A Framework for the Development of Cooperative Configuration Agents

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 to support supply chain integration of configurable products is still an open research issue. Current configurator approaches are designed for solving local configuration problems but there is no support for the integration of multiple configuration systems. In order to facilitate distributed configuration of customizable products we employ cooperating configuration agents capable of managing requests and posting configuration subtasks to remote configuration agents. For integrating different knowledge representation formalisms of configuration agents we apply broadly used configuration domain specific modeling concepts to design shareable ontologies which can be interpreted by other agents. These concepts are defined as UML (Unified Modeling Language) stereotypes which can be automatically translated into a configuration agent's knowledge representation.

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). A new challenge for the employment of configuration systems is their integration into the value chain of one or more product families (supply chain integration). While supply chain management of standardized, mostly well defined products is quite well supported by upcoming electronic marketplaces, auctioning mediators, or automatic purchasing systems, current configuration technology does not offer concepts and tools supporting supply chain integration of configurable products.

Configuration systems have likewise progressed from their rule-based origins to the use of higher level representations such as various forms of constraint satisfaction, description logics, or functional reasoning. This variety of representation formalisms causes incompatibilities between configurators and the question has to be answered how to integrate these systems. The increased use of knowledge-based configurators in various application domains as well as the increasingly complex tasks tackled by such systems ultimately lead to both the knowledge bases and the resulting configurations becoming larger and more complex. In this context, effective knowledge acquisition is crucial since configurator development

R. Loganantharaj et al. (Eds.): IEA/AIE 2000, LNAI 1821, pp. 24–33, 2000.

time is strictly limited, i.e. the product and the corresponding configuration system have to be developed in parallel.

For the development of configuration agents, we show how these challenges can be met by using a standard design language (UML - Unified Modeling Language [12]) as notation in order to simplify the construction of a logic-based description of the domain knowledge as well as the construction of a common interpretable ontology which serves as basis for inter-agent communication. UML is widely applied in industrial software development as a standard design language. We employ the extension mechanism of UML (stereotypes) in order to express configuration domain specific modeling concepts. The configuration agent development process is represented in Fig. 1. First we select a shared product model represented in UML (provided by a supplier) and a communication protocol, which together constitute a shared ontology for cooperative configuration. The shared product model represents those component types and constraints of a supplier‘s configurable product, which should be visible for other configuration agents, i.e. can be integrated in their local product model. The shared product model is imported and integrated into the local product model which is designed in UML (1). After syntactic checks of the correct usage of the modeling concepts (2) this model is non-ambiguously translated into logical sentences (3). The resulting configuration agent is validated by the domain expert using test runs on examples (4). In case of unexpected results, the product model can be revised (1), otherwise it can be employed in productive use. Additionally, if the generated configuration agent acts as a supplier as well, those components and constraints, which should be visible to customer agents must be made public, i.e. provided as a shareable product model (5).

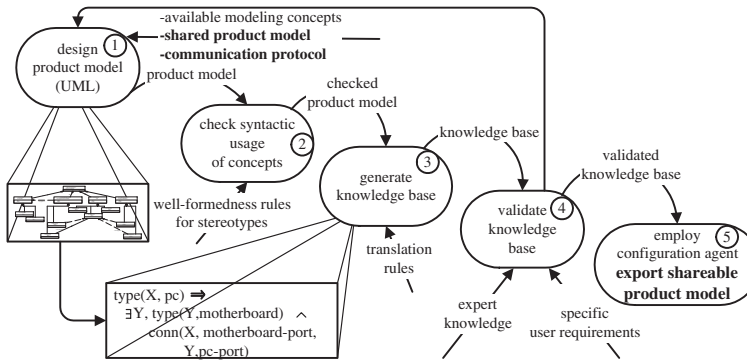

Fig. 1. Configuration Agent Development Process

The paper is organized as follows. First we characterize a configuration task and give a formal definition of a distributed configuration task (Section 2). By giving an example from the domain of hardware/software configuration we discuss commonly used concepts for modeling configuration domains and their definition in UML. Further we show how these concepts can automatically be trans-

lated into an executable logic representation (Section 3). In Section 4 we show how to integrate two configuration agents using XML (Extensible Markup Language) as communication language. In Section 5 we discuss the structure of our prototype system. Sections 6 and 7 contain related work and conclusions.

2 Distributed Configuration Task

In practice, configurations are built from a predefined catalog of component types of a given application domain. These component types (*types*) are described through a set of properties (*attributes*), and connection points (*ports*) representing logical or physical connections to other components. The *attributes* have an assigned domain (*dom*). The domain description (*DD*) of a configuration task contains this information and additional constraints on legal configurations. The actual configuration problem has to be solved according to a set of specific user requirements (*SRS*). Based on this definition of a local configuration task [5] we define a *Distributed Configuration Task* through the following sets of logical sentences.

- $DD = \bigcup DD_i$, where DD_i is the *DD* of agent i ($i \in \{1..n\}$ and n is the number of configuration agents[1]).
- SRS $= \bigcup SRS_i$.

A configuration result is described through sets of logical sentences *(COMPS, ATTRS, CONNS)*. In these sets, the employed components, attribute values, and established connections of a concrete customized product are represented.

- $COMPS = \bigcup COMPS_i$, where $COMPS_i$ represents sets of literals of the form *type(c,t)*. *t* is included in the set of *types* defined in DD_i. The constant *c* represents the identifier of a component.
- $CONNS = \bigcup CONNS_i$, where $CONNS_i$ represents sets of literals of the form *conn(c1,p1,c2,p2)*. *c1, c2* are component identifiers from $COMPS_i$. *p1 (p2)* is a port of the component *c1 (c2)*.
- $ATTRS = \bigcup ATTRS_i$, where $ATTRS_i$ represents sets of literals of the form *val(c,a,v)*, where *c* is a component identifier, *a* is an attribute of that component, and *v* is the actual value of the attribute (selected from the domain of the attribute).

The concept of a *Consistent Distributed Configuration* is defined as follows.

Definition: Consistent Distributed Configuration. *If (DD, SRS) is a configuration problem and COMPS, CONNS, and ATTRS represent a configuration result, then the configuration is consistent exactly iff DD* $\cup$ *SRS* $\cup$ *COMPS* $\cup$ *CONNS* $\cup$ *ATTRS can be satisfied.*

We specify that *COMPS* includes all required components, *CONNS* describes all required connections, and *ATTRS* includes a complete value assignment to all

[1] Note that DD_i can also include imported product models.

variables in order to achieve a complete distributed configuration[2]. Let AX_{comp} be the additional sentences for completeness purpose.

In order to assure completeness and correctness of the distributed configuration w.r.t. the overall configuration task the following sentence must hold:

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

This sentence is fulfilled if we allow in *DD* only sentences using *type, conn*, and *val* literals since $COMPS \cup CONNS \cup ATTRS \cup AX_{comp}$ is a complete theory w.r.t. these literals. A distributed configuration, which is consistent and complete w.r.t. the domain description and the customer requirements, is called a *Valid Distributed Configuration.*

In order to calculate solutions for a given distributed configuration task we employ *asynchronous backtracking* [14], which offers the basis for bounded learning strategies supporting the reduction of search efforts. This efficient revision of requirements and design decisions is of particular interest for integrating configurators, since supplier agents eventually discover conflicting requirements (nogoods) which must be communicated back to the requesting configurator.

3 Construction of Configuration Agents

In order to coordinate the distributed configuration process and to permit knowledge interchange between configuration agents, parts of the agents' knowledge bases must be shared. For representing the shared configuration knowledge we employ configuration domain specific modeling concepts defined as UML stereotypes. Note that UML is no precondition for designing product structures but alleviates construction and maintenance of these models as has been shown in [4]. Configuration models, that are designed using these concepts can be translated into the logic representation of a specific configuration agent.

The shared knowledge together with an agreed upon communication protocol are the constitutive parts of a common ontology which is the prerequisite for the communication between configuration agents. In [2] different interpretations of ontologies in AI are discussed. First, ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge. They provide the potential terms for describing the knowledge about the domain. Second, ontologies are a body of knowledge describing some domain using a representation vocabulary. Exactly this second interpretation is represented in our approach to distributed configuration. In the following we shortly discuss the configuration domain specific modeling concepts expressed as UML stereotypes and their translation into an executable logic representation[3].

[2] This is accomplished by additional logical sentences which can be generated using the domain description (see [5] for more details).

[3] A detailed discussion on the translation rules can be found in [4].

We do not employ any configuration agent specific knowledge representation, but propose a translation into the component port model formalism, which is well established for modeling and solving configuration problems [7]. In general, consistency-based tools based on this model can use the logic theory derived from the UML model. Fig. 2 shows the knowledge bases of two configuration agents (hardware and software configuration agent). The shared configuration knowledge is identified by emphasized components and constraints between these components (*software-package,development-environment,desktop-publishing, os,os-1,os-2,os-3,text-editor*). Exactly these components are incorporated into the local configuration model of the hardware configuration agent. In order to understand the graphical notation we shortly define the basic modeling concepts specific for the configuration domain and show the translation into the component port representation.

– **Component Types** We use a stereotype class for representing component types since some limitations on these classes have to hold, e.g. there are no methods, attributes are limited to simple data types and enumerations. The component types of the hardware configuration agent in Fig. 2 are translated into the following *types, attributes,* and *ports* definitions:

types={pc, floppy-unit, hd-unit, scsi,...}.
attributes(pc)={maxprice},...
dom(pc, maxprice)={0..10000},...
ports(pc)={floppy-unit-port, hd-unit-port, motherboard-port,...}.

– **Requires** *X requires Y* means, if *X* is part of the product then *Y* must be part of the product too. The constraint *motherboard-2 requires p-II* is translated as follows:

$$\text{type(ID1, motherboard-2)} \Rightarrow \exists(\text{ID2})\ \text{type(ID2, p-II)} \wedge \text{conn(ID2, motherboard-port, ID1, cpu-port)}.$$

– **Incompatible** *X incompatible Y* denotes the fact that two components cannot be used within the same configuration. The *incompatible* relation is defined as a binary relation with a multiplicity of *1..1* in the UML model. The constraint *motherboard-1 incompatible p-I* is translated as follows:

$$\text{type(ID1, motherboard-1)} \wedge \text{type(ID2, p-I)} \wedge \text{conn(ID2, motherboard-port, ID1, cpu-port)} \Rightarrow \text{false}.$$

– **Resources** They impose additional constraints on the possible product structure. Some components can contribute to a resource whereas others are consuming a specified quantity from the resource. In an actual configuration the resources must be balanced, i.e. the consumed resources must not exceed the provided resources. The contribution and consumption of a resource is modeled through relations *consumes* and *produces*. In our example, the *price* of a *pc* component must be less than or equal to the maximum price imposed by the customer (*maxprice* in Fig. 2).

– **Ports and Connections** Not only the quantity and kind of the employed components can be important, but also how different components are connected to each other. Components can be connected through connection points (ports). One port can only be connected to exactly one other port, e.g. *at-bus-connector* is connected to *at-bus-slot* (Fig. 2).

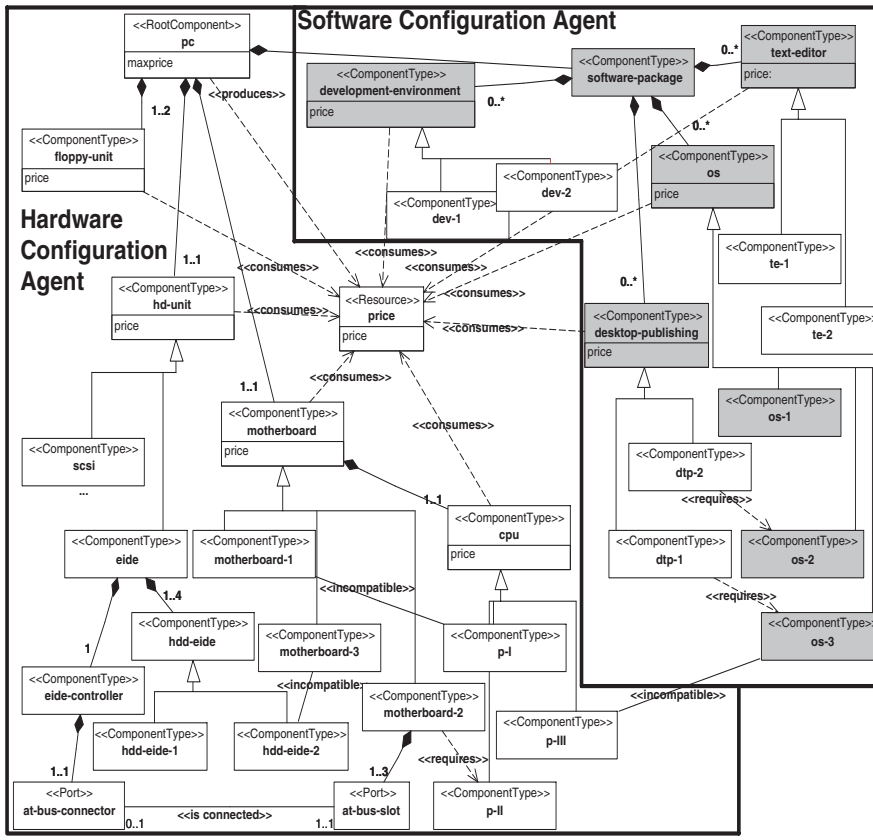

Fig. 2. Configuration Knowledge Bases

– **Additional modeling concepts and constraints** The discussed modeling concepts have shown to cover a wide range of application areas for configuration [11]. Despite this, some application areas might have a need for special modeling concepts not covered so far. UML stereotypes are an extension mechanism with which further modeling concepts can be built into UML by adapting the corresponding meta model. In order to define restrictions on the usage of the stereotypes well-formedness rules can be defined in OCL (Object Constraint Language), which is an integrative part of UML.

Beside the definition of the shared configuration knowledge, the model of the configuration agents' dynamics is the second major part of a common ontology. In order to model the allowed states of an agent's configuration process we employ state charts. State transitions are triggered by messages from (to) remote configuration agents. A simple example of an agent communication protocol is given in Fig. 3. First, the local agent (B) receives a request for configuration from a remote configuration agent denoted as agent A (1). B either calculates a consistent configuration and returns the result to A (2) or posts a configuration subtask to agent C (3), or is not able to find a consistent solution (6). If B receives a consistent subconfiguration from C (4), it continues the local search process and returns a found solution to A. If C does not find a consistent solution (5), B tries to calculate an alternative solution considering the nogoods from C[4]. If A accepts the solution received from B, this acceptance is communicated to C (7), otherwise A does not accept the found solution (8).

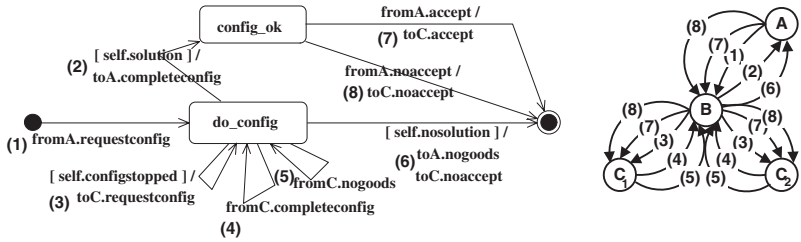

Fig. 3. Agent Communication Protocol

4 Example: Distributed Configuration Using XML

In order to show the interaction process between configuration agents based on the shared ontology discussed in Section 3 we give a simple example for configuring computer systems. For supporting knowledge interchange between the involved configuration agents we employ XML (Extensible Markup Language [13]) as agent communication language which is a standard for information exchange in the Internet. In order to enable the exchange of configuration knowledge via XML we need to translate the common ontology modeled in UML (see Section 3) into XML DTDs which define the structure of exchangeable XML messages between the configuration agents. DTDs are primarily used for checking the conformity of received XML documents containing configuration data with the defined ontological commitments.

In the following example XML documents represent the relevant configuration information which is exchanged between the agents. While agent B is

[4] Nogoods represent conflicting value instantiations causing contradictions in C's knowledge base.

capable of configuring computer hardware, agent C calculates software configurations. In the following scenario B is contacted by a customer for configuring a *pc* system. C is responsible for configuring the *software-package* since detailed knowledge about software configuration is not available for B.

First, B is contacted by the user via the user interface. The customer orders one *floppy-unit*, one *eide-unit* with one *hdd-eide-1*, a *motherboard-2*, a *p-II cpu*, and a *software-package* consisting of an *os-1* and a *desktop-publishing* system. The *floppy-unit*, *motherboard*, *eide-unit*, and *cpu* are configured without communicating with other configuration agents. For configuring the *software-package* B contacts C. The components *os-1* and *desktop-publishing* are sent as initial (partial) configuration to C as follows [5].

¡requestconfig¿
¡software-package¿ ¡os-1/¿ ¡desktop-publishing/¿
¡/software-package¿
¡/requestconfig¿

C tries to expand the partial configuration and detects the incompatibility between *os-1* and *desktop-publishing* since a *desktop-publishing* system either requires an *os-2* or an *os-3*. The following message is sent from C to B.

¡nogoods¿
¡software-package¿ ¡os-1/¿ ¡desktop-publishing/¿
¡/software-package¿
¡/nogoods¿

B includes the nogoods as constraints into its local knowledge base. Since B is unable to calculate a consistent solution, the nogoods are presented to the customer. The customer decides to order *os-1* without a corresponding *desktop-publishing* system, i.e. the following message is sent from B to C.

¡requestconfig¿
¡software-package¿ ¡os-1/¿
¡/software-package¿
¡/requestconfig¿

C accepts B‘s configuration and sends the following message to B.

¡completeconfig¿
¡software-package¿ ¡os-1¿¡price¿100¡/price¿¡/os-1¿
¡/software-package¿
¡/completeconfig¿

B accepts the configuration by sending an ¡accept/¿ message to C.

5 Prototype Development Environment

In order to realize the concepts discussed in this paper we have implemented a prototype development environment for cooperative configuration agents. The

[5] Messages are divided into three different layers conforming the Knowledge Sharing Effort (KSE) model of knowledge interchange (content, message, and communication layer). For reasons of space limitations we only present portions of the communicated XML documents.

CASE tool *Rational Rose* is employed for specifying the agents product model as well as the communication protocol. For alleviating the model interchange between different modeling environments we translate the agent models specified in *Rational Rose* into a neutral XMI (XML Metadata Interchange [10]) representation which is further translated into the knowledge representation of the corresponding configuration agents. In order to calculate configurations we employ an industrial-strength configuration engine (*ILOG Configurator C++ library*). Configuration agents are implemented as distributed COM (Component Object Model) objects. The translation from XML to C++ is done by integrating the IBM *XMLC* XML parser.

6 Related Work

In order to solve large scale configuration problems, [3] propose a network of design agents that represent part catalogs and design constraints. The whole problem is decomposed into sub-problems of manageable size which are solved by the agents. Designing large scale products requires the cooperation of a number of different experts. In the SHADE (Shared Dependency Engineering) project [9] a KIF [8] formalism was used for representing engineering ontologies. Giving an example of a spring construction, the integration of a project engineering agent responsible for the definition of the component hierarchy and basic properties of mechanical components, a spring design agent responsible for the design of the detailed technical structure and an optimization agent responsible for optimization tasks is shown. The main focus of [3] is on efficient distributed design problem solving, [9] concentrate on the integration of different design views, whereas our concern is to provide effective support of distributed configuration problem solving , where knowledge is distributed between different agents having a restricted view on the whole configuration process.

For the exchange of knowledge between agents in a Web-based environment XML is an adequate representation language since it is an evolving standard for information interchange in the Internet. XML standards are defined for various application areas, e.g. XMI is a standard for exchanging models defined in a modeling language conforming the MOF standard (Meta Object Facility).

Automated generation of logic-based knowledge bases through translation of domain specific modeling concepts expressed in terms of a standard design language like UML has not been discussed so far. Comparable research has been done in automated and knowledge-based Software Engineering [6]. [1] define a formal semantics for object model diagrams based on OMT for supporting assessments of requirement specifications. Our work is complementary since our goal is the generation of executable logic descriptions.

7 Conclusions

In this paper we have presented an UML-based environment for the construction of cooperative configuration agents. The integration of businesses by Inter-

net technologies boosts the demand for integrative coordination of knowledge-based systems, such as configuration systems which have to move from standalone to cooperative systems. In this context the agent paradigm is a quite suitable approach since each configurator can be seen as an autonomous acting entity, receiving requests and demanding customizable parts. A precondition for integrating configuration agents is the definition of a common ontology using concepts interpreted in the same way by the configuration agents. In order to integrate configuration agents in an Internet environment XML is the appropriate language for exchanging knowledge between agents, since it is a wide spread standard and easy to integrate in Web-based applications.

References

1. R.H. Bourdeau and B. Cheng. A Formal Semantics of Object Models. *IEEE Transactions on Software Engineering*, 21,10:799–821, 1995. 32
2. B. Chandrasekaran, J. Josephson, and R. Benjamins. What Are Ontologies, and Why do we Need Them? *IEEE Intelligent Systems*, 14,1:20–26, 1999. 27
3. T.P. Darr and W.P. Birmingham. An Attribute-Space Representation and Algorithm for Concurrent Engineering. *AIEDAM*, 10,1:21–35, 1996. 32
4. A. Felfernig, G. Friedrich, and D. Jannach. UML as domain specific language for the construction of knowledge-based configuration systems. In *Proceedings of SEKE'99*, pages 337–345, Kaiserslautern, Germany, 1999 [6]. 27
5. G. Friedrich and M. Stumptner. Consistency-Based Configuration. In *AAAI Workshop on Configuration, Technical Report WS-99-05*, pages 35–40, Orlando, Florida, 1999. 26, 27
6. M. Lowry, A. Philpot, T. Pressburger, and I. Underwood. A Formal Approach to Domain-Oriented Software Design Environments. In *Proceedings 9th Knowledge-Based Software Engineering Conference*, pages 48–57, Monterey, CA, 1994. 32
7. S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings of the 11th IJCAI*, pages 1395–1401, Detroit, MI, 1989. 28
8. 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. 32
9. G.R. Olsen, M. Cutkosky, J.M. Tenenbaum, and T.R. Gruber. Collaborative Engineering based on Knowledge Sharing Agreements. In *Proceedings of ACME Database Symposium*, pages 11–14, Minneapolis, MN, USA, 1994. 32
10. Object Management Group (OMG). XMI Specification. *www.omg.org*, 1999. 32
11. H. Peltonen, T. Männistö, T. Soininen, J. Tiihonen, A. Martio, and R. Sulonen. Concepts for Modeling Configurable Products. In *Proceedings of European Conference Product Data Technology Days*, pages 189–196, Sandhurst, UK, 1998. 29
12. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. 25
13. W3C. Extensible Markup Language (XML). *www.w3.org*, 1999. 30
14. M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem. *IEEE Transactions on Knowledge and Data Engineering*, 10,5:673–685, 1998. 27

[6] A revised and extended version of this paper will appear in the International Journal of Software Engineering and Knowledge Engineering (IJSEKE).