CHAPTER 7

Rapid Knowledge Base Development for Product Configuration Systems using the Unified Modeling Language

Alexander Felfernig

Gerhard Friedrich

Dietmar Jannach

Markus Zanker

Institut für Wirtschaftsinformatik und Anwendungssysteme,

Universität Klagenfurt

ABSTRACT

Knowledge-based product configuration systems play an important role in modern business strategies. These systems support the sales representative or the technical engineer to cope with the complexity of configurable products, the huge number of available variants, and different restrictions on allowed product constellations. Nowadays, typical knowledge-based configuration systems are not well integrated into standard software development processes but use proprietary knowledge representation formalisms which are not understandable for domain experts. In this chapter we show how the Unified Modeling Language (UML) can be applied as domain-oriented notation for the design of configuration knowledge bases. The provided modeling concepts can be used for comprehensible knowledge acquisition and are given precise semantics, such that an automatic translation to executable configuration knowledge bases is feasible. Following a discussion on these modeling concepts we show how their expressiveness can be enhanced by using the Object Constraint Language (OCL) and how the construction of complex configuration models can be supported.

7.1 INTRODUCTION

In today's rapidly changing, globalizing markets traditional mass production paradigms appear anachronistic. Mass production is increasingly replaced by customer-individual production of highly variant products. Companies are forced to diversify their product spectrum in order to be able to fulfill the individual needs of customers. The additional

costs for offering customer-individual products must be minimized in order to be able to provide customizable products in a competitive fashion. "Mass-customization"[3] appeared as a new paradigm representing the trend towards the production of highly variant products under mass production speed and pricing conditions. This paradigm imposes increasing demands on the development and maintenance of software supporting sales and production of highly variant products. This software must be able to handle rapidly changing, complex constraints on the products and on the corresponding processes supporting quotation, order processing, production, delivery, and maintenance. In particular knowledge-based configuration systems (configurators) are increasingly applied for supporting processes related to highly variant products.

There exists a variety of application domains for product configuration systems. One well known application domain is the telecommunication industry, where telecommunication systems support switching functionality for enterprise-wide telephone services. Furthermore, these systems also provide additional services like voice-over-ip, messaging services, ISDN connections, video-telephony, or video-conferencing. In this context configurators are used for calculating bill-of-materials representing the configuration of a switching system. The configuration of audiovisual equipment, automobiles, computer systems, trucks, airplanes, modular furniture (kitchens etc.), industrial products (e.g. valves, actuators, or controls), or light control systems are further application areas for knowledge-based configuration.

Effective application of configuration technology implies a number of improvements. The configuration system automatically checks the requirements imposed by the customer with regard to given marketing constraints, technical constraints, and constraints concerning the production process, which reduces the response time to given customer requests. Automated configuration avoids errors in the quotation and order processing phase. Consequently, time consuming reconfigurations of non-realizable orders are avoided and the time between equipment sales and delivery/installation of the product is decreased.

Configuration systems can significantly contribute to the improvement of processes related to highly variant products and are of strategic importance for enterprises. However, the development and maintenance of these systems is facing a set of challenges, which must be tackled in order to allow an effective application of configuration technology. First, the complexity of the task requires the sophisticated knowledge of experts. This knowledge must be effectively acquired and translated into a corresponding configuration knowledge base. Second, configurator development time is short and strictly limited since the development of the product and the product configuration system have to be done in parallel. Finally, the configuration knowledge base has to be adapted continuously because of changing sets of available components and configuration constraints.

In the following we show, how the knowledge acquisition task for configuration knowledge bases can be effectively supported by applying the Unified Modeling Language (UML[36]) as domain-specific representation language for building configuration knowledge bases. We employ UML, since this language is widely applied in industrial software development as a standard design language supporting the software development process starting with the requirement analysis phase proceeding until the implementation phase. UML has a built-in constraint language which supports a formal definition of constraints on models that are designed using the graphical concepts of UML class diagrams. UML is extensible for domain-specific purposes, i.e. the semantics of the basic modeling concepts of the language can be further refined in order to be able to provide domain-specific modeling concepts, which allow a more intuitive construction

of the corresponding models. Finally, we have made excellent experiences in using UML designs for validation by technical experts.

7.2 CONFIGURATION TASK

In order to avoid time-consuming development of specific designs for each customer, customizable products are designed following a building block principle where basic parts can be configured into different sorts of assemblies. A configuration task can be characterized through a set of available component types, a description of their properties, namely attributes and possible attribute values, connection points (also called ports), and constraints on legal combinations of those constitutive elements. Given some customer requirements, the result of computing a configuration is a set of components, corresponding attribute valuations, and connections satisfying all constraints and customer requirements.

In Felfernig et al.[20] an application-independent first-order logic-based definition of a configuration task is given. In this work a configuration problem is defined as a logic theory based on two sets of logical sentences.

1. Domain description (DD). Configurations are composed from components corresponding to component definitions in a component catalog – these definitions are described by a set of *types*. Attributes of component types are described by the function *attributes*, the domains of those attributes are described by the function *dom*. Finally, ports of component types are described by the function *ports*. The domain of these functions are sets of constants describing the corresponding attributes, attribute domains, and ports. These sets including a set of constraints on legal combinations of components, connections, and value instantiations are assumed to be contained in the set DD.

2. System Requirements Specification (SRS). Most configuration problems incorporate some kind of additional requirements (e.g. customer requirements) which describe additional constraints respectively initial components which must be part of the configuration. These constraints are contained in SRS.

In Felfernig et al.[20] a configuration problem is defined as follows.

Definition (Configuration Problem). *In general 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, and SRS specifies the particular system requirements which define an individual configuration problem instance. A configuration CONF is described by a set of positive ground literals whose predicate symbols are in the set of CONL.*

Note that - depending on the configuration domain – the set of predicates in CONL is extensible if needed. For example, a predicate could be defined specifying that a certain port of a component must be left unconnected. A consistent configuration is defined as follows in Felfernig et al.[20].

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

The intuitive definition of a consistent configuration allows determining the validity of partial configurations, but does not require completeness of the configuration (e.g. a partial configuration only containing a floppy disk is a consistent configuration). It

is necessary, that a configuration explicitly includes all needed components as well as their connections and attributes. Therefore we need to introduce an explicit formula for each predicate symbol in CONL (completeness axioms) to guarantee this completeness property. In order to stay within first-order logic, this property is modeled by first order formulae. A set of completeness axioms[20] assures the completeness of CONF. Such a complete configuration is denoted as CCONF.

Definition (Valid and Irreducible Configuration). *Let (DD, SRS, CONL) be a configuration problem. A configuration CONF is valid iff DD* $\cup$ *SRS* $\cup$ *CCONF is satisfiable. CONF is irreducible if there exists no other valid configuration* $CONF_{sub}$ *such that* $CONF_{sub} \subset CONF$.

As mentioned in Friedrich,Stumptner[23] the above definitions represent a high-level view of the language developed and used in the COCOS configuration project[24], which used as representation formalism a generative constraint satisfaction scheme that can be defined by a direct mapping from the consistency-based semantics sketched in this section.

7.3 BUILDING CONFIGURATION KNOWLEDGE BASES IN UML

The unavailability of methods for handling the increasing size and complexity of software systems triggered the “software crisis” in traditional software development. Similarly, knowledge-based systems development experienced increasing challenges triggered by the size and complexity of the knowledge bases. R1/XCON[5] can be seen as an representative example for the complexity of knowledge base maintenance. In 1989 the knowledge base had about 31.000 components and about 17.500 rules (engineering, manufacturing, and marketing rules) with a changing rate about 40% (rule additions, deletions, and modifications) per year.

In order to tackle these challenges, a number of methods and tools supporting the expert system development process were proposed (e.g. Schreiber et al.[42]). Using these approaches, the development of knowledge-based systems was supported and improved by the provision of diagrammatic notations and formal description languages. Similar approaches were also developed in Software Engineering research in order to effectively handle the implementation of software systems and to support a structured development process for these systems. These approaches have progressed from functional decomposition and data-driven techniques and methods (e.g. Cameron[9]) to object-oriented analysis and design techniques and methods such as Rumbaugh et al.[33], Booch et al.[7]. Moving from data-driven and functional approaches, research focused on the development of object-oriented analysis and design approaches, which are nowadays also accepted in industrial software development processes. These approaches reduce the “semantic gap” between the considered universe of discourse and the abstract model and increase the understandability and maintainability of the resulting models (Jacobson et al.[27]).

The Unified Modeling Language (UML[36]) is the result of an integration of the object-oriented approaches of Booch et al.[7], Jacobson et al.[27], and Rumbaugh et al.[33], which is well established in industrial software development processes. UML is applicable throughout the whole software development process from the requirements analysis phase to the implementation phase providing different notations. One of the major advantages of UML is the principle of extensibility of the basic meta-model. Following this principle the basic modeling concepts defined in the UML meta-model can be further extended in order to support the definition of domain-specific modeling

concepts for specific application areas. The set of domain-specific modeling concepts including constraints on their syntactical usage are defined in an UML *profile*[1].

State-of-the-art approaches for developing software systems provide methods and techniques which support an effective construction of those systems. However, the transformation of informal requirements into an implemented system is still a challenge for the development of knowledge-based systems as well as for traditional software development. In the following we show how to integrate the development of configuration systems into the standard software development process by using UML as a knowledge acquisition front-end which allows the automatic generation of executable configuration knowledge bases.

7.3.1 Example: UML Configuration Model

For the following discussions the simple UML configuration model shown in Figure will serve as working example.

This model represents the generic product structure, i.e. all possible variants of a configurable car. The basic structure of the product is modeled using classes, generalization, and aggregation. The set of possible products is restricted through a set of constraints which are related to technical restrictions, economic factors, and restrictions according to the production process. The used modeling concepts can be seen as an ontology in the sense of Chandrasekaran et al.[10], i.e. ontologies are theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge. This ontology represents an integration of different basic configuration paradigms (Soininen et al.[40]). Component types (represented through the stereotype "ComponentType") represent the constituent parts of a configuration model; *car*, *engine*, or *gearing* are examples for such component types (see Figure 7.1). Component types are organized in a part-of hierarchy, where aggregations between component types can be either composite or shared aggregations in the sense of the UML semantics[32].

Furthermore, resources and functions are further basic concepts which are represented as stereotyped classes in a UML configuration model[2]. Parts of a configuration problem can be seen as a resource balancing task, where some of the component types produce some resources and others are the consumers. In many cases some parts of the configuration model contain information which are relevant for the customer, e.g. not every screw of a car must be presented to the customer. Functions are used to represent exactly those parts of the configuration model. Similar to component types, functions can be organized using partonomies and taxonomies.

Beside the structural information, a configuration model also contains a set of constraints on the correct usage of the components and functions within a configuration. Such constraints can also be expressed graphically.

In Figure 7.1 the existence of an *automatic gearing* requires the existence of an *otto engine* within the final configuration. Furthermore, a *4-wheel gearing* must not be used in combination with a *diesel engine* within the final configuration. These constraints can be used to express basic restrictions on the configuration model, but must not be seen as sufficient for expressing all kinds of constraints on a configuration model. In order to

[1] For building product models in UML we have defined an UML configuration profile.

[2] These concepts are not used in the example model.

express constraints on configuration models which cannot be expressed graphically, we employ the Object Constraint Language (OCL)[3].

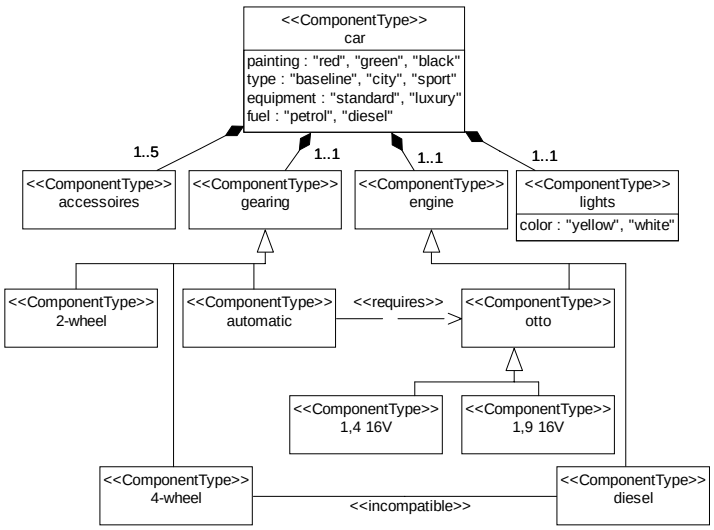

Figure 7.1 A simple UML Configuration Model

7.3.2 Introducing Domain-specific Modeling Concepts in UML

In order to introduce a modeling concept into the configuration profile, the following steps are necessary.

1. Define the new concept (a new stereotype or tagged value[4]) and state the well-formedness rules for its correct usage within the model. Well-formedness rules are expressed as OCL constraints on the UML metamodel. In order to restrict the usage of the stereotyped dependency "requires" (see the requires relation between the *automatic gearing* and the *otto engine* in Figure 7.1) to the connection of two component types, the following constraint can be formulated.

context dependency inv:
self.stereotype.name="requires" implies
self.client.stereotype.name="ComponentType" and
self.supplier.stereotype.name="ComponentType".

[3] The application of OCL will be discussed in Section 7.4.

[4] *Stereotypes* and *tagged values* are basic mechanisms provided by UML for defining restrictions on the UML meta-model.

2. Define the semantics of the concept for the configuration domain by stating the facts and constraints induced to the logic theory when using the concept. These translation rules are discussed in detail in Felfernig et al.[18),19)].

In order to state additional constraints on the product model which cannot be expressed graphically, the Object Constraint Language (OCL) can be employed. In Section 7.4 we sketch, how OCL can be used to define additional constraints on configuration models.

7.3.3 Derivation of Configuration Knowledge Bases

In order to support the automatic construction of a configuration knowledge base from the conceptual UML model, a clear definition of the semantics of the employed modeling concepts has to be provided (see Felfernig et al.[18),19)]). This is achieved by defining translation rules from the conceptual configuration model to a logical theory[20)]. Such a theory includes the description of a part library and constraints on those parts. The result of the translation is a set of first-order logical sentences that form a domain description which can be used by a configuration system which is based on the component-port representation (Mittal,Frayman[29)] and Friedrich,Stumptner[23)]).

In the following we do not discuss the translation rules in detail, but give a couple of examples for the result of applying the translation rules. A discussion of the rules for translating UML class diagrams and OCL constraints into configuration knowledge bases can be found in Felfernig et. al[18),19)].

The configuration knowledge base which corresponds to the UML configuration model of Figure 7.1 is the following.

types={car, accessoires, gearing, engine, lights, 2-wheel, automatic, 4-wheel, otto, 1-4-16V, 1-9-16V, diesel}.
attributes(car)={painting, type, equipment, fuel}.
attributes(lights)={color}.
dom(car, painting)={red, green, black}.
dom(car, type)={baseline, city, sport}.
dom(car, equipment)={standard, luxury}.
dom(car, fuel)={petrol, diesel}.
dom(lights, color)={yellow, white}.
ports(car)={accessoires1,...,accessoires5, gearing, engine, lights}.
ports(accessoires)={car}.
ports(gearing)={car}.
ports(engine)={car}.
ports(lights)={car}.

The component types of Figure 7.1 are translated into a set of *types* with corresponding *attributes* and attribute *domains (dom)*. Furthermore, the partof relationships are translated into *port* constants which are used to express connections between components in a configuration.

The constraints defined on the configuration model of Figure 7.1 are translated into the logic representation of a configuration problem as follows. It is assumed, that *CONL={type/2, conn/4, val/3},* where *type(ID, a)* indicates that *ID* is of type *a*, *conn(ID1, p1, ID2, p2)* indicates that component *ID1* is connected via port *p1* with *ID2*

and *ID2* is connected with *ID1* via port *p2*. Furthermore, *val(ID, a, c)* indicates that the attribute *a* of *ID* has the value *c*.

An "automatic" gearing requires the existence of an "otto" engine:
type(ID1, automatic) $\wedge$ *type(ID2, car)* $\wedge$ *conn(ID1, car, ID2, gearing)* $\Rightarrow$ $\exists$ *ID3 type(ID3, otto)* $\wedge$ *conn(ID3, car, ID2, engine).*

A "4-wheel" gearing is incompatible with a "diesel" engine:
type(ID1, 4-wheel) $\wedge$ *type(ID2, car)* $\wedge$ *type(ID3, diesel)* $\wedge$ *conn(ID1, car, ID2, gearing)* $\wedge$ *conn(ID3, car, ID2, engine)* $\Rightarrow$ *false.*

The form of logical sentences is restricted to a subset of range-restricted[5] first-order-logic with set-extension, the variables[6] are all-quantified if not explicitly mentioned. In order to assure decidability, the term-depth is restricted to a fixed number. For implementation purposes the logical sentences can be regarded as instantiation schemes[23)] for the translation into other representation formalisms, such as a Generative Constraint Satisfaction Problem (GCSP[38)]) representation.

A configuration result consists of a set of positive ground literals whose predicate symbols are in CONL. Assuming that *CONL={type/2, conn/4, val/3}*, the following set of ground literals could represent a configuration result CONF.

CONF={type(id1, car). type(id2, accessoires), type(id3, automatic). type(id4, 1_4_16V). type(id5, lights). val(id1, painting, red). val(id1, type, city). val(id1, equipment, standard). val(id1, fuel, petrol). val(id5, color, white). conn(id1, accessoires, id2, car). conn(id1, gearing, id3, car). conn(id1, engine, id4, car). conn(id1, lights, id5, car)}.

7.4 EXTENDING CONFIGURATION MODELS USING OCL

The Object Constraint Language (OCL[45)]) is an expression language based on first-order logic which enables the definition of constraints on object-oriented models. OCL was strongly influenced by the development of Syntropy[11)], an object-oriented modeling language which combined the diagrammatic representation concepts of OMT[33)] with the formality of Z[14)]. OCL complements the graphical (semiformal) notation of UML by providing a precise vocabulary for expressing constraints on graphical models. Since UML is a wide-spread modeling language in industry, OCL itself establishes an increasingly important role in the field of formal specification languages. However, the definition of the OCL semantics is based on a proposed syntax (represented as context-free grammar) and additional textual descriptions and examples. Although this is a quite intuitive approach for demonstration purposes, a corresponding formal definition is needed. In order to provide a semantics for OCL constraints in product configuration, we propose a set of translation rules that transform those constraints into the logic representation discussed in the previous sections[7].

A constraint imposed to the configuration model of Figure 7.1 could state that a *red car* requires *yellow lights*. This constraint can be formulated as follows in OCL.

---

[5] Every variable of the consequence part of the clause is also contained in the antecedent part.

[6] Variables are starting with an upper case letter.

[7] A discussion of the translation rules can be found in Felfernig et al.[19)].

context car inv:
car.painting="red" implies car.lights.color="yellow"

This constraint is built from the following parts:

- *Context:* A context describes for which class the constraint has to hold. In the above example the constraint has to hold for all *car* instances.
- *Navigation expression:* The navigation expression *car.lights* results in the lights component connected with the car instance. The name *"lights"* is the default name of the association from the class *car* to the class *lights* since no association name is specified here[32].
- *Attribute access:* access to attributes of objects. In the above example, the attributes *painting* and *color* are accessed.
- *Operator "=":* In the above example this operator is used for comparing the attribute values of *painting* and *color* with corresponding constants ("red", "yellow").

Such an OCL constraint can be translated into a closed logical formula following an implication schema, where the right hand side (RHS) contains the all-quantified variables which correspond to the result of the evaluations of navigation expressions and the corresponding operations. The left hand side (LHS) contains the corresponding logical consequence, i.e. the evaluation results of the right hand side connected with the logical and relational operators.

The example OCL constraint on the car configuration model is translated as follows into the logic representation of a configuration problem.

$$type(ID1, car) \wedge ResultSet1=\{ID1\} \wedge connected\_set\_a(ResultSet1, painting, ResultSet2) \wedge connected\_set(ResultSet1, lights, ResultSet3) \wedge connected\_set\_a(ResultSet3, color, ResultSet4) \wedge Val1 \in ResultSet2 \wedge Val2 \in ResultSet4 \Rightarrow (Val1=red \Rightarrow Val2=yellow).$$

The predicates *connected_set* and *connected_set_a* are used to express a component navigation respectively an attribute access in a given OCL navigation expression, e.g. *connected_set(ResultSet1, lights, ResultSet3)* expresses the navigation from a *car* instance to the connected *light* instance, *connected_set_a(ResultSet1, painting, ResultSet2)* expresses an access to the attribute *painting* of the class *car*.

7.5 STRUCTURING CONFIGURATION KNOWLEDGE BASES

The application of UML for building configuration models is motivated by the wide-spread use of the language, the high degree of understandability, the extendibility for domain-specific purposes, and the availability of a built-in constraint language which allows the definition of complex constraints. However, when modeling highly variant products, that offer the customer a vast amount of interdependent options, expressiveness restrictions of a single (partitioned) diagram are approached. The diagrammatic depiction becomes harder to maintain and understand as the number of elements and constraints depicted in the diagram increases. In order to improve the understandability and maintainability of UML configuration models, we introduce the notion of contextual diagrams (see Felfernig et al.[17),21)]), which provide a means for structuring constraints in a configuration model.

7.5.1 UML Packages

Before introducing contextual diagrams, let us have a look at the basic structuring mechanisms provided in UML. A package is simply defined in UML as a grouping of model elements. Packages themselves may be nested within other packages. It is therefore straightforward to partition a diagram into packages under the premise of high cohesion among the elements in the same package and low coupling between different packages.

The strictly defined hierarchical package structure can not only be used to partition the model knowledge, but as well to structure the solving process of the underlying configuration system. In some cases parts of a configuration problem can be solved more or less independently. This property can be used to structure the solving process, which leads to smaller problem sizes and less computational complexity in the solving process. The partitioning of configuration problems for allowing efficient calculations of solutions for large configuration problems is discussed in Fleischanderl,Haselböck[25]. The configuration of switching systems is one of the most complex application areas for knowledge-based configuration. As mentioned in Fleischanderl et al.[24], configurations representing switching systems can become large and complex. Fleischanderl et al.[24] describe an example system with about 43.000 components, 215.000 attributes and 120.000 ports. In order to allow an efficient calculation of solutions, the problem domain must be separated into different independent sub-problems – exactly this property was exploited in Fleischanderl,Haselböck[25].

7.5.2 Additional Structuring Mechanisms

Parts of the configuration knowledge sometimes apply only in a specific context. For example, depending on the country, different keyboards must be added to a personal computer configuration, furthermore the language of the operating system must be adequately configured. Consequently, the country is a parameter of the configuration model, which determines a set of additionally relevant constraints. If the customer wants to have a certain type of car (e.g. a car of type *baseline*), this selection can also impose further constraints which only apply in the case that this specific type is selected. Figure 7.2 shows the result of adding a set of additional constraints to the configuration model shown in Figure 7.1.

Such constraints can concern different classes sometimes stored in different packages of the configuration model. In real world settings, where the number of constraints and component types is large, the maintenance of models following the strategy shown in Figure 7.2 obviously leads to maintenance problems and is a challenge for the knowledge engineer as well as for the technical expert. For example, if constraints concerning the type of a car must be changed, these constraints are spread all over the configuration model assigned to classes stored in different packages. A closer look at the constraints shows that the constraints can be categorized into groups having the same preconditions. Exactly this property can be used for partitioning the whole model into different views, where each view contains only a restricted set of constraints. We denote such kind of views as contextual diagrams. Constraints concerning a specific context are organized in a central location (contextual diagram), which significantly alleviates the handling of changes. In the example given, constraints concerning the car type *baseline* can be organized in one single contextual diagram.

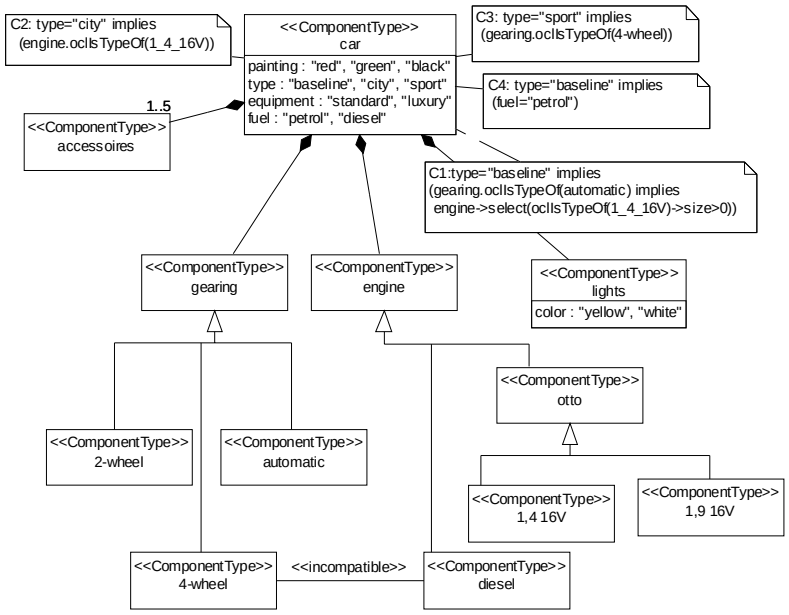

Figure 7.2 Configuration Model with additional constraints

7.5.3 Contextual Diagrams

The notion of context has been discussed in different research areas[15), 28), 39), 13), 43), 4)] using quite different interpretations. Overviews of different interpretations of the notion of context can be found in Charlton,Wallace[13)], Akman,Surav[4)]. Informally, a context can be interpreted as a general condition under which an event, action, etc. takes place. The interpretation of the notion of context used in the following is similar to the notion defined in McCarthy[28)] who proposes a formalism *ist(c,p)* in order to define a context *c*, in which the proposition *p* must hold, i.e. *p is true* in *c*. On the graphical level contexts are represented as contextual diagrams. Using contextual diagrams, the graphical representation (GREP) of a configuration knowledge base is organized in a context hierarchy (see Figure 7.3). An important relation defined on contexts is "$\prec$" (see McCarthy[28)]). Such an relation $c_1 \prec c_2$ defines a partial ordering over contexts meaning that the context c_2 contains all the information of the context c_1 and probably more. Except $GREP_{root}$, each contextual diagram $GREP_i$ has an assigned precondition $GREPP_i$, which represents the conditions, under which the constraints defined in $GREP_i$ apply. This precondition can be interpreted as a set of circumstances surrounding the actual configuration process, in which an additional set of constraints must hold. The root model $GREP_{root}$ contains basic structural elements of the configuration model including basic constraints on this model. Contextual diagrams $GREP_i$ can be constructed by imposing further constraints on a copy of an already existing contextual diagram. The result of this construction process is a hierarchy of contextual diagrams (see Figure 7.3), in which each contextual diagram $GREP_i$ either represents the root of the hierarchy ($GREP_{root}$), or is derived from another contextual diagram. Note that all the preconditions

of contextual diagrams starting from $GREP_{root}$ along the path to $GREP_i$ are included in $GREPP_i$. For example, the precondition $GREPP_3$ of contextual diagram $GREP_3$ in Figure 7.3 is *c* including the precondition of $GREP_1$, namely *a*, i.e. $a \wedge c$.

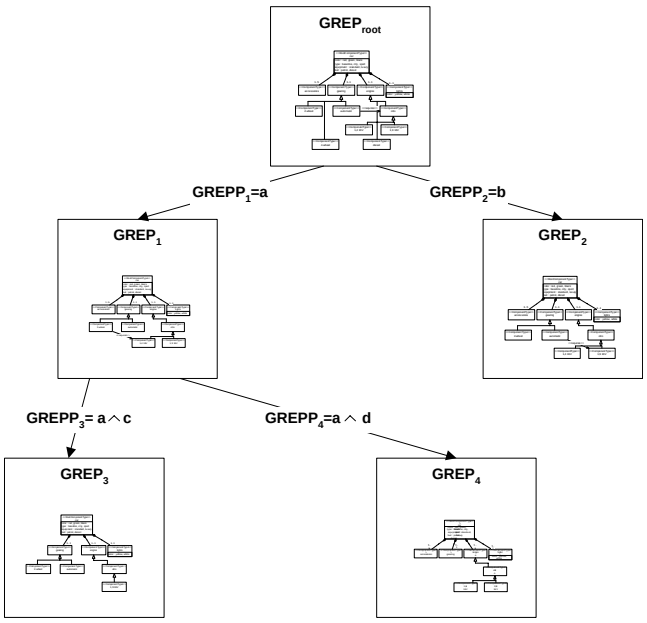

Figure 7.3 A simple Context Hierarchy

We define a context as follows:

Definition (Context). A context c_i is represented by a set of logical sentences $DD_i \subset DD$, where DD represents the domain description. DD_i contains those logical sentences, which solely apply in c_i and are derived from a corresponding contextual diagram $GREP_i$.

Figure 7.4 contains the constraints concerning a car of type *baseline*, which are now included in a separate contextual diagram. $GREPP_{baseline}$ in this diagram is simply the expression *type="baseline"*, which can be formulated as OCL constraint. Note that the knowledge engineer or the domain expert are not forced to enter this precondition textually as OCL constraint, but rather define those restrictions directly in the diagram itself, e.g. by reducing the domain of the corresponding attributes[8]. Constraints, which apply in the actual context can also be entered in a similar way.

[8] More complex conditions must be formulated as OCL expressions.

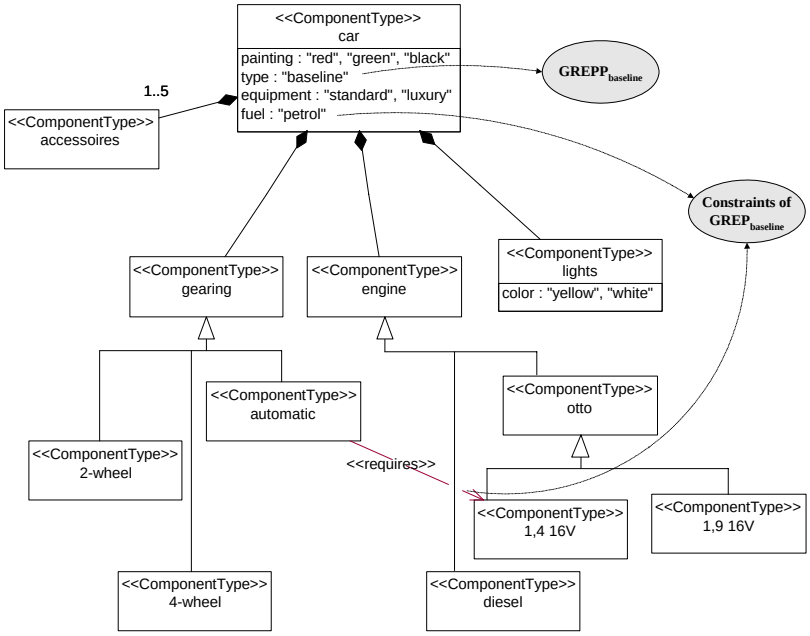

Figure 7.4 Contextual Diagram $GREP_{baseline}$

In order to define a precondition for the contextual diagram $GREP_{baseline}$, the designer of the knowledge base simply reduces the domain of the attribute *type* to *"baseline"* (see Figure 7.4). The constraint *fuel="petrol"* can be entered in a similar way. Finally, a graphical constraint *automatic gearing requires 1,4 16V engine* is defined using a *requires* dependency.

The example contextual diagram of Figure 7.4 is translated as follows into the logic representation of a configuration problem, where DD_{root} corresponds to the derived domain description of Section 7.3.3.

An automatic gearing requires a 1_4_16V engine:

type(ID1, car) ∧ ResultSet1={ID1} ∧ connected_set_a(ResultSet1, type, ResultSet2) ∧ Val1 ∈ ResultSet2 ∧ Val1 = baseline ∧ type(ID2, automatic) ∧ conn(ID1, gearing, ID2, car) ⇒ ∃ID3 type(ID3, 1_4_16V) ∧ conn(ID3, car, ID1, engine).

The fuel must be "petrol":
type(ID1, car) ∧ ResultSet1={ID1} ∧ connected_set_a(ResultSet1, type, ResultSet2) ∧ Val1∈ ResultSet2 ∧ Val1=baseline ∧ connected_set_a(ResultSet1, fuel, ResultSet3) ∧ Val2 ∈ ResultSet3 ⇒ (Val2=petrol).

7.6 RELATED WORK

The goal of Aldanondo et al.[1] is to propose modeling concepts enabling a non-computer specialist to describe generic configuration models. Based on a set of requirements concerning different classes of configuration problems a set of modeling concepts is presented which is based on the Dynamic CSP (see Mittal,Falkenhainer[30]) approach. This approach seems to have its advantages when modeling configuration problems for Dynamic CSP solving, but also seems to be restricted to the representation of different kinds of variables with the corresponding constraints. Compared to the approach presented in this chapter, the problem of representing configuration knowledge on an abstract level is solved by providing a graphical representation of problem variables rather than using representations which are decoupled from a specific problem representation. An overview of different representation formalisms used in knowledge-based configuration is given in Stumptner[41].

In Tiihonen et al.[44] a set of modeling concepts for building configuration models on a conceptual level is presented which is similar to the modeling concepts presented in Soininen et al.[40]. In addition to these modeling concepts a set of modeling guidelines is proposed in order to support a correct application of the concepts. Compared to our approach, no concepts for representing constraints are defined, furthermore no clear semantics for the modeling concepts are given.

In the recent years several fields in AI focused research on the improvement of inter-operability of knowledge-based systems by developing ontologies. In Chandrasekaran et al.[10] an ontology is defined as a theory about the sorts of objects, properties of objects, and relationships between objects that are possible in a specified domain of knowledge. An earlier definition was given by Neches et al.[31] who define an ontology as follows: “An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary”. Also a well known definition of the term ontology is given in Gruber[26] who defines an ontology as an “explicit specification of a conceptualization”. Cranefield,Purvis[12] propose the application of UML as ontology construction language as an alternative to the AI approaches such as KIF (Neches et al.[31]). The usage of UML is motivated by the graphical representation of the modeling concepts which is more intuitive than a linear textual representation. Furthermore the existence of a large group of users which is already familiar with the UML is seen as a strong argument for the application of the language. Compared to the approach presented in this chapter there is no support for automatic translation of ontologies into an executable representation. In Robbins et al.[37] an approach is presented for applying UML for modeling software architectures. The built-in extension mechanisms of UML are applied in order to define an ADL[9]-specific meta-model inside UML, i.e. a UML profile for exchanging architecture descriptions. In this context OCL is used to formulate constraints (well-formedness rules) on the meta-model.

In Bourdau,Cheng[8] a formal semantics for OMT[33] object model diagrams is given. A formal description of the system architecture containing the allowed states of the system is generated, which can be used to support verification and validation tasks. The main focus of this work is the generation of a formal specification in order to support the assessment of requirement specifications, whereas our approach concentrates on the definition of translation rules which support the automatic generation of a knowledge base which can be directly incorporated into a configuration environment.

---

[9] Architecture Description Language.

The requirement for concepts supporting a precise definition of constraints on object-oriented models motivated the development of the Object Constraint Language (OCL[45]). However, the semantics of the language is semiformal, i.e. it is defined in terms a context free grammar and a set of additional textual descriptions[35]. In order to support e.g. testing a specification (represented in OCL) against an actual implementation, a precise semantics of the specification language is a prerequisite. In Richters,Gogolla[34] a precise definition of the OCL is presented which similar to an algebraic specification. Baar et al.[6] discuss experiences in the application of OCL in industrial software development processes. In principle, OCL seems to be quite useful and software engineers and even domain experts with a technical background are able to apply OCL for stating formal constraints on a given object-model. Especially software engineers accepted OCL because of the similarities of its syntax to object-oriented programming languages. However, Baar et al.[6] point out that additional, more intuitive concepts are needed in order to support an effective introduction of OCL constraints. They made the observation that software engineers tried to change an objects state, what is prohibited by the declarative semantics of OCL. In order to tackle this challenge, Baar et al.[6] introduce the notion of constraint schemes. These schemes represent parameterizable constraints, which can be differently instantiated depending on the actual situation. For example, a constraint schema could restrict the occurrence of objects of a class to an upper bound. In this case the upper bound is represented by a variable which must be instantiated in order to instantiate the corresponding OCL constraint.

Mechanisms for structuring knowledge bases in the configuration domain are discussed in Fleischanderl,Haselböck[25], where concepts similar to packages are applied in order to structure large configuration knowledge bases. Mechanisms for structuring configuration knowledge bases are also discussed in Feldkamp et al.[16], where the SyDeR approach is presented. For resolving the problem of defining complex constraints over the product structure, the concept of an interface is introduced. Using interfaces, complex constraints are not assigned to components, but are part of the interface which describes a connection between several components. Compared to this approach of using explicit interface definitions in order to organize complex constraints, contextual diagrams reorganize constraints themselves by using preconditions as structuring criteria.

7.7 CONCLUSIONS AND FUTURE WORK

In this chapter we have presented the basic concepts for improving the configuration knowledge base development process by applying and extending the basic modeling concepts provided by the Unified Modeling Language (UML). The approach enhances the application of Software Engineering techniques to knowledge-based systems by providing a knowledge-acquisition front-end for knowledge-based configuration systems. Extensible standard design languages like UML are able to provide a basis for introducing and applying rigorous formal descriptions of application domains. These languages are comprehensible and are widely adopted in established industrial software development processes. The automated generation of specialized software applications allows rapid generation of prototypes and furthermore improves the requirements engineering phase through short feedback cycles. Software development departments are enabled to incorporate formal description languages into their standard development process. The application of systems based on such languages is no more restricted to specialists with corresponding knowledge in the area of formal description languages. The design model is comprehensible for domain experts as well and can be adapted and validated without the need of specialists. By applying domain-specific modeling

concepts domain experts are enabled to acquire configuration knowledge without having knowledge in modeling configuration knowledge bases using proprietary representations of configurator languages. This significantly contributes to a reduction of the knowledge acquisition bottleneck. Since the configuration knowledge base is represented on a conceptual level, configuration knowledge bases maintenance can be realized as well on a conceptual level, which leads to an increased degree of maintainability.

The application of the graphical modeling concepts of UML has its limits when building configuration knowledge bases[10], since in most domains there exist constraints which can not be represented graphically. The Object Constraint Language (OCL) is a language integrated in UML which was exactly developed for this purpose. In this work we have shown how to apply OCL for the formulation of complex constraints on UML configuration models.

The increasing size and complexity of configuration knowledge bases requires the provision of adequate modeling and structuring mechanisms. When modeling highly variant products which offer the customer a vast amount of interdependent options and choices, expressiveness restrictions are approached when using one single diagram. When acquiring configuration knowledge one has to cope with an intermingled and interdependent structural and functional product architecture. Exactly this challenge makes additional graphical representation concepts necessary. Motivated by the high complexity of large configuration knowledge bases we proposed a structuring mechanism called contextual diagrams.

There are a couple of further issues in the context of developing knowledge-based configuration systems, which we consider as a subject of future work. Knowledge acquisition using a conceptual modeling language (such as the UML) has shown to increase the applicability of knowledge-based configuration systems. However, the question has to be answered how to extend this approach in order to integrate additional and alternative concepts, such as natural language interfaces, which would allow the domain expert to explain the product structure and corresponding constraints in a more intuitive way without having knowledge in applying conceptual modeling techniques. The development of configurator user interfaces is strongly correlated to the development of configurator knowledge bases, since changes in the knowledge base trigger change requests for the corresponding user interface. Consequently, the support of the automatic construction of the configurator user interface implies corresponding reductions in development efforts. As has been shown in this chapter, modeling techniques from the area of Software Engineering have great potential in increasing the applicability of formal methods. Similar to interface generators in 4GL development environments, configurator user interfaces could be generated from the conceptual representation of the configuration knowledge base, i.e. an UML model of a configurable product can be further used to automatically generate the corresponding configurator user interface. The provision of software supporting the automatic generation of configurator user interfaces must be seen as a first step towards the realization of a sophisticated presentation layer for complex products and services. Different types of users (inexperienced, expert, etc.), different interaction styles (configuration from scratch, case-based configuration), and different preferences concerning products (customer is interested in multimedia personal computers, customer is primarily interested in server systems) impose the requirement for developing concepts for adaptive, i.e. personalisable user interfaces for configuration systems. The question has to be answered, whether the presented approach to knowledge acquisition can be enhanced in order to consider personalization concepts as well. The vision in this field

[10] In other domains as well.

could be to build a "virtual salesman" (some kind of configuration agent), which actively supports a personalized configuration process, i.e. determines the way the product is presented to the customer and the way the configuration system interacts with the customer.

One major task in the context of distributed configuration is the identification of concepts for integrating different configuration systems based on different knowledge representation formalisms. The component-port representation used in our work for formalizing configuration models built in UML has shown to be a widely accepted representation formalism for configuration problems. However, a general architecture for integrating different configurators has to be identified, which supports the integration of heterogenous configuration systems as well. Agent-based approaches for integrating knowledge-based systems seem to be a promising approach for the configuration domain. Agent communication languages (such as KQML[22]) support the communication between heterogenous knowledge-based systems, furthermore the corresponding knowledge representation languages enable the construction of ontologies, which support knowledge interchange between the engaged systems. The challenge here is to define ontologies suitable for integrating a large set of configuration systems.

The Semantic Web (Berners-Lee[2]) is the vision of developing enabling technologies for the Web which support access to its resources not only to humans but as well to applications often denoted as agent-based systems providing services such as information brokering, information filtering, intelligent search or synthesis of services. The technology provided by the Semantic Web community seems to be applicable for the configuration domain as well – especially for describing the capability of configuring certain product types as a kind of Web service and as a basis for defining common ontologies used as communication basis in distributed configuration.

BIBLIOGRAPHY

1) Aldanondo, M., Moynard, G. and Hamou, K.H.: General configurator requirements and modeling elements, In *Workshop on Configuration*, edited by Stumptner, M., (Berlin), pp. 1-6 (2000).
2) Berners-Lee, T.: *Weaving the Web*, (Orion Business Books) (1999).
3) Anderson, D.M.: Agile Product Development for Mass Customization, McGraw-Hill (1997).
4) Akman, V., and Surav, M., Steps Towards Formalizing Context: In *AI Magazine*, Vol. 17, pp. 55-72 (1996).
5) Barker, V.E., O'Connor, D.E., Bachant, J.D. and Soloway, E.: Expert systems for configuration at Digital: XCON and beyond, In *Communications of the ACM*, Vol. 32, No. 3, pp. 298-318 (1989).
6) Baar, T., Hähnle, R., Sattler, T. and Schmitt, T.H.: Entwurfsmustergesteuerte Erzeugung von OCL Constraints, In *Informatik* 2000, edited by Mehrhorn, K. and Snelting, G., Vol. 30, Jahrestagung der Gesellschaft für Informatik, (Springer), pp. 389-404 (2000).
7) Booch, G.: Object-Oriented Analysis and Design with Applications, In *Addison Wesley Object Technology Series* (1994).
8) Bourdeau, R.H. and Cheng, B.: A Formal Semantics of Object Models, In *IEEE Transactions on Software Engineering*, Vol. 21, No. 10, pp. 799-821(1995).
9) Cameron, J.:JSP&JSD: *The Jackson Approach to Software Methodology, 2nd edition*, IEEE Computer Society (1989).

10) Chandrasekaran, B., Josephson, J. and Benjamins, R.: What Are Ontologies, and Why do we Need Them? In *IEEE Intelligent Systems*, Vol. 14, No. 1, pp. 20-26 (1999).
11) Cook, S. and Daniels, J.:Designing Object Systems – Object Oriented Modeling with Syntropy, (Prentice Hall) (1994).
12) Cranefield, S. and M. S. Purvis: UML as an Ontology Modelling Language, In *Proceedings of the Workshop on Intelligent Information Integration*, (16th International Conference on Artificial Intelligence, Stockholm) (1999).
13) Charlton, C. and Wallace, K:Reminding and context in design, In *Proceedings 6th International Conference on Artificial Intelligence in Design* (AID'oo), (Boston: Kluwer Academic Publishers), pp. 569-588 (2000).
14) Diller, A.: Z – An Introduction to Formal Methods, John Wiley & Sons (1994).
15) Delisle, N.M. and Schwartz, M.D.: Contexts – a partitioning concept for hypertext, In *ACM Transactions on Information Systems*, Vol. 5, No. 2, pp. 168-186 (1987).
16) Feldkamp, F., Heinrich, M. And Gramann, M.: SyDeR System Development For Reusability, In *AIEDAM* Vol. 12, No. 4, pp. 373-382 (1998).
17) Felfernig A. and Zanker, M.: Diagrammatic Acquisition of Functional Knowledge for Product Configuration Systems with the Unified Modeling Language, In *Proceedings to International Conference on Theory and Application of Diagrams* (Diagrams'2000), Springer Lecture Notes in Artificial Intelligence, Vol. 1889, (Edinburgh), pp. 361-375 (2000).
18) Felfernig, A., Friedrich, G. and Jannach, D.: UML as domain-specific language for the construction of knowledge-based configuration systems. International Journal of Software Engineering and Knowledge Engineering (IJSEKE), 10,4: pages 449-469 (2000).
19) A. Felfernig, G. Friedrich, and D. Jannach: Generating product configuration knowledge bases from precise domain extended UML models, In *Proceedings 12th International Conference on Software Engineering and Knowledge Engineering*, (Chicago), pp. 284-293 (2000).
20) Felfernig, A., Friedrich, G., Jannach, D. and Stumptner, D.: Consistency-Based diagnosis of Configuration Knowledge Bases, In *Proceedings of the 14th European Conference on Artificial Intelligence* (ECAI'2000), (Berlin), pp. 146-150 (2000).
21) Felfernig, A., Jannach, D. and Zanker, M.: Contextual Diagrams as Structuring Mechanisms for Designing Configuration Knowledge Bases in UML. In *Proceedings of 3rd International Conference on the Unified Modeling Language* (UML'2000), Springer Lecture Notes in Computer Science, Vol. 1939, (York), pp. 240-254 (2000).
22) Finin, T., Labrou, Y. and Mayfield, J.: KQML as an agent communication language, In *Software Agents*, edited by Bradshaw, J., (Cambridge: MIT Press) (1997).
23) Friedrich, G. And Stumptner, M.: Consistency-Based Configuration, In *AAAI Workshop on Configuration, Technical Report* WS-99-05, (Orlando), pp. 35-40 (1999).
24) Fleischanderl, G. And Friedrich, G.: A. Haselböck, H. Schreiner, and M. Stumptner, Configuring Large Systems Using Generative Constraint Satisfaction, In *IEEE Intelligent Systems, Special Issue on Configuration*, Vol. 13, No. 4, edited by Freuder, E. and Falting, B., (IEEE) pp. 59-68 (1998).
25) Fleischanderl, G. and Haselböck, A.: Thoughts on Partitioning Large-Scale Configuration Problems, In *AAAI Fall Symposium on Configuration*, edited by Faltings, B. and Freuder, E., pp. 45-54 (1996).
26) Gruber. T.: A translation approach to portable ontology specifications, *Knowledge Acquisition*, pp. 199-220 (1993).

27) Jacobson, I., Christerson, M. and Övergaard, G.: Object-oriented Software Engineering – A Use-Case Driven Approach, (Addison Wesley) (1992).
28) McCarthy, J., 1993: Notes on formalizing context. In *Proceedings of the 13th IJCAI*, (Chambery, France) pp. 555-560 (1993).
29) Mittal, S. and Frayman, F.: Towards a Generic Model of Configuration Tasks,. In *Proceedings of the 11th IJCAI*, (Detroit) pp. 1395-1401 (1989).
30) Mittal, S. and Falkenhainer, B.: Dynamic Constraint Satisfaction Problems, In *Proceedings AAAI 1990*, (Boston), pp. 25-32 (1990).
31) Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, Senator, T. and Swartout, W.: Enabling technology for knowledge sharing, In *AI Magazine*, Vol. 12 No. 3, pp. 36-56 (1991).
32) Unified Modeling Language Specification Ver. 1.3 June (1999).
33) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: Object-Oriented Modeling and Design, In *Prentice Hall International Editions*, (New Jersey) (1991).
34) Richters, M. and Gogolla, M.: On Formalizing the UML Object Constraint Language OCL, In *Proceedings 17th International Conference on Conceptual Modeling* (ER'98) (1998).
35) Rational Software, Microsoft, Hewlett Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, Intellicorp, I-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies, and Softeam, OCL Specification ver. 1.1. (1997).
36) Rumbaugh, J., Jacobson, I. and Booch, G.: The Unified Modeling Language Reference Manual, (Addison Wesley) (1998).
37) Robbins, J.E., Medvidovic N., Redmiles, D.F. and Rosenblum D.S.: Integrating Architecture Description Languages with a Standard Design Method, In *Proceedings 20th ICSE*, (Kyoto) pp. 209-218 (1998).
38) Stumptner, M. and Haselböck. A.: A generative constraint formalism for configuration problems, In *Advances in Artificial Intelligence: Proceedings of the 3rd Congress of the Italian Association for Artificial Intelligence* AI*IA'93, edited by Torasso, P., (Springer, Berlin, Heidelberg) pp. 302-313 (1993).
39) Siegel, M., Sciore, E. and Salveter, S.: A Method for automatic rule derivation to support semantic query optimization, In *ACM Transactions on Database Systems*, Vol. 17, pp. 563-600 (1992).
40) Soininen, T., Tiihonen, J., Männistö, T. and Sulonen, R.: Towards a General Ontology of Configuration, in AIEDAM, Vol. 12, No. 4, pp. 357-372 (1998).
41) Stumptner. M.: An overview of knowledge-based configuration, In *AI Communications*, Vol. 10, No. 2 (1997).
42) Schreiber, A.T., Wielinga, B.J., DeHoog, R., Akkermans, H. and van de Velde, W.: CommonKADS: A Comprehensive Methodology for KBS Development, In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 6, pp.28-37 (1994).
43) Theodorakis, M., Analyti, A., Constantopoulos, P. and Spyratos, N.: Context in information bases, In *Proceedings of 3rd International Conference on Cooperative Information Systems* (CoopIS'98), (New York City: IEEE Computer Society), pp. 260-270 (1998).
44) Tiihonen, J., Lehtonen, T., Soininen, T., Pulkinen, A., Sulonen, R. and Riitahuhta, A.: Modeling Configurable Product Families, In *Proceedings of the 12th International Conference on Engineering Design* (ICED'99), edited by Lindemann, U., Birkhofer, H., Meerkamm, H. and Vajna, S., (München), pp. 1139-1142 (1999).
45) Warmer, J. and Kleppe, A.: The Object Constraint Language – Precise Modeling with UML, In *Addison Wesley Object Technology Series* (1999).