Customer-Adaptive and Distributed Online Product Configuration in the CAWICOMS Project

Abstract

This paper reports on the goals and first results of the CAWICOMS[1] project. The main goals of this EU-funded project are to provide technologies both for customer-adaptive Web-interfaces for the configuration of mass-customized products as well as for the integration of configuration systems along the supply-chain.

Within this paper we first motivate the demand for personalized and adaptive Web-interfaces for the domain of product configuration. In addition, we sketch scenarios where product configuration takes place at several stages in the supply chain and the involved configuration systems have to cooperatively solve a distributed configuration task.

An overview of the CAWICOMS architecture is given as well as first results on distributed problem solving and the integration into the surrounding enterprise's software infrastructure.

1. Introduction

Web-based product configuration tools enable businesses to market complex customizable products and services by using the new technologies of electronic commerce, whereby customers can tailor the configurable products according to their specific needs and requirements. Although there is a long history of knowledge-based product configuration technology [13], typically, these stand-alone systems do not meet the new requirements imposed by online (sales) configuration in a Web of cooperating product and service providers.

The CAWICOMS project is carried out in order to overcome those shortcomings of current configuration technology and is conducted by an international consortium consisting both of partners from academia and industry. The main objectives of the project fall into two categories, namely adaptiveness of the interface and distributedness of the configuration task. We will shortly sketch the basic environment that makes these aspects become important features for future configuration technology.

Adaptive user interfaces: When selling complex customizable products online, there may be a wide range of different classes of users of the configurator that differ in properties such as skills, needs and knowledge levels. In current Web-based configurators there is typically only one standard interface having a predefined interaction style that can not be tailored to these different types of users. A typical example for such a possible adaptation is given, if the system behavior depends on the level of experience of the user. In case of an experienced one, the product may be configured at some detailed technical level, whereas a novice user will possibly only enter some high-level features and will have a need for more guidance and extra help during the configuration process.

An adaptive configuration tool will classify the current user with respect to different properties (e.g., expertise or interest in product characteristics such as reliability) and generate an interface that adapts flexibly according to this classification. The goal is that the online selling (configuration) system behaves like a real sales person to the extent that it changes its own behavior while interacting with different customers by eliciting information about their requirements and making specific suggestions in a personalized way.

Distributed configuration: The business model of the digital economy (*B2B e-commerce*) shows a trend towards specialized solution providers that are organized in a supply chain. The final product offered to the customer consists of an assembly of subproducts provided by the individual specialists and manufacturers. This fact intensifies the demand for supply chain integration which is quite well known for standardized products but for the case of complex and configurable products it is still an open issue. Nowadays' configuration and supply-chain technology does not provide adequate means for the cooperation between different configuration systems. Based on the fact that the individual subproducts of the suppliers may all be configurable there is an increasing demand for solutions that integrate different configurators in order to assemble a complete system that satisfies both the customer requirements and the given constraints on legal product constellations for the overall solution. Such integration is needed because we cannot assume a central point of (configuration) knowledge, e.g., because of confidentiality issues. Furthermore, for the integration we have to face the problem of heterogeneity, i.e., we cannot assume that the participating configuration systems rely on the same knowledge representation or problem solving mechanisms. Therefore, we have to define adequate protocols and knowledge exchange mechanisms between such systems.

Application scenarios: The work in CAWICOMS is driven by two major real-world application scenarios provided by the industrial partners.

- Configuration of private telecommunication solutions,
- Configuration of virtual private networks (VPN).

For presentation purposes, we will shortly sketch the requirements for distributed problem solving and adaptivity for the domain of the telecom switching systems.

[1] This work takes place with support of the IST-Programme of the European Union (Contract: IST-1999-10688 CAWICOMS). CAWICOMS is the acronym for *Customer-adaptive web interface for the configuration of products and services with multiple suppliers*.

Our partner (or one of the resellers) sells telecommunication solutions to its customers that include e.g., in-house telephony, switching of calls to the local telephone exchange for outside calls, end-user devices, as well as add-on products like *voice-mail* or *voice-over-IP* applications. While the main switch is manufactured in-house (and the configuration knowledge is completely known) the add-on products mentioned above are provided by suppliers (or another organizational unit) and are in turn customizable. The detailed knowledge of how to configure the supplier's products is only known by the supplying organization and incorporated in a *local* configuration knowledge base. However, the main switch and the add-ons cannot be configured independently, because there are some interdependencies between these subproducts. For instance, when adding the *voice-over-IP* application, additional hardware has to be installed and connected to the main switch onto one of the free slots. Furthermore, installing the additional hardware may e.g., require a special power supply. Note that it is not possible nor reasonable to integrate the complete knowledge into one large knowledge base due to security or maintainability reasons, knowing that there are different points of knowledge and expertise involved. Our goal is therefore to find a way of *knowledge sharing* based on a common ontology of configuration and domain concepts. Note that it is important for the customer to have only one single point of interaction (configuration) according to the marketing paradigm of selling *solutions* and not *products.*

The telecommunication scenario also shows some interesting challenges concerning the personalization of the user interface. As already mentioned, the configuration system may be used by heterogeneous users, having different expertise and skill levels: small switching systems may be sold directly, i.e., the actual end-user configures the switch and will need a simple interface with advanced user guidance. For larger systems, a sales representative or sales engineer will use the system for different purposes. First, during the *quotation phase*, only a few high-level parameters have to be entered that are possibly personalized to the preferences of the user, e.g., the sequence of interaction, level of additional technical explanations or for instance country-specific defaults. In the successive order processing phase technical details, e.g., physical components and their structure, have to be specified for production. Consequently, the configurator interface has to provide different levels of details and default values. Non-expert users might not understand these technical details; therefore, the system could hide them and calculate default configurations, on the basis of the recognized customer’s preferences.

The rest of the paper is organized as follows. In Section 2, we will shortly describe the specific adaptivity requirements as well as applicable techniques for the domain of product configuration and online selling. Section 3 gives a definition for a distributed configuration task based on a logic theory. Afterwards, we sketch the overall architecture of the CAWICOMS framework, discuss the technology used for the different components of the architecture and describe the requirements for the integration into the existing software infrastructure. The final section contains conclusions and describes future work to be done in the CAWICOMS project.

2. Personalized Interaction

Based on the specific requirements of personalized interaction in the product configuration domain, we will now outline the personalization techniques used in the CAWICOMS framework. In particular we show how the interaction with the user can be specifically tailored based on estimates of the user's properties (characteristics).

2.1 Estimating user properties

Basically, we will estimate the users' characteristics (the user model) by observing their behavior in different situations during several(interactive) configuration sessions. As inference mechanism, we will employ Bayesian networks [9], whereby the details of these inferences is beyond the scope of this paper. The relevant user properties that can be utilized for personalization of the interaction fall into two categories: *user's interest* and *user's knowledge*.

2.1.1 User’s interests

For determining the user's interests and exploiting them for personalization, we assume that the configurable artefact itself can be characterized by a set of high-level properties (that we call *dimensions*) that are not part of the knowledge base of the configurator itself. Typical dimensions for a telecommunication switch are for instance *reliability*, *security capabilities*, or *performance*. For each user of the system we will try to determine a relative weight for each dimension describing the user's interest, which is done by ascribing *Multi-Attribute-Utility Theory (MAUT)* [14] as evaluation process to the user. Later on, we utilize those estimates of the user interests to generate personalized defaults during the interactive configuration session. This can be done by relating these high-level dimensions to properties of the configurable system, e.g., the *reliability* dimension may (among others) be related to the type of the power supply to be suggested by the configurator.

We use the user's observable behavior as evidence for inferring his/her specific interests and for finding the relative weights of the different dimensions, whereby the estimates are maintained in the user model in form of a probability distribution. The following user actions can be exploited during an interactive session.

- Self assessment of the user: the user describes her/his interests for the individual dimensions manually (e.g., being questioned by the configuration system).
- Accepting/rejecting a solution: if the user accepts or rejects a proposed configuration, we can determine the rating of the proposed configuration according to the high-level dimensions and adapt our current user model, i.e., if the user rejects a very reliable system we may infer that reliability is not an important feature.

- Input of a parameter value: Whenever the user enters a choice for a configurable parameter (whether changing a proposed value or entering a new one), and the parameter is related to some dimension, we may update the user model. As an example consider a user selecting an optional *encryption package* for the switch, we may revise the current user model and assume that the user's interest in the dimension *security* will be high.

2.1.2 User's knowledge

The other category of user properties that can be exploited for personalization is the user's domain expertise (*knowledgeability*). Again, we try to estimate the user's knowledgeability based on his/her observable behavior and later on utilize the user model during the interaction process. (See [6] for details on how to estimate the knowlegeability of a person). The representation of the user's expertise is twofold.

Knowledgeability by dimension: Usually, the user's knowledgeability is not an undifferentiated feature. For example, in the domain of cars, some users may know very much about aspects regarding the *environmental-friendliness* of the car, but they may not be familiar with the features which have implications on its *sportiness*. Therefore the user's knowledgeability for each dimension has to be estimated. Note, that these dimensions could be the same dimensions that are used to measure the interests of the user.

Furthermore, we relate configurable parameters to these dimensions and attach a weight to the configurable parameters describing the technical difficulty, i.e., we define whether some parameter is a low-level technical feature that can only be configured by a technical expert or if the selection of a parameter value can be done also by a sales-representative or end-customer. Based on an estimate of the user's knowledgeability in a certain dimension and the difficulty of a configurable parameter, it can be estimated whether the current user will be able to specify a value (knows the implications of selecting a certain value) or if e.g., additional help should be displayed or the configurator should propose a default value.

Knowledge of parameters: Beside the estimates on knowledgeability in general dimensions we can also represent information in the user model about estimated knowledgeability for a certain configuration parameter. We may increase the knowledgeability estimate for a particular parameter, if the user changes this specific parameter value manually (in several configuration sessions) or is explicitly informed about the meaning of a parameter (e.g., by an additional help screen).

2.2 Adapting the interface

The personalization of the user interface is based on the exploitation of personalization strategies, on the basis of the user's interests and knowledgeability. As the user interface is dynamically generated during the interaction with the customer, the level of detail addressed during the configuration of the product can be continuously adapted to the most recent hypotheses about her/his knowledgeability and interests. Several aspects of the interaction can be customized: For instance, the layout of the interface, the amount of information to be displayed and the type of questions asked during the configuration of the product (see [1] for the definition of personalization strategies for a B2C application). In this description, we will focus on the last aspect, which strongly depends on the user's knowledgeability and interests and is critical to the usability of the configuration system.

The basic idea is that the information in the user model and the knowledge about the relation to the product model should be exploited as much as possible to predict the user's choices during the configuration process. The ultimate goal is to carry on the configuration process while limiting the number of questions asked to the user (therefore gaining shorter interactions) and avoiding difficult questions, which the user might not be able to answer.

Another important criterion to decide whether a direct question can be avoided concerns the *criticality* of a parameter during the configuration process. This level of criticality is also defined for the individual configurable parameters in the personalization component.

- If a parameter is extremely critical, the system can not take the responsibility for a decision on its values and the user must be asked about it. In this case, if the user is not assumed to know the parameter, the system can support his/her decision by providing additional information about the meaning of the parameter and the impact of its values on general properties of the product (its dimensions).
- If the parameter is not too critical, the system can determine a value by applying defaults, given the information about the user's interests, or by eliciting indirect information from her/him. Defaults can be inferred by the system on the basis of the user's interests in certain product dimensions. For example, if the user model predicts that the user is interested in a high reliability, two power supplies, instead of one, would be the default value for the parameter.

Of course, a user should be enabled to define personal default values manually. The user-defined default values will then override the default values estimated by the system.

Finally, indirect questions can be exploited when the predictions of the user model are not certain enough to support the exploitation of defaults. In this case, the system can ask questions about concepts more abstract than the parameter to be filled in, or about related product dimensions. Depending on the user's answer, a suitable default value will be applied. For example, the number of power supplies to be placed into a telecommunication switch could be derived by asking the user about the preferred level of reliability.

3. Distributed configuration

In order to allow a set of distributed configurators to co-operatively solve a configuration task, we employ a general

logical framework based on the component-port representation [8]. In practice, a configurable artifact is composed of a set of predefined component types that are characterized by a set of properties and can be interconnected using pre-enumerated connection points (ports) [2]. We assume that the configuration knowledge (available types as well as additional logical sentences restricting the allowed product configurations) are contained in the domain descriptions (*DD*) of the configurators. In addition, there are domain-independent sentences C_{Basic} describing e.g., that one port can only be connected to one other port.

Configuration problems are solved according to some specific user requirements (*SRS*) whereby these requirements are again expressed in terms of logical sentences. For describing an actual configuration result we restrict ourselves to a limited number of predicates contained in a set *CONL*. Examples are the *type(c,t)* predicate that associates a component *c* with one of the predefined types, or the *conn(c1,p1,c2,p2)* predicate that describes a connection between two component instances on port *p1* and *p2* respectively.

Definition (Configuration Problem): *A configuration problem is described by a triple (DD,SRS,CONL) where DD and SRS are sets of logical sentences, and CONL is a set of predicate symbols. DD represents the domain description, SRS the system requirements specification for a configuration problem instance. A configuration CONF is described by a set of positive ground literals whose predicate symbols are in CONL.* ❒

Definition (Consistent Configuration): *Given a configuration problem (DD,SRS,CONL), a configuration CONF is consistent if* $DD \cup SRS \cup CONF$ *is satisfiable.*❒

To ensure completeness of the configuration, additional completeness axioms have to be added and the notion of *valid* and *complete configuration* can be defined ([4]).

However we cannot assume that the problem can be solved in a centralized approach due to e.g., security reasons, i.e., none of the participating configurators has complete knowledge about the problem domain. Therefore we will define a distributed configuration problem and sketch how the distributed search takes place.

3.1 Defining Distributed Configuration

Analogously to the definition of the configuration problem, we define a *Distributed Configuration Problem*:

Definition (Distributed Configuration Problem): *A distributed configuration problem involving n different configurators is described by a triple* $(DD_{set}, SRS_{set}, CONL)$ *where*

$DD_{set} = \{DD_1, ..., DD_n\}$ *and*
$SRS_{set} = \{SRS_1, ..., SRS_n\}$.

Each element of DD_{set} *and of* SRS_{set} *is a set of logical sentences and CONL is a set of predicate symbols.* DD_k *corresponds to the domain description of configuration system k and* SRS_k *specifies its requirements. A configuration CONF is described by a set of positive ground literals whose predicate symbols are in CONL.* ❒

Definition (valid solution to a distributed configuration problem): *Given a distributed configuration problem* $(DD_{set}, SRS_{set}, CONL)$*, a configuration CONF is valid iff* $DD_k \cup SRS_k \cup \overline{CONF}$ *is satisfiable* $\forall k \in \{1, \ldots, n\}$. $\overline{CONF}$ *denotes a configuration extended with the completeness axioms.* ❒

In order to come to a complete solution for the distributed problem, configurators exchange partial configurations. The domain descriptions DD_k and the system requirements SRS_k are basically independent except the assertions regarding the configuration. We define a property called *defined interfacing*, where we ensure disjoint predicate symbols in DD_k and SRS_k but allow common predicate symbols in *CONL*. This helps us relating the centralized and the distributed configuration problems.

Theorem: *Let (DD,SRS,CONL) be a configuration problem and* $(DD_{set}, SRS_{set}, CONL)$ *be a distributed configuration problem with defined interfacing, where*

$DD = \bigcup_{dd \in DD_{set}} dd$ *and*
$SRS = \bigcup_{srs \in SRS_{set}} srs$.

CONF is a valid configuration for (DD,SRS,CONL) iff CONF is a valid solution for the distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$. ❒

For a sketch of the proof see [3]. It builds on the property of defined interfacing. As only predicates from *CONL* are used for describing a configuration result, it must be ensured that the configuration result *CONF* does not conflict with any of the involved configurators (resp. with their knowledge bases and system requirements i.e. $DD \cup SRS$).

Definition (Conflict): *Let (DD,SRS,CONL) be a configuration problem and CONF be a consistent set of sentences in CONL. CONF is a conflict for (DD,SRS,CONL) iff* $DD \cup SRS \mid\!- \neg CONF$. ❒

In the following we describe the rationale behind our approach towards distributed configuration. The configuration agents configure sequentially as well as concurrently and send their local solutions to a facilitating agent that acts as a coordinator. The latter unifies all received solutions and communicates them back to the involved agents. If all configuration agents determine the validity of the joint configuration then a common solution is found. As soon as a configuration agents detects a conflict with the joint configuration, the others are informed and measures for conflict resolution are taken. This resolution strategy has to ensure that a conflict never occurs twice during a session and that the non-existence of a valid configuration for the overall task is detected.

However, in a realistic supply chain setting we may assume the existence of a tree-structured ordering that defines the dependencies among the participating entities. Consequently, a partial sequentialization of the solution search for a distributed configuration task exists. One destined configuration agent will act as a main vendor that also fulfills the task of the facilitator for the supplying con-

figurators. The main vendor or service provider configures locally as far as possible and, once the configuration process reaches a phase where supplied parts need to be specified in more detail, a configuration solution is requested from the supplier. This way the solution space of the suppliers is restricted and the probability for conflict occurrence obviously diminished. The configuration agent of highest order starts the configuration process that is then resumed by the configurators of lower order. Agents of the same order may perform an asynchronous solution search. Further a partial ordering of configurators can also be used for an advanced negotiation strategy for conflict resolution, where agents with lower priority are the first ones to repair their configuration [15].

The distribution of configuration knowledge among the configuration agents is given for organizational and privacy reasons of business entities. However, this partitioning of the domain knowledge was not done randomly. Each configurator comprises the local knowledge necessary to customize products and services of the company behind. Configurable products and services being part of a distributedly configured overall solution may share resources and have defined interfaces and connection points vs. each other. However they basically represent independent objects that show some form of low coupling towards each other. Consequently, the local knowledge bases must not be seen as arbitrary logical theories, they are characterized by the sets of domain concepts in use. Based on this partitioning of configuration knowledge, we are able to identify the vocabulary of product domain concepts that specifies the configuration capabilities and informational needs of each agent on the domain of the overall problem. Each agent has informational interest on components and connections for which it has constraints defined in its local knowledge base. By employing domain ontologies we give an abstract description of the product domain of a configuration agent. Therefore each agent needs only a restricted view on the overall configuration.

4. CAWICOMS architecture

In this section we will give an overview of the architecture of the CAWICOMS framework for distributed configuration with adaptive user interfaces. Figure 1 depicts a typical deployment scenario for the CAWICOMS framework with one main vendor and a supplier of an add-on product.

4.1 Integration aspects

Product configuration systems (online selling systems) are typically not stand-alone applications but are integrated into an existing business software infrastructure (e.g., an Business-2-Business or ERP platform providing a product catalog or product ordering functionality).

Therefore we have to define interfaces between the CAWICOMS components and such a generic B2B system, in order to a) retrieve information about the configurable product (i.e., component descriptions or pricing) and b) hand over the results of the configuration process back to the underlying system in order to trigger e.g., the ordering process. Furthermore, supply-chain integration typically involves some long-term contracts and communication between the partners to automatically trigger an order at the

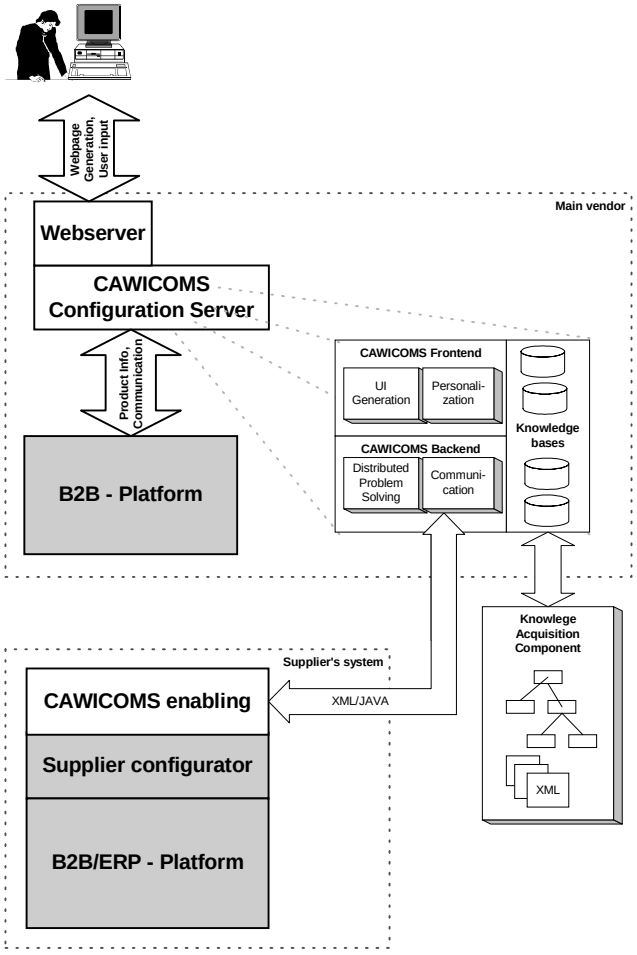

Figure 1 CAWICOMS Architecture

supplier system. For supply chain integration of standard, not-configurable products, several tools and technologies are available (that for instance involve an e.g., XML-based communication protocol). For our framework we assume, that such a platform exists between the participants of the supply chain.

On the other hand, there is a wide range of configuration technologies available [13], that differ both in knowledge representation and reasoning. The goal of the CAWICOMS project is to integrate different existing configurators. Therefore, we define a set of minimal functionalities that such an existing configurator has to supply, e.g., check and/or complete a partial configuration, as well as a simple communication protocol between configurators. For supporting cooperative problem solving we therefore have to use adapter components (*CAWICOMS enabling*) that translate the requests expressed using our general communication protocol to the proprietary representation of the local configurator.

4.2 Distributed Problem Solving

The main component of the *CAWICOMS Backend* provides functionality for the distributed problem-solving task. At the main vendor's site a configurator with facilitation capabilities is located that has an integrated view on the supplier's products. Note that this integrated view only contains relevant portions of the supplier's product structures and is constructed during the knowledge acquisition phase based on common ontological concepts (see subsection on knowledge acquisition). The CAWICOMS Backend processes user requests (e.g., selection of options or initiation of compatibility checks) that are entered by the user and handed over by the *CAWICOMS Frontend*. When a request for a configuration solution is entered, the facilitator initiates the distributed problem solving process and contacts the supplying configurators if needed.

From a technological point of view the following techniques are applied in our framework. We use and extend the state-of-the-art industrial configuration library ILOG Configurator ([7]) as the basis for the distributed problem-solving component. We extend it with a modular component that enables the communication of partial configurations and configuration requests to the suppliers. Furthermore a generic layer is developed that facilitates the exchange of complex data structures (that are typically used in the configuration domain, e.g., partial configurations). The exchange of information can be done both in terms of JAVA data structures as well as by means of XML documents. In addition, this layer also defines a protocol for data exchange (e.g., passing user choices or partial configurations) but also for method invocation, e.g., initiation of the solving process. This data-exchange layer and the protocol is used both for the communication between configurators along the supply chain as well as for communication to the Frontend with the effect, that the individual components of the CAWICOMS Configuration server (e.g., the configurator at the main vendor site or even the user interface generation component) can be exchanged by other tools. Finally, the possibility of exchanging XML documents with a defined protocol between cooperating configurators allows easy integration of existing configurators at the supplier's site.

4.3 Generation of personalized user interfaces

The *CAWICOMS Frontend* is responsible for the generation of personalized user interfaces. The two main activities carried on during the interaction with a customer are:

- the acquisition of the information necessary to the configuration process (by asking the user, or by predicting suitable values, on the basis of the user's knowledgeability and estimated preferences).
- the presentation of the (possibly partial) solutions produced by the back-end.

Both tasks can be personalized by exploiting specific strategies, given the information stored in the user model. A back-ground activity carried on by the Frontend during the interaction concerns the analysis of the user's actions, aimed at discovering precise information about her/him and revise the user model accordingly.

The Frontend exploits a set of knowledge bases, storing information about the properties characterizing the most typical classes of users (novices, experts, and so forth), user modeling acquisition strategies and personalization strategies. The Frontend also exploits a domain ontology, storing product information which is essential to the personalization task. This ontology describes information about products which can be used to describe them and ask information about the preferred values in a perspective easily understandable to the user (in contrast to the configuration-based perspective adopted in the back-end).

Based on the estimate of the user's properties the Frontend selects a personalization strategy, for instance "to present only top-level features and no technical details" and generates an adequate user interface using the predefined available graphical elements. Furthermore the Frontend is able to preset certain configurable options with default values, whereby these defaults correspond to the estimated interests of the current user.

The user inputs and the defaults suggested by the Front-end are handed to the Backend and the distributed problem solving mechanism is initiated. After the calculation of results – that are passed back using the generic data exchange mechanism – these results are also presented to the user in a personalized way, i.e., for instance the technical details are omitted.

4.4 Knowledge acquisition

One of the most important issues within the CAWICOMS framework is knowledge acquisition and representation. Because of the wide variety of knowledge representation and reasoning techniques that were developed during the long history of product configuration, up to now there is no agreed upon standard for designing product models or knowledge exchange. However, recent trends show that extensions relying on the component-port model for configuration from [8], as well as conceptual modeling techniques (e.g., based on the *Unified Modeling Language*) are useful in finding a unified view on the configuration problem ([1], [10], [12]). This view on the configuration problem can also be found in commercial configuration tools [7]. Our knowledge acquisition approach relies on the idea of using the *Unified Modeling Language* to model generic product models [1], because the notation is widely used in industrial environments and we are able to compile the knowledge bases mostly automatically from the comprehensive graphical depictions.

For the integration of the different product models of the individual suppliers we employ the approach described in [5] that relies on the exchange of *functional architectures*: Using this technique the suppliers *publish* relevant parameters of their product models in order to be integrated into the main view of the facilitator. Because of this partial publishing, there is no need for having a centralized point of knowledge because e.g., low-level technical details are never shown to

the others. During this (manual) integration phase the relevant knowledge on the supplier product has to be modeled with this modeling mechanism which shows to be possible even for systems that are e.g., rule-based because of the generality of the modeling technique.

Finally, the integrated product model can be automatically translated into a standardized XML-based representation that is enhanced with additional knowledge, i.e., personalization information (e.g., user-specific defaults) or distribution information (e.g., which supplying configurators can provide this component). The XML representation can then be parsed in order to parameterize the main configurator (i.e., load the product model) and the user interface generator.

5. Conclusions

The CAWICOMS project aims at two important aspects of future configuration and online selling technology for both the Business-2-Consumer (B2C) and the Business-2-Business (B2B) e-commerce segments. We described the demand for personalized user interfaces for selling complex technical products to users with different knowledge levels and experience (B2C) as well as the need for supply-chain integration of configurable products (B2B).

At the current status of the project we are currently implementing and integrating the individual modules for a prototype system. In order to support portability and scalability, the framework is developed in a Java-based, component-oriented environment. We are using the *Java2 Enterprise Edition (J2EE)* [11] framework that provides tools and techniques for developing distributed applications with automatic load balancing, persistency, naming services, and Web-page generation using *Java Server Pages*. Furthermore, the individual modules will be developed in a component-oriented manner (*Enterprise Java Beans*) in order to support portability across application servers as well as easy exchangeability of individual components.

Finally, component oriented technologies are showing to be adequate for distributed software development and we are looking forward to gather experiences on distributed project management and distributed software development, which will be an important issue in the present and future globalized software development industry.

6. References

[1] Ardissono L. and Goy A., Tailoring the Interaction With Users in Web stores. User Modeling and User-Adapted Interaction, 10(4), , Kluwer Academic Publishers, 2000, pp. 251-303.

[2] Felfernig, A., Friedrich, G., and Jannach, D., UML as domain specific language for the construction of knowledge-based configuration systems, in: International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 10(4), 2000, pp. 449-470.

[3] Felfernig, A., Friedrich, G., and Jannach, D., Distributed Configuring, Technical Report, University of Klagenfurt, 2001.

[4] Felfernig, A., Friedrich, G., Jannach, D., and Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases, In Proceedings 14th European Conference on Artificial Intelligence (ECAI'2000), Berlin, Germany, 2000.

[5] Felfernig A., Friedrich G., Jannach D., Zanker, M. Integrating Knowledge-Based Configuration Systems by Sharing Functional Architectures. Proc. 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, French Riviera, France, 2000.

[6] Jameson A., Knowing what others know: Studies in intuitive psychometrics. PhD thesis, University of Amsterdam, Netherlands, 1990.

[7] Mailharro, D. A Classification and Constraint-based Framework for Configuration, AI EDAM, Vol. 12 (1998), Cambridge University Press, 1998.

[8] Mittal, S., and Frayman, F.: Towards a generic model of configuration tasks, Proc. Intl. Joint Conference on Artificial Intelligence (IJCAI'89), Detroit, Morgan Kaufmann, 1989.

[9] Pearl, J. Probabilistic reasoning in intelligent systems: Networks of plausible inference. San Mateo, CA: Morgan Kaufmann, 1988.

[10] Peltonen, H., Männistö, T., Soininen, T., Tiihonen, J., Martio, A., and Sulonen, R.: Concepts for Modeling Configurable Products. In Proceedings of European Conference Product Data Technology Days 1998, Quality Marketing Services, Sandhurst, UK, 1998, pp. 189-196.

[11] Perrone, P., Chaganti, V., Building Java Enterprise Systems with J2EE, Sams Publishing, Indianapolis, 2000.

[12] Soininen, T., Tiihonen, J., Männistö, T., and Sulonen, R.: Towards a general ontology of configuration, AIEDAM, special issue: Configuration Design, Vol. 12(4), 1998, pp. 357-372.

[13] Stumptner, M.: An overview of knowledge-based configuration, AI Communications 10(2), 1997

[14] Winterfeldt, D. von and Edwards W., Decision analysis and behavioral research. Cambridge, England: Cambridge University Press, 1986.

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

Further Information: *http://www.cawicoms.org*