*, 1999.

[8] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem. *IEEE Transactions on Knowledge and Data Engineering*, 10,5:673–685, 1998.