UML as knowledge acquisition frontend for Semantic Web configuration knowledge bases

*Alexander Felfernig, *Gerhard Friedrich, *Dietmar Jannach,
+Markus Stumptner, and *Markus Zanker
*Institut für Wirtschaftsinformatik und Anwendungssysteme,
Produktionsinformatik, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria,
email: {*felfernig,friedrich,jannach,zanker*}*@ifit.uni-klu.ac.at*.
+University of South Australia, Advanced Computing Research Centre,
5095 Mawson Lakes (Adelaide), SA, Australia,
email: *mst@cs.unisa.edu.au*.

Abstract

The trend towards highly specialized solution providers cooperatively offering configurable products and services to their customers requires the extension of current (standalone) configuration technology with capabilities of knowledge sharing and distributed configuration problem solving. On the one hand, a standardized representation language is needed in order to tackle the challenges imposed by heterogeneous representation formalisms of state-of-the-art configuration environments (e.g. description logic or predicate logic based configurators), on the other hand it is important to integrate the development and maintenance of configuration systems into industrial software development processes. We show how to support both goals by demonstrating the applicability of the Unified Modeling Language (UML) for configuration knowledge acquisition and by providing a set of rules for transforming UML models into configuration knowledge bases specified by languages such as OIL or DAML+OIL which represent the foundation for the description of configuration Web services.

1 Introduction

There is an increasing demand for applications providing solutions for configuration tasks in various domains (e.g. telecommunications industry, automotive industry, or financial services) resulting in a set of corresponding configurator implementations (e.g. [3,12,17,28]). Informally, configuration can be seen as a special kind of design activity [22], where the configured product is built of a predefined set of component types and attributes, which can be composed conform to a set of corresponding constraints.

Triggered by the trend towards highly specialized solution providers cooperatively offering configurable products and services, joint configuration by a set of business partners is becoming a key application of knowledge-based configuration systems. The configuration of virtual private networks (VPNs) or the configuration of enterprise network solutions are application examples for distributed configuration processes. In the EC-funded research project CAWICOMS[1] the paradigm of Web services is adopted to

[1] CAWICOMS is the acronym for Customer-Adaptive Web Interface for the Configuration of products and services with Multiple Suppliers (EU-funded project IST-1999-10688).

accomplish this form of business application integration. In order to realize a dynamic matchmaking between service requesters and service providers, configuration services are represented as Web services describing the capabilities of potentially cooperating configuration systems. In the following we show how the concepts needed for describing configuration knowledge can be represented using semantic markup languages such as OIL [11] or DAML+OIL [26].

From the viewpoint of industrial software development, the integration of construction and maintenance of knowledge-based systems is an important prerequisite for a broader application of AI technologies. When considering configuration systems, formal knowledge representation languages are difficult to communicate to domain experts. The so-called knowledge acquisition bottleneck is obvious, since configuration knowledge acquisition and maintenance are only feasible with the support of a knowledge engineer who can handle the formal representation language of the underlying configuration system.

The Unified Modeling Language (UML) [21] is a widely adopted modeling language in industrial software development. Based on our experiences in building configuration knowledge bases using UML [9], we show how to effectively support the construction of Semantic Web configuration knowledge bases using UML as knowledge acquisition frontend. The provided UML concepts constitute an ontology consisting of concepts contained in de facto standard configuration ontologies [9,24]. Based on a description logic based definition of a configuration task we provide a set of rules for automatically translating UML configuration models into a corresponding OIL representation[2].

The approach presented in this paper enhances the application of Software Engineering techniques to knowledge-based systems by providing a UML-based knowledge acquisition frontend for configuration systems. Vice versa, reasoning support for Semantic Web ontology languages can be exploited for checking the consistency of UML configuration models. The resulting configuration knowledge bases enable knowledge interchange between heterogenous configuration environments as well as distributed configuration problem solving in different supply chain settings. The presented concepts are implemented in a knowledge acquisition workbench which is a major part of the CAWICOMS configuration environment.

The paper is organized as follows. In Section 2 we give an example of a UML configuration knowledge base which is used for demonstration purposes throughout the paper. In Section 3 we give a description logic based definition of a configuration task - this definition serves as basis for the translation of UML configuration models into a corresponding OIL-based representation (Section 4). Section 5 discusses related work.

2 Configuration knowledge base in UML

The Unified Modeling Language (UML) [21] is the result of an integration of object-oriented approaches of [4,16,20] which is well established in industrial software development. UML is applicable throughout the whole software development process

[2] Note that OIL text is used for presentation purposes - the used concepts can simply be transformed into a DAML+OIL representation.

from the requirements analysis phase to the implementation phase. In order to allow the refinement of the basic meta-model with domain-specific modeling concepts, UML provides the concept of *profiles* - the configuration domain specific modeling concepts presented in the following are the constituting elements of a UML *configuration profile* which can be used for building configuration models. UML profiles can be compared with ontologies discussed in the AI literature, e.g. [6] defines an ontology as a theory about the sorts of objects, properties of objects, and relationships between objects that are possible in a specific domain. UML *stereotypes* are used to further classify UML meta-model elements (e.g. classes, associations, dependencies). Stereotypes are the basic means to define domain-specific modeling concepts for profiles (e.g. for the configuration profile). In the following we present a set of rules allowing the automatic translation of UML configuration models into a corresponding OIL representation.

For the following discussions the simple UML configuration model shown in Figure 1 will serve as a working example. This model represents the generic product structure, i.e. all possible variants of a configurable $Computer$. 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 concepts stem from connection-based [19], resource-based [17], and structure-based [25] configuration approaches. These configuration domain-specific concepts represent a basic set useful for building configuration knowledge bases and mainly correspond to those defined in the de facto standard configuration ontologies [9,24]:

Component types. Component types represent the basic building blocks a final product can be built of. Component types are characterized by attributes. A stereotype *Component* is introduced, since some limitations on this special form of class must hold (e.g. there are no methods).

Generalization hierarchies. Component types with a similar structure are arranged in a generalization hierarchy (e.g. in Figure 1 a *CPU1* is a special kind of *CPU*).

Part-whole relationships. Part-whole relationships between component types state a range of how many subparts an aggregate can consist of (e.g. a *Computer* contains at least one and at most two motherboards - *MBs*).

Compatibilities and requirements. Some types of components must not be used together within the same configuration - they are incompatible (e.g. an *SCSIUnit* is incompatible with an *MB1*). In other cases, the existence of one component of a specific type requires the existence of another special component within the configuration (e.g an *IDEUnit* requires an *MB1*). The compatibility between different component types is expressed using the stereotyped association *incompatible*. Requirement constraints between component types are expressed using the stereotype *requires*.

Resource constraints. Parts of a configuration task can be seen as a resource balancing task, where some of the component types produce some resources and others are consumers (e.g., the consumed hard-disk capacity must not exceed the provided hard-disk capacity). Resources are described by a stereotype *Resource*, furthermore stereotyped dependencies are introduced for representing the producer/consumer relationships between different component types. Producing component types are related to resources using the *produces* dependency, furthermore consuming component types are related to resources using the *consumes* dependency. These dependencies are annotated with values representing the amount of production and consumption.

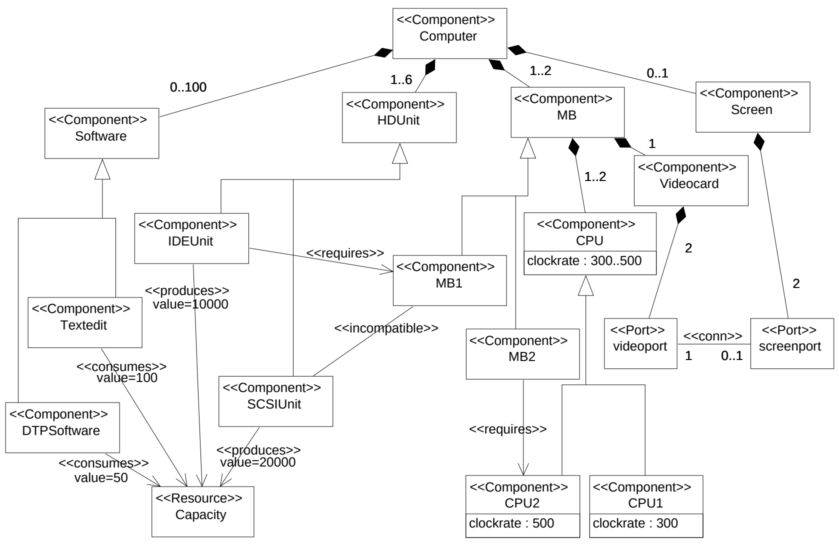

Fig. 1. Example configuration model

Port connections. In some cases the product topology - i.e., exactly how the components are interconnected - is of interest in the final configuration. The concept of a port (stereotype *Port*) is used for this purpose (e.g. see the connection between *Videocard* and *Screen* represented by the stereotype *conn* and the ports $videoport$ and $screenport$).

3 Description logic based definition of a configuration task

The following description logic based definition of a configuration task [10] serves as a foundation for the formulation of rules for translating UML configuration models into a corresponding OIL representation[3]. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [5], where $\mathcal{RN}$ is a disjunctive union of roles and features.

[3] A detailed discussion on the equivalence of configuration problems defined in description logics and those defined in predicate logic can be found in [10].

Definition 1 (Configuration task): In general we assume a configuration task is described by a triple $(DD, SRS, CLANG)$. DD represents the domain description of the configurable product and SRS specifies the particular system requirements defining an individual configuration task instance. $CLANG$ comprises a set of concepts $C_{Config} \subseteq \mathcal{CN}$ and a set of roles $R_{Config} \subseteq \mathcal{RN}$ which serve as a configuration language for the description of actual configurations. A configuration knowledge base KB = $DD \cup SRS$ is constituted of sentences in a description language. □

In addition we require that roles in $CLANG$ are defined over the domains given in C_{Config}, i.e. $range(R_i)$ = $CDom$ and $dom(R_i)$ = $CDom$ must hold for each role $R_i \in R_{Config}$, where $CDom \doteq \bigsqcup_{C_i \in C_{config}} C_i$. We impose this restriction in order to assure that a configuration result only contains individuals and relations with corresponding definitions in C_{Config} and R_{Config}. The derivation of DD will be discussed Section 4, an example for SRS could be "two $CPUs$ of type $CPU1$ and one CPU of type $CPU2$", i.e. SRS={(instance-of $c1$, $CPU1$), (instance-of $c2$, $CPU1$), (instance-of $c3$, $CPU2$)}, where $CLANG$={$CPU1$, $CPU2$, ...}.

Based on this definition, a corresponding configuration result (solution) is defined as follows [10], where the semantics of description terms are given using an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a domain of values and $(\cdot)^{\mathcal{I}}$ is a mapping from concept descriptions to subsets of $\Delta^{\mathcal{I}}$ and from role descriptions to sets of 2-tuples over $\Delta^{\mathcal{I}}$.

Definition 2 (Valid configuration): Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$ be a model of a configuration knowledge base KB, $CLANG = C_{config} \cup R_{config}$ a configuration language, and $CONF = COMPS \cup ROLES$ a description of a configuration. $COMPS$ is a set of tuples $\langle C_i, INDIVS_{C_i} \rangle$ for every $C_i \in C_{config}$, where $INDIVS_{C_i} = \{ci_1, \ldots, ci_{n_i}\} = C_i^{\mathcal{I}}$ is the set of individuals of concept C_i. These individuals identify components in an actual configuration. $ROLES$ is a set of tuples $\langle R_j, TUPLES_{R_j} \rangle$ for every $R_j \in R_{config}$ where $TUPLES_{R_j} = \{\langle rj_1, sj_1 \rangle, \ldots, \langle rj_{m_j}, sj_{m_j} \rangle\} = R_j^{\mathcal{I}}$ is the set of tuples of role R_j defining the relation of components in an actual configuration. □

A valid configuration for our example domain is $CONF$=$\{\langle CPU1, \{c1, c2\} \rangle, \langle CPU2, \{c3\} \rangle, \langle MB1, \{m1\} \rangle, \langle MB2, \{m2\} \rangle, \langle$*mb-of-cpu*$, \{\langle m1, c1 \rangle, \langle m1, c2 \rangle, \langle m2, c2 \rangle\} \rangle, ...\}$.

The automatic derivation of an OIL-based configuration knowledge base requires a clear definition of the semantics of the used UML modeling concepts. In the following we define the semantics of UML configuration models by giving a set of corresponding translation rules into OIL. The resulting knowledge base restricts the set of possible configurations, i.e. enumerates the possible instance models which strictly correspond to the UML class diagram defining the product structure.

4 Translation of UML configuration model into OIL

For the modeling concepts discussed in Section 2 we now present a set of rules for translating those concepts into an OIL-based representation. This mapping is a basis for the representation of configurator capabilities as configuration Web service.

In the following $GREP$ denotes the graphical representation of the UML configuration model.

Rule 1 (Component types): Let c be a component type, a an attribute of c, and d be the domain of a in $GREP$, then DD is extended with

class-def c.
slot-def a.
c: slot-constraint a cardinality 1 d.

For those component types $c_i, c_j \in \{c_1, ..., c_m\}$ ($c_i \neq c_j$), which do not have any supertypes in $GREP$, DD is extended with

disjoint c_i, c_j. □

Example 1 (Component type CPU): class-def CPU.
slot-def $clockrate$.
CPU: slot-constraint $clockrate$
cardinality 1 ((min 300) and (max 500)).
disjoint CPU MB. disjoint MB $Screen$. ... □

Subtyping in the configuration domain means that attributes and roles of a given component type are inherited by its subtypes. In most configuration environments a disjunctive and complete semantics is assumed for generalization hierarchies, where the disjunctive semantics can be expressed using the $disjoint$ axiom and the completeness can be expressed by forcing the superclass to conform to one of the given subclasses as follows.

Rule 2 (Generalization hierarchies): Let u and $d_1, ..., d_n$ be classes (component types) in $GREP$, where u is the superclass of $d_1, ..., d_n$, then DD is extended with
$d_1, ..., d_n$: subclass-of u.
u: subclass-of (d_1 or ... or d_n).
$\forall$ d_i, $d_j \in \{d_1, ..., d_n\}$ ($d_i \neq d_j$) : disjoint d_i d_j. □

Remark 1: Attribute and role inheritance must not be addressed in the translation rules for OIL since they are defined in OIL. □

Example 2 ($CPU1$, $CPU2$ subclasses of CPU): $CPU1$: subclass-of CPU.
$CPU2$: subclass-of CPU.
CPU: subclass-of ($CPU1$ or $CPU2$).
disjoint $CPU1$ $CPU2$. □

Part-whole relationships are important model properties in the configuration domain. In [1,23] it is pointed out that part-whole relationships have quite variable semantics depending on the regarded application domain. In most configuration environments, a part-whole relationship is described by the two basic roles *partof* and *haspart*. Depending on the intended semantics, different additional restrictions can be placed on the usage of those roles.

UML provides two different facets of part-whole relationships which are also widely used for configuration problem representation, namely *composite* and *shared* part-whole relationships. If a component is a compositional part of another component then strong ownership is required, i.e., it can not be part of another component at the same time. If a component is a non-compositional (shared) part of another component, it can be shared between different components. Multiplicities used to describe a part-whole relationship denote how many parts the aggregate can consist of and between how many aggregates a part can be shared if the aggregation is non-composite. In our $Computer$ configuration example we only use composite part-whole relationships.

Rule 3 (Composite part-whole relationships): Let w and p be component types in $GREP$, where p is a compositional part of w and ub_p is the upper bound, lb_p the lower bound of the multiplicity of the part, and ub_w is the upper bound, lb_w the lower bound of the multiplicity of the whole. Furthermore let *w-of-p* and *p-of-w* denote the names of the roles of the part-whole relationship between w and p, where *w-of-p* denotes the role connecting the part with the whole and *p-of-w* denotes the role connecting the whole with the part, i.e., *p-of-w* $\sqsubseteq haspart$, *w-of-p* $\sqsubseteq partof_{composite}$, and $partof_{composite}$ $\sqsubseteq partof$.
DD is extended with
slot-def *w-of-p* subslot-of $partof_{composite}$
inverse *p-of-w* domain p range w.
slot-def *p-of-w* subslot-of *haspart* inverse *w-of-p*
domain w range p.
p: slot-constraint *w-of-p* min-cardinality lb_w w.
p: slot-constraint *w-of-p* max-cardinality ub_w w.
p: slot-constraint $partof$ cardinality 1 w.
w: slot-constraint *p-of-w* min-cardinality lb_p p.
w: slot-constraint *p-of-w* max-cardinality ub_p p. □

Remark 2: The semantics of *shared* part-whole relationships ($partof_{shared} \sqsubseteq partof$) are defined by simply restricting the upper bound and the lower bound of the corresponding roles, i.e., the constraint on the $partof$ role cardinality (see Rule 3) can be omitted - this constraint expresses the fact that when introducing an exclusive part-whole relationship, the part component must be connected to exactly one whole and no additional part-whole relationships of the part are allowed. □

Example 3 (MB partof $Computer$): slot-def *computer-of-mb*
subslot-of $partof_{composite}$
inverse *mb-of-computer*
domain MB range $Computer$.
slot-def *mb-of-computer* subslot-of $haspart$
inverse *computer-of-mb*
domain $Computer$ range MB.
MB: slot-constraint *computer-of-mb*
min-cardinality 1 $Computer$.
MB: slot-constraint *computer-of-mb*
max-cardinality 1 $Computer$.
MB: slot-constraint $partof$
cardinality 1 $Computer$.
$Computer$: slot-constraint *mb-of-computer*
min-cardinality 1 MB.

$Computer$: slot-constraint *mb-of-computer*
max-cardinality 2 MB. □

Assumptions concerning partof structures. In the following we show how the constraints contained in a UML product configuration model (e.g., an $IDEUnit\ requires$ an $MB1$) can be translated into a corresponding OIL representation. For a consistent application of the translation rules it must be ensured that the involved components are within the same sub-configuration w.r.t. the part-of hierarchy, i.e., the involved components must be connected to the same instance of the component type that represents the common root for these components. In order to allow a correct derivation of constraints, the involved component types must have a unique common component type as predecessor and a unique path to the common root in $GREP$ (in Figure 2 the component type *Computer* is the unique common root of the component types *IDEUnit* and *CPU1*). If this uniqueness property is not satisfied, the meaning of the imposed (graphically represented) constraints becomes ambiguous, since one component can be part of more than one substructure and consequently the scope of the constraint becomes ambiguous.

For the derivation of constraints on the product model we introduce the abbreviation $navpath$ representing a navigation expression over roles. For the definition of $navpath$ the UML configuration model can be interpreted as a directed graph, where component types are represented by vertices and part-whole relationships are represented by edges. Because of the inheritance properties of roles, $partof$ ($haspart$) roles are inherited by the leaf nodes of generalization hierarchies. Consequently, generalization hierarchies do not influence the construction of navigation paths.

Definition 3 (Navigation expression): Let $path(c_1, c_n)$ be a path from a component type c_1 to a component type c_n in $GREP$ represented through a sequence of expressions of the form $haspart(C_i, C_j, Name_{Ci})$ denoting a direct partof relationship between the component types C_i and C_j, where $Name_{Ci}$ represents the name of the corresponding $haspart$ role, i.e., $path(c_1, c_n)$ =
$< haspart(c_1, c_2, name_{c1}),$
$haspart(c_2, c_3, name_{c2}), ...,$
$haspart(c_{n-1}, c_n, name_{cn-1}) >$
Based on the definition of $path(c_1, c_n)$ we can define $navpath(c_1, c_n)$ for DD as
c_1: slot-constraint $name_{c1}$
has-value(slot-constraint $name_{c2}$ ...
has-value(slot-constraint $name_{cn-1}$
has-value c_n)...). □

Example 4 ($navpath(Computer, CPU1)$): $Computer$: slot-constraint *mb-of-computer*
has-value (slot-constraint *cpu-of-mb*
has-value $CPU1$). □

The concept of a *common root* is based on the definition of $navpath$ as follows.

Definition 4 (Common root): A component type r is denoted as common root of the component types c_1 and c_2 in $GREP$, *iff* there exist navigation paths $path(r, c_1)$,

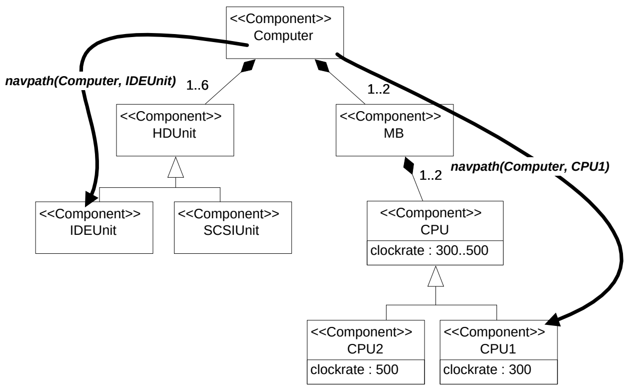

Fig. 2. Navigation paths from $Computer$ to $CPU1$ and $IDEUnit$

$path(r, c_2)$ and there does not exist a component type r', where r' is a direct or indirect part of r with navigation paths $path(r', c_1)$, $path(r', c_2)$. □

When regarding the example configuration model of Figure 1, MB is the common root of CPU and $Videocard$. Conform to Definition 4 the component type $Computer$ is not a common root of CPU and $Videocard$.

Requires constraints. A requires constraint between two component types c_1 and c_2 in $GREP$ denotes the fact that the existence of an instance of component type c_1 requires that an instance of component type c_2 exists and is part of the same (sub)configuration.

Rule 4 (Requires constraints): Given the relationship c_1 *requires* c_2 between the component types c_1 and c_2 in $GREP$ with r as common root of c_1 and c_2, then DD is extended with

r: ((not($navpath(r, c_1)$)) or $navpath(r, c_2)$). □

The condition part of the implication describes a path from the common root to the component c_1; the consequence contains a corresponding path to the required component c_2.

Example 5 ($IDEUnit$ requires $MB1$): $Computer$: ((not (slot-constraint
hdunit-of-computer has-value $IDEUnit$))
or (slot-constraint *mb-of-computer*
has-value $MB1$)) □

Compatibility constraints. A compatibility constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ denotes the fact that the existence of a tuple of instances corresponding to the types in c is not allowed in a final configuration (result).

Rule 5 (Compatibility constraints): Given a compatibility constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ with r as common root of $\{c_1, c_2, ..., c_n\}$, then DD is extended with
r: (not(($navpath(r, c_1)$) and
($navpath(r, c_2)$) and ... and
($navpath(r, c_n)$)...). □

Example 6 ($SCSIUnit$ incompatible with $MB1$): $Computer$: (not ((slot-constraint *hdunit-of-computer* has-value $SCSIUnit$) and
(slot-constraint *mb-of-computer*
has-value $MB1$))). □

Resource constraints. Resource constraints can be modeled in UML using stereotyped classes representing types of resources and stereotyped dependencies with a corresponding tagged value indicating resource production and consumption. Resource balancing tasks [17] are defined within a (sub)tree (context) of the product configuration model. To map a resource balancing task into OIL, additional attributes (res_p and res_c in the following) have to be defined for the component types acting as producers and consumers. Additionally we have to introduce aggregate functions as representation concepts, which are currently neither supported in OIL nor DAML+OIL. The following representation of aggregate functions is based on the formalism presented in [2], where a set of predicates P_i associated with binary relations (e.g., $\leq$, $\geq$, $<$, $>$) over a value domain $dom(D)$ and a set of aggregation functions $agg(D)$ (e.g., $count, min, max, sum$) are defined. When regarding the path leading to the concept whose feature values are aggregated, [2] require that all but the last one of the roles in the path must be features - this assumption is taken into account in the following formalization of resource constraints.

Rule 6 (Resource constraints): Let p = $\{p_1, p_2, ..., p_n\}$ be producing component types and c = $\{c_1, c_2, ..., c_m\}$ be consuming component types of resource res in $GREP$. Furthermore, let res_p be a feature common to all component types in p, and res_c be a feature common to the types in c, where the values of res_p and res_c are defined by the tagged values of the consumes and produces dependencies in $GREP$.

A resource constraint for DD can be expressed as r : $P(\Sigma(navpath\ (r, p) \circ res_p)$, $\Sigma(navpath\ (r, c) \circ res_c)$, where r represents the common root of the elements in c and p.

Note, that r : $P(\Sigma(navpath\ (r, p) \circ res_p)$, $\Sigma(navpath\ (r, c) \circ res_c)$ represents the concept $P(r_1^1\ r_2^1\ ...\ r_{n-1}^1\ \Sigma(r_n^1 \circ res_p), r_1^2\ r_2^2\ ...\ r_{m-1}^2\ \Sigma(r_m^2 \circ res_c))$. □

Example 7 (Capacity needed by $Software$ $\leq$ Capacity provided by $HDUnit$):
$DTPSoftware$: slot-constraint $Capacity$
cardinality 1 (equal 50).
$Textedit$: slot-constraint $Capacity$
cardinality 1 (equal 100).
$SCSIUnit$: slot-constraint $Capacity$
cardinality 1 (equal 20000).

$IDEUnit$: slot-constraint $Capacity$
cardinality 1 (equal 10000).
$Computer$: lesseq
sum(*sw-of-computer*$\circ Capacity$),
sum(*hdunit-of-computer*$\circ Capacity$). $\Box$

Port connections. Ports in the UML configuration model represent physical connection points between components (e.g., a $Videocard$ can be connected to a $Screen$ using the port combination $videoport_1$ and $screenport_2$). In UML we introduce ports using classes with stereotype $Port$ - these ports are connected to component types using part-whole relationships.

In order to represent port connections in OIL, we introduce a concept $Port$ which is described by a role indicating the related component concept (role $compnt$) and a role $portname$ which describes the portname of the connection. Finally, the role $conn$ describes the relation to the second involved port concept. Note, that the inverse of the role $conn$ is the role $conn$ of the connected port.

Rule 7 (Port types): Let $\{a, b\}$ be component types in $GREP$, $\{pa, pb\}$ be the corresponding connected port types, $\{m_a, m_b\}$ the multiplicities of the port types w.r.t. $\{a, b\}$[4], and $\{\{lb_{pa}, ub_{pa}\}, \{lb_{pb}, lb_{pb}\}\}$ the lower bound and upper bound of the multiplicities of the port types w.r.t. $\{pa, pb\}$, then DD is extended with
class-def pa subclass-of Port.
class-def pb subclass-of Port.
pa: slot-constraint $portname$
cardinality 1 (one-of pa_1 ... pa_{ma}).
pa: slot-constraint $conn$ min-cardinality lb_{pa} pb.
pa: slot-constraint $conn$ max-cardinality ub_{pa} pb.
pa: slot-constraint $conn$ value-type pb.
pa: slot-constraint $compnt$ cardinality 1 a.
pb: slot-constraint $portname$
cardinality 1 (one-of pb_1 ... pb_{mb}).
pb: slot-constraint $conn$ min-cardinality lb_{pb} pa.
pb: slot-constraint $conn$ max-cardinality ub_{pb} pa.
pb: slot-constraint $conn$ value-type pa.
pb: slot-constraint $compnt$ cardinality 1 b. $\Box$

Example 8 ($Videocard$ connected to $Screen$): class-def $videoport$ subclass-of Port.
class-def $screenport$ subclass-of Port.
$videoport$: slot-constraint $portname$
cardinality 1 one-of ($videoport_1$ $videoport_2$).
$videoport$: slot-constraint $conn$
min-cardinality 0 $screenport$.
$videoport$: slot-constraint $conn$
max-cardinality 1 $screenport$.

[4] No different lower bounds and upper bounds are allowed here.

$videoport$: slot-constraint $conn$
value-type $screenport.$
$videoport$: slot-constraint $compnt$
cardinality 1 $Videocard.$ ... $\Box$

Note, that the above definitions for ports do not guarantee that the connections are established within a sub-configuration. In order to formulate such restrictions, variables are needed as placeholders for the corresponding connection individuals - the usage of variables is not supported in current versions of Semantic Web ontology languages such as OIL or DAML+OIL.

Using the defined structure for port connections, the constraint "a $Videocard$ must be connected via $videoport_1$ with a $Screen$ via $screenport_1$" can be formulated as follows.

Example 9: $Videocard$: (slot-constraint *videoport-of-videocard* has-value
((slot-constraint $portname$ has-value
(one-of $videoport_1$)) and
(slot-constraint $conn$ has-value
((slot-constraint $compnt$ has-value $Screen$) and
(slot-constraint $portname$ has-value
(one-of $screenport_1$)))))). $\Box$

The application of the modeling concepts presented in this paper has its limits when building configuration knowledge bases - in most domains there exist complex constraints that do not have an intuitive graphical representation. Happily, (with some minor restrictions discussed in [10]) we are able to represent such constraints using languages such as OIL or DAML+OIL. UML itself has an integrated constraint language (Object Constraint Language - OCL [27]) which allows the formulation of constraints on object structures. The translation of OCL constraints into representations of Semantic Web ontology languages is the subject of future work, a translation into a predicate logic based representation of a configuration problem has already been discussed in [8]. The current version of our prototype workbench supports the generation of OIL-based configuration knowledge bases from UML models which are built using the modeling concepts presented in this paper, i.e. concepts for designing the product structure and concepts for defining basic constraints (e.g. *requires*) on the product structure.

5 Related Work

The definition of a common representation language to support knowledge interchange between and integration of different knowledge-based systems are important issues in the configuration domain. In [24] one approach to collect relevant concepts for modeling configuration knowledge bases is presented. The defined ontology is based on Ontolingua [15] and represents a synthesis of resource-based, function-based, connection-based, and structure-based configuration approaches. This ontology is a kind of meta-ontology which is similar to the UML profile for configuration models presented in this paper. Conforming to the definition of [6] a UML configuration model is an ontology, i.e. it restricts the sort of objects relevant for the domain, defines the possible properties of objects and the relationships between objects. Compared to the approach presented in this paper, [24] do not provide a formal semantics for the proposed modeling concepts.

The work of [7] shows some similarities to the work presented in this paper. Starting with a UML ontology (which is basically represented as a class diagram) corresponding JAVA classes and RDF documents are generated. The work presented in this paper goes one step further by providing a UML profile for the configuration domain and a set of rules allowing the automatic derivation of executable configuration knowledge bases. The correspondence between Semantic Web ontology languages and UML is shown on the object level as well as on the constraint level, where a set of domain specific constraints (e.g. *requires*) are introduced as stereotypes in the configuration profile - for these constraints an representation in OIL has been shown.

Most of the required means for expressing configuration knowledge are already provided by current versions of Semantic Web knowledge representation languages. However, in order to provide full fledged configuration knowledge representation, certain additional expressivity properties must be fulfilled - this issue is discussed in [10], where aggregation functions, n-ary relationships, and the provision of variables have been identified as the major required add-ons for ontology languages such as DAML+OIL. Within the Semantic Web community there are ongoing efforts to increase the expressiveness of Web ontology languages. DAML-L [18] is a language which builds upon the basic concepts of DAML. XML Rules [14] and CIF (Constraint Interchange Format) [13] are similar approaches with the goal to provide rule languages for the Semantic Web.

6 Conclusions

In this paper we presented an approach to integrate the development of configuration knowledge bases for the Semantic Web into standard industrial software development processes. Founded on a description logic based definition of a configuration task, we presented a set of rules for translating UML configuration models into a corresponding OIL-based representation enabling model checking for UML configuration models and knowledge sharing between different configurators in Web-based environments. Our approach supports effective sharing and integration of configuration knowledge on a graphical level which becomes one of the major issues in the context of distributed configuration problem solving. The concepts presented in this paper are implemented in a corresponding configuration knowledge acquisition workbench.

References

1. A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-Whole Relations in Object-Centered Systems: An Overview. *Data & Knowledge Engineering*, 20(3):347–383, 1996.
2. F. Baader and U. Sattler. Description Logics with Concrete Domains and Aggregation. In *Proceedings of the* 13^{th} *European Conference on Artificial Intelligence (ECAI '98)*, pages 336–340, Brighton, UK, 1998.
3. V.E. Barker, D.E. O'Connor, J.D. Bachant, and E. Soloway. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM*, 32(3):298–318, 1989.

4. G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley Object Technology Series, 1994.
5. A. Borgida. On the relative expressive power of description logics and predicate calculus. *Artificial Intelligence*, 82:353–367, 1996.
6. B. Chandrasekaran, J. Josephson, and R. Benjamins. What Are Ontologies, and Why do we Need Them? *IEEE Intelligent Systems*, 14,1:20–26, 1999.
7. S. Cranefield. UML and the Semantic Web. In *Semantic Web Working Symposium*, Stanford, CA, USA, 2001.
8. A. Felfernig, G. Friedrich, and D. Jannach. Generating product configuration knowledge bases from precise domain extended UML models. In *Proceedings of the* 12^{th} *International Conference on Software Engineering and Knowledge Engineering (SEKE'2000)*, pages 284–293, Chicago, USA, 2000.
9. A. Felfernig, G. Friedrich, and D. Jannach. UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 10(4):449–469, 2000.
10. A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. A Joint Foundation for Configuration in the Semantic Web. *Technical Report KLU-IFI-02-05*, 2001.
11. D. Fensel, F. vanHarmelen, I. Horrocks, D. McGuinness, and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
12. G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.
13. P. Gray, K. Hui, and A. Preece. An Expressive Constraint Language for Semantic Web Applications. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 46–53, Seattle, WA, 2001.
14. B.N. Grosof. Standardizing XML Rules. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 2–3, Seattle, WA, 2001.
15. T. Gruber. Ontolingua: A mechanism to support portable ontologies. *Technical Report KSL 91-66*, 1992.
16. I. Jacobson, M. Christerson, and G. Övergaard. *Object-oriented Software Engineering - A Use-Case Driven Approach*. Addison- Wesley, 1992.
17. E.W. Jüngst M. Heinrich. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings of the* 7^{th} *IEEE Conference on AI applcia-tions (CAIA)*, pages 257–264, Miami, FL, USA, 1991.
18. Sh. McIlraith, T.C. Son, and H. Zeng. Mobilizing the Semantic Web with DAML-Enabled Web Services. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 29–39, Seattle, WA, 2001.
19. S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings* 11^{th} *International Joint Conf. on Artificial Intelligence*, pages 1395–1401, Detroit, MI, 1989.
20. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. New Jersey, USA, 1991.
21. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
22. D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. In B. Faltings and E. Freuder, editors, *IEEE Intelligent Systems, Special Issue on Configuration*, volume 13, pages 50–58. IEEE, 1998.
23. U. Sattler. Description Logics for the Representation of Aggregated Objects. In *Proceedings of the* 14^{th} *European Conference on Artificial Intelligence (ECAI 2000)*, pages 239–243, Berlin, Germany, 2000.

24. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a General Ontology of Configuration. *AI Engineering Design Analysis and Manufacturing Journal, Special Issue: Configuration Design*, 12(4):357–372, 1998.
25. M. Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2), June, 1997.
26. F. vanHarmelen, P.F. Patel-Schneider, and I. Horrocks. A Model-Theoretic Semantics for DAML+OIL. *www.daml.org*, March 2001.
27. J. Warmer and A. Kleppe. *The Object Constraint Language - Precise Modeling with UML*. Addison Wesley Object Technology Series, 1999.
28. J.R. Wright, E. Weixelbaum, G.T. Vesonder, K.E. Brown, S.R. Palmer, J.I. Berman, and H.H. Moore. A Knowledge-Based Configurator that supports Sales, Engineering, and Manufacturing at AT&T Network Systems. *AI Magazine*, 14(3):69–80, 1993.