Transforming UML Domain Descriptions into Configuration Knowledge Bases for the Semantic Web

Alexander Felfernig[1] Gerhard Friedrich[1] Dietmar Jannach[1]
Markus Stumptner[2] Markus Zanker[1]

[1] Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik
Universität Klagenfurt, Universitätsstr. 65-67,
9020 Klagenfurt, Austria
felfernig,friedrich,jannach,zanker@ifit.uni-klu.ac.at

[2] Advanced Computing Research Center
University of South Australia
5095 Mawson Lakes (Adelaide), SA, Australia
mst@cs.unisa.edu.au

Abstract. In this chapter we emphasize how we can integrate Web-based sales systems for highly complex customizable products and services by making use of the descriptive representation formalisms of the Semantic Web. Building on the equivalence between the consistency based definition and the description logic based definition of configuration [1], we are capable of transforming our application domain-independent meta-model for modeling configuration knowledge [2] into the emerging standards of the Semantic Web initiative, such as DAML+OIL. Furthermore, we discuss how these standardized description languages can be used to derive capability descriptions for semantic configuration Web services.

1 Introduction

The easy access to vast information resources offered by the World Wide Web (WWW) opens new perspectives for conducting business. One of these is the goal of the research project CAWICOMS to enable configuration systems to deal simultaneously with configurators of multiple suppliers over the Web [3]. The technical approach taken resembles state-of-the-art electronic marketplaces. Many-to-many relationships between customers and different suppliers are enabled, thus replacing inflexible one-to-one relations dating to the pre-internet era of EDI (electronic data interchange). We resolved the problem of heterogeneity of product and catalogue descriptions by imposing a common representation formalism for product models on all market participants based on UML. The textual representation of the graphical product models uses XML-Schema definitions. The eXtensible Markup Language (XML[1]) serves as a flexible data format definition language that allows to communicate tree structures

[1]See http://www.w3c.org/xml for reference.

with a linear syntax. However, single standards for conducting B2B electronic commerce will not exist. As content transformation between those catalog and document standards is far from being a trivial task [4], the Semantic Web offers the perspective of dynamic knowledge transformations by reasoning on semantics.

In this chapter we will outline how configuration systems can flexibly cooperate in an ontology-based approach through the use of the Web service paradigm. The capability of each configuration system can be expressed by the evolving language standards of the Semantic Web initiative [5, 6]. In [1] it is shown that a consistency based and a description logic based configuration knowledge representations are equivalent under certain restrictions. Therefore, product model representations in our UML-based meta-model for configuration knowledge representation can be transformed into OIL resp. DAML+OIL [7]. We showed the transformation of the configuration meta-model into a configuration knowledge base using predicate logic already in [2].

In Section 2 we will give an example configuration model and in Section 3 a description logic based definition of configuration is given. Examples for the translation rules into a corresponding OIL (Section 4) representation and a discussion on semantic configuration Web services (Section 5) finally follow.

2 Configuration knowledge base in UML

The Unified Modeling Language (UML) [8] is the result of an integration of object-oriented approaches of [9, 10, 11] which is well established in industrial software development. UML is applicable throughout the whole software development process 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. [12] 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 [13], resource-based [14], and structure-based [15] 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 [2, 16]:

- **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 specific 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.

- **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 [1] serves as a foundation for the formulation of rules for translating UML configuration models into a corresponding OIL representation[2]. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [17], where $\mathcal{RN}$ is a disjunctive union of roles and features.

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

[2] In the following we assume that the reader is familiar with the concepts of OIL. See [7] for an introductory text.

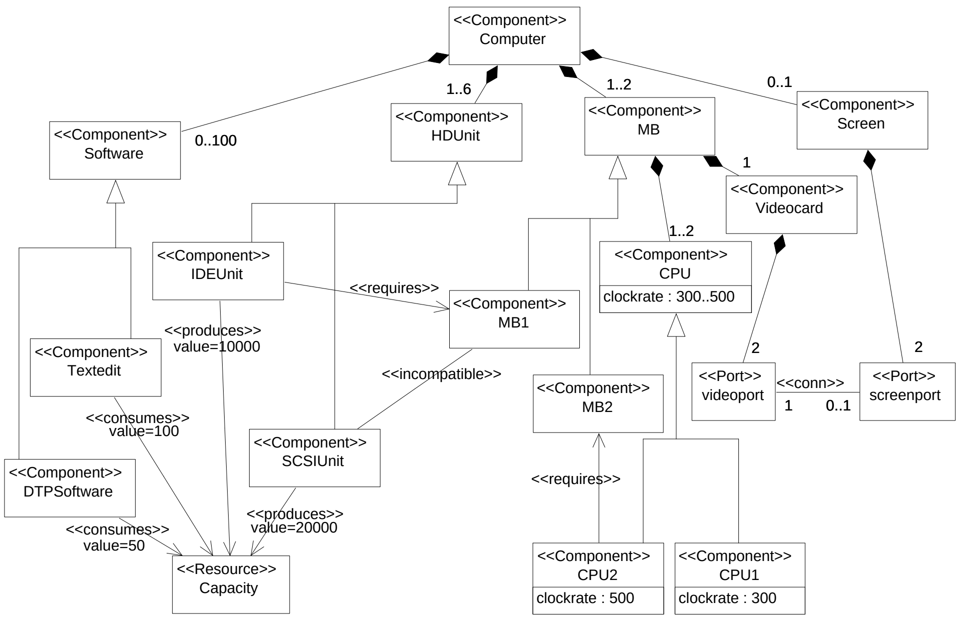

Figure 1: Example configuration model

$R_{Config} \subseteq \mathcal{RN}$ 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 in the next section, 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 [18], 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 $CONF$=$\{\langle CPU1, \{c1, c2\}\rangle, \langle CPU2, \{c3\}\rangle, \langle MB1, \{m1\}\rangle, \langle MB2, \{m2\}\rangle$, ⟨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.

The equivalence between the definitions of valid configurations in description logic (Definition 2) and the definition in first-order predicate logic [19] is shown in [1]. It refers to the translation function $\mathcal{T}\langle\cdot\rangle$ in [17] that translates concepts, roles, terms, and axioms of a description language ($\mathcal{DL}$) without transitive closure into equivalent formulas in the first order predicate logic $\ddot{\mathcal{L}}^3_{CNT}$. Note that $\ddot{\mathcal{L}}^3_{CNT}$ allows only monadic and dyadic predicates and restricts the number of counting quantifiers and subformulas to at most three free variables. Building on this equivalence we are therefore capable of transforming our UML based configuration models [2] into the Semantic Web standards, such as DAML+OIL, that are founded on description logics.

4 Translation of UML configuration models into OIL

As already pointed out in the previous section the translation rules give exact semantics to the UML modeling concepts. In [2] translation rules that transform the graphical representation into a predicate logic configuration knowledge base have been already given. In the following, $GREP$ denotes the *graphical representation* of the UML configuration model. For the model elements of $GREP$ (i.e., component types, generalization hierarchies, part-whole relationships, requirement constraints, incompatibility constraints) we propose rules for translating those concepts into a description logic based representation. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [17], where $\mathcal{RN}$ is a disjunctive union of roles and features.

Rule 1 (Component types): Let c be a component type in $GREP$, then

- DD_{DL} is extended by class-def c.

For all attributes a of c in $GREP$, and d the domain of a in $GREP$,

- DD_{DL} is extended by

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

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_{DL} is extended by

  $d_1, ..., d_n$: 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.$\square$

Example 2 ($CPU1$, $CPU2$ subclasses of CPU):

$CPU1$: subclass-of CPU.
$CPU2$: subclass-of CPU.
CPU: subclass-of ($CPU1$ or $CPU2$).
disjoint $CPU1$ $CPU2$. $\square$

Part-whole relationships are important model properties in the confi guration domain. In [20], [16], [21] it is pointed out that part-whole relationships have quite variable semantics depending on the application domain. In most confi guration 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. Note that we do not require acyclicity since particular domains such as software confi guration allow cycles on the type level. In the following we discuss two facets of part-whole relationships which are widely used for confi guration knowledge representation [16] and are also provided by UML, namely *composite* and *shared* part-whole relationships. In UML composite part-whole relationships are denoted by a black diamond, shared part-whole relationships are denoted by a white diamond[3]. If a component is a *compositional part* of another component then strong ownership is required, i.e., it must be part of exactly one component. 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*. The basic structure of a part-whole relationship is shown in Figure 2.
Rule 3 (Part-whole relationships): Let w and p be component types in $GREP$, where p is a 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* $\sqsubseteq Partof_{mode}$, where $Partof_{mode} \in \{partof_{composite}, partof_{shared}\}$. A part p can be either a shared part (concept $part_{shared}$) or a composite part (concept $part_{composite}$). Given a part-whole relationship between p and w in $GREP$, then

- DD_{DL} is extended by

[3]Note that in our $Computer$ configuration example we only use composite part-whole relationships - as mentioned in [16], composite part-whole relationships are often used when modeling physical products, whereas shared part-whole relationships are used to describe abstract entities such as services.

slot-def *w-of-p* subslot-of $Partof_{mode}$ 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: subclass-of ($part_{shared}$ or $part_{composite}$).
p: slot-constraint *w-of-p* min-cardinality lb_w w.
p: slot-constraint *w-of-p* max-cardinality ub_w w.
w: slot-constraint *p-of-w* min-cardinality lb_p p.
w: slot-constraint *p-of-w* max-cardinality ub_p p.

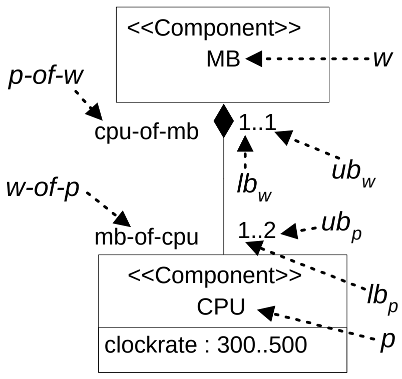

Figure 2: Part-whole relationship (p:part, w: whole)

Remark 1: The following properties have to hold for shared and composite part-whole relationships.

- Each shared part is connected to at least one whole, i.e.,

  (DD_{DL}) $part_{shared}$: slot-constraint $partof_{shared}$ min-cardinality 1 top.

- Each composite part is connected to exactly one whole, i.e.,

  (DD_{DL}) $part_{composite}$: slot-constraint $partof_{composite}$ min-cardinality 1 top.
  slot-constraint $partof_{composite}$ max-cardinality 1 top.

- A shared part cannot be a composite part at the same time, i.e.,

  (DD_{DL}) *disjoint* $part_{shared}$ $part_{composite}$

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: subclass-of ($part_{shared}$ or $part_{composite}$)
MB: slot-constraint *computer-of-mb* min-cardinality 1 $Computer$.
MB: slot-constraint *computer-of-mb* max-cardinality 1 $Computer$.
$Computer$: slot-constraint *mb-of-computer* min-cardinality 1 MB.
$Computer$: slot-constraint *mb-of-computer* max-cardinality 2 MB. □

Necessary part-of structure properties. In the following we show how the constraints contained in a product confi guration 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 components involved are parts of the same sub-confi guration, i.e., the involved components must be connected to the same instance of the component type that represents the common root[4] for these components - the components are within the same mereological context [16]. This can simply be expressed by the notion that component types in such a hierarchy must each have a unique superior component type in $GREP$. If this uniqueness property is not satisfi ed, 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 macro $navpath$ as an abbreviation for a navigation expression over roles. For the defi nition of $navpath$ the UML confi guration model can be interpreted as a directed graph, where component types are represented by vertices and part-whole relationships are represented by edges.

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.* Furthermore, *$Name_{Ci}$ represents the name of the corresponding $haspart$ role. Such a path in $GREP$* is represented as $path(c_1, c_n)$ =

$< haspart(c_1, c_2, name_{c1}), haspart(c_2, c_3, name_{c2}), ..., haspart(c_{n-1}, c_n, name_{cn-1}) >$

Based on the definition of $path(c_1, c_n)$ we can define the macro $navpath(c_1, c_n)$ as

slot-constraint $name_{c1}$
 has-value(slot-constraint $name_{c2}$ ...
  has-value(slot-constraint $name_{cn-1}$ has-value c_n)...).

For the translation into DD_{LOG} the macro $navpath(c_1, c_n)$ is defined as

$\exists Y_1, Y_2, ..., Y_{n-1}, Y_n :$

$name_{c1}(Y_1, X) \ \wedge name_{c2}(Y_2, Y_1) \wedge ... \wedge name_{cn-1}(Y_n, Y_{n-1}) \wedge c_n(Y_n),$

where X is a free variable quantified outside the scope of this expression and represents an instance of concept $c1$. □

Example 4 ($navpath(Computer, CPU1)$):

slot-constraint *mb-of-computer*
  has-value (slot-constraint *cpu-of-mb* has-value $CPU1$). □

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

Definition 4 (Nearest common root) A component type r is denoted as nearest common root of the component types c_1 and c_2 in $GREP$, iff there exist paths $path(r, c_1)$, $path(r, c_2)$ and there does not exist a component type r', where r' is a part[5] of r with paths $path(r', c_1)$, $path(r', c_2)$. □

When regarding the example confi guration model of Figure 1, MB is the nearest common root of CPU and $Videocard$. Note that the component type $Computer$ is not a nearest common root of CPU and $Videocard$ but the nearest common root of $CPU1$ and $IDEUnit$ (see Figure 3.

[4]In Figure 1 the component type *Computer* is the unique common root of *IDEUnit* and *CPU1*.

[5]In this context $partof$ is assumed to be transitive.

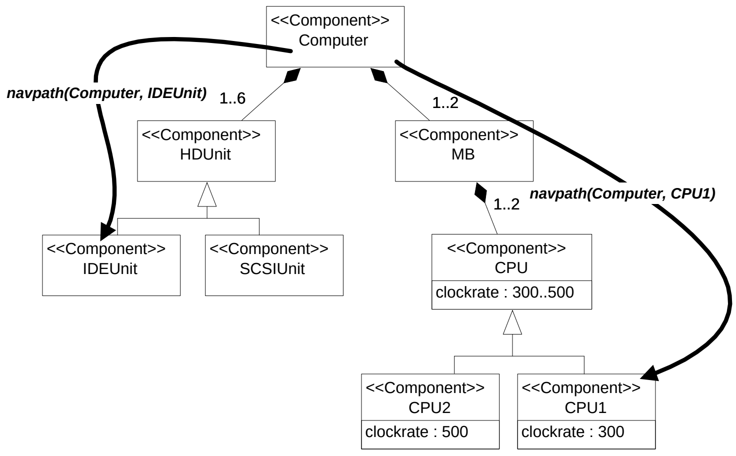

Figure 3: Navigation paths from $Computer$ to $CPU1$ and $IDEUnit$

Requirement 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 an instance of component type c_2 in the same (sub)configuration.
Rule 4 (Requirement constraints): Given the relationship c_1 *requires* c_2 between the component types c_1 and c_2 in $GREP$ with r as nearest common root of c_1 and c_2, then

- DD_{DL} is extended by r: ((not($navpath(r, c_1)$)) or $(navpath(r, c_2))$). □

The condition part of the inner implication is a path from the nearest common root to the component c_1; the consequent is a 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 $MB1$)). □

Incompatibility constraints. An *incompatibility* 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 (Incompatibility constraints): Given an *incompatibility* constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ with r as nearest common root of $\{c_1, c_2, ..., c_n\}$, then

- DD_{DL} is extended by

  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. In order to introduce resource constraints, additional expressivity requirements must be fulfilled - this issue will be discussed in [1].

Port connections. Ports in the UML configuration model (see Figure 4) 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 relationships.

In order to represent port connections in OIL, we introduce them via a separate concept $Port$[6]. The role $compnt$ indicates the component concept that the port belongs to, the role $portname$ determines its name, and the role $conn$ describes the relation to the counterpart port concept of the connected component.

Rule 6 (Port connections): 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 with respect to $\{a, b\}$[7], and $\{\{lb_{pa}, ub_{pa}\}, \{lb_{pb}, lb_{pb}\}\}$ the lower bound and upper bound of the multiplicities of the port types with respect to $\{pa, pb\}$, then

- DD_{DL} is extended by

  class-def pa subclass-of $Port$.
  class-def pb subclass-of $Port$.
  pa: slot-constraint $portname$ cardinality 1 (one-of pa_1 ... pa_{ma}).[8]
  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.

  Example 7 ($Videocard$ 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 $screenport$.
  $videoport$: slot-constraint $conn$ value-type $screenport$.
  $videoport$: slot-constraint $compnt$ cardinality 1 $Videocard$.
  $screenport$: slot-constraint $portname$ cardinality 1 (one-of $screenport_1$ $screenport_2$).

[6] Note that in OIL there are only predicates with arity 1 or 2 available, therefore the representation of port connections must be realized by the definition of additional concepts.

[7] In this context no differentiation between lower and upper bound is needed since the number of ports of a component is exactly known beforehand.

[8] In this context pa_i denotes one pa port.

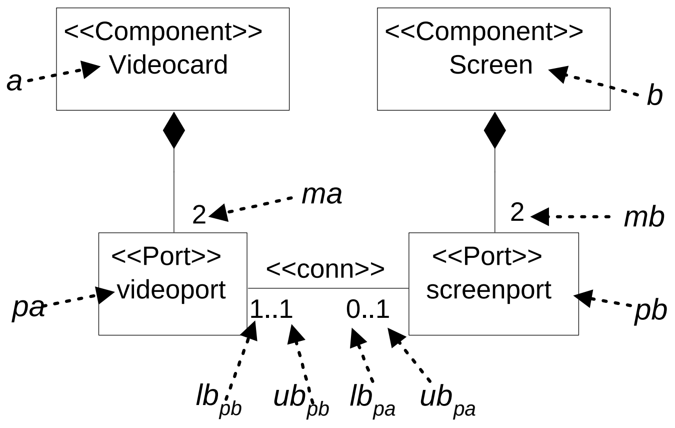

Figure 4: Ports in the configuration model

$screenport$: slot-constraint $conn$ min-cardinality 1 $videoport$.
$screenport$: slot-constraint $conn$ max-cardinality 1 $videoport$.
$screenport$: slot-constraint $conn$ value-type $videoport$.
$screenport$: slot-constraint $compnt$ cardinality 1 $Screen$. □

Using the port connection structure defined above, the constraint "a $Videocard$ must be connected via $videoport_1$ with a $Screen$ via $screenport_1$" can be written as follows.
Example 8:

$Videocard$: (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 $screenport_1$)))))). □

The application of the graphical modeling concepts presented in this paper has its limits when building complete configuration knowledge bases. Typically, they do not only include the product structure itself, but also more complex constraints that cannot be represented graphically. Happily, (with some minor restrictions discussed in [1]) we are able to represent these constraints using languages such as OIL or DAML+OIL. UML itself has an integrated constraint language (Object Constraint Language - OCL [22]) 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 [23].

5 Semantic configuration Web services

When it comes to multi-site product configuration, problem solving capabilities are distributed over several business entities that need to cooperate on a customer request for joint service provision. This Peer-to-Peer (P2P) interaction approach among a dynamic set of participants without a clear assignment of *client* and *server* roles asks for applying the paradigm of *Web services* [24]. It stands for encapsulated application logic that is open to accept requests from any peer over the Web.

Basically, a Web Service can be defined as an interface that describes a collection of provided operations. Consequently, we can interpret the application logic that configures a product (i.e. a confi gurator) as a standardized Web service. In the CAWICOMS approach [3] service requests and their replies are enabled by a *WebConnector* component that owns an object model layer that accesses the internal constraint representation of the confi guration engine. This object model layer represents the advertised confi guration service descriptions. A service requestor agent can, therefore, impose its service request via an *edit-query* onto the object-model layer and retrieves the confi guration service result via a *publish-query*. In our workbench implementation this matchmaking task is, therefore, performed as part of the search process for a confi guration solution of a constraint-based confi gurator engine. For the internal representation of the advertised service models as well as the service requests, an object-oriented framework for constraint variables is employed [25]. Reasoning on service requirements as well as on service decomposition is performed by the underlying Java-based constraint solver.

The offered Web services are employed by interface agents (i.e. the *Frontend* in terms of the CAWICOMS architecture) that interact with human users via a Web interface as well as by confi guration agents that outsource confi guration services as part of their problem solving capabilities. Formally, when implementing a Web service the following issues need to be addressed [24]:

- *Service publishing* - the provider of a service publishes the description of the service to a service registry which in our case are confi guration agents with mediating capabilities. Within this registry the basic properties of the offered confi guration service have to be defi ned in such a way that automated identifi cation of this service is possible.

- *Service identification* - the requestor of a service imposes a set of requirements which serve as the basis for identifying a suitable service. In our case, we have to identify those suppliers that are capable of supplying goods or services that match the specifi c customer requirements.

- *Service execution* - once a suitable service has been identifi ed, the requirements need to be communicated to the service agent that can be correctly interpreted and executed. UDDI, WSDL, and SOAP are the evolving technological standards that allow the invocation of remote application logic based on XML syntax.

Now, following the vision behind the Semantic Web effort [5, 6], the sharing of semantics is crucial to enable the WWW for applications. In order to have agents automatically searching, selecting and executing remote services, representation standards are needed that allow the annotation of meaning of a Web service which can then be interpreted by agents with the help of ontologies.

In the following, we sketch a Web service scenario that focuses on enabling automated procurement processes for customisable items (see Figure 5). Basically there are two different types of agents, those that only offer confi guration services (L) and those that act as suppliers as well as requestors for these services (I). The denotation of agent types derives from viewing the informational supply chain of product confi guration as a tree[9] where a confi guration system constitutes either an *inner node* (I) or a *leaf node* (L).

Agents of type I have therefore the mediating functionality incorporated that allows the offering agents to advertise their confi guration services. Matchmaking for service identifi cation is performed by the mediating capability that is internal to each confi gurator at an inner

[9]Note, that only the required configuration services are organized in a tree structure, which must not hold for the involved companies in the value chain of a product.

node. It is done on the semantic level that is eased by multi-layered ontological commitments (as discussed in the preceding subsection) among participants. It is assumed that suppliers share application domain ontologies that allow them to describe the capabilities of their offered products and services on the semantic level. An approach that abstracts from syntactical specifi cs and proposes a reasoning on the semantic level also exists for transforming standardized catalog representations in [4].

In the confi guration domain an abstract service description can be interpreted as a kind of standardized functional description of the product[10]. Furthermore, agents in the role of customers (service requestors) can impose requirements on a desired product; these requirements can be matched against the functional product description provided by the suppliers (service providers). If one or more supplier descriptions match with the imposed requirements, the corresponding confi guration service providers can be contacted in order to fi nally check the feasibility of the requirements and generate a customized product/service solution.

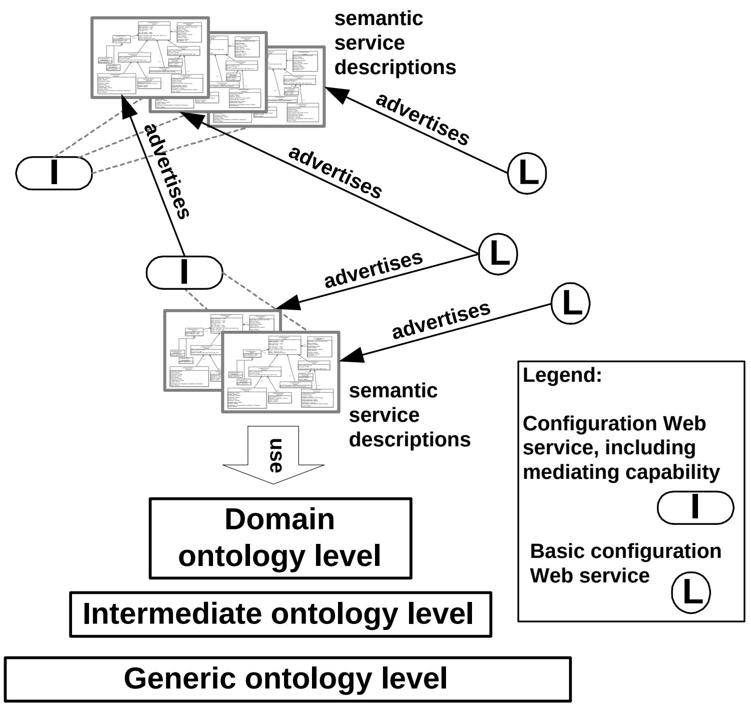
semantic service descriptions
advertises
advertises
advertises
advertises
advertises
I
I
L
L
L
semantic service descriptions
use
Domain ontology level
Intermediate ontology level
Generic ontology level
Legend:
Configuration Web service, including mediating capability
Basic configuration Web service

Figure 5: Web service scenario

Consequently, it is necessary to have semantic descriptions of the demanded services that allow us to implement effi cient matchmaking between supply and demand. Within these semantic annotations, restrictions on the domain and cardinality of slots, constraints on connections and structure, as well as the possible types of classes are possible. Furthermore, offered component instances can be represented as subconcepts (e.g. read from a catalog) of the classes of the service domain-specifi c confi guration ontology. Additional supplier-specifi c constraints can be introduced.

Therefore, markup languages are required that enable a standardized representation of service profi les for advertisement of services as well as defi nitions of the process model. In this way, the task of identifying appropriate services and the decomposition of a service

[10]In [13] this kind of description is denoted as a functional architecture of the configured product.

request into several separate requests can be performed by domain independent mediators. Due to the lack of these standards, this mediating functionality is in the current state of the CAWICOMS Workbench performed by application logic integrated into the configuration systems. DAML-S[11] is an example for an effort underway that aims at providing such a standardized semantic markup for Web services that builds on top of DAML+OIL. In this way semantic capability descriptions are possible that will then allow to implement semantic configuration Web services.

6 Summary

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 potential future description standards for Web services.

In [1] we build on the equivalence of a consistency based and a description logic based definition of configuration. Therefore, we are capable of transforming our application domain-independent meta-model for modeling configuration knowledge [2] into the emerging standards of the Semantic Web initiative, such as DAML+OIL, that set the stage for semantic capability descriptions that will be exploited by the next generation of Web services.

References

[1] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., Zanker, M.: Configuration Knowledge Representations for Semantic Web Applications. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM) (to appear) (2003)

[2] Felfernig, A., Friedrich, G., 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** (2000) 449–469

[3] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Meyer, M., Petrone, G., Schäfer, R., Zanker, M.: A Framework for Rapid Development of Advanced Web-based Configurator Applications. In: Proceedings of the 15th European Conference on Artificial Intelligence - Prestigious Applications of Intelligent Systems (PAIS 2002), Lyon, France (2002)

[4] Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M., Flett, A.: Product Data Integration in B2B E-Commerce. IEEE Intelligent Systems **16** (2001) 54–59

[5] Berners-Lee, T.: Weaving the Web. Harper Business (2000)

[6] Hendler, J.: Agents and the Semantic Web. IEEE Intelligent Systems **16** (2001) 30–37

[7] Fensel, D., VanHarmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems **16** (2001) 38–45

[8] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)

[11] See http://www.daml.org/services for reference.

[9] Booch, G.: Object-Oriented Analysis and Design with Applications. Addison-Wesley Object Technology Series (1994)

[10] Jacobson, I., Christerson, M., Övergaard, G.: Object-oriented Software Engineering - A Use-Case Driven Approach. Addison- Wesley (1992)

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall International Editions, New Jersey, USA (1991)

[12] Chandrasekaran, B., Josephson, J., Benjamins, R.: What Are Ontologies, and Why do we Need Them? IEEE Intelligent Systems **14** (1999) 20–26

[13] Mittal, S., Frayman, F.: Towards a Generic Model of Configuration Tasks. In: Proceedings 11^{th} International Joint Conference on Artificial Intelligence (IJCAI), Detroit, MI (1989) 1395–1401

[14] Heinrich, M., J¨ungst, E.: A resource-based paradigm for the configuring of technical systems from modular components. In: Proceedings of the 7^{th} IEEE Conference on AI applciations (CAIA). (1991) 257–264

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

[16] Soininen, T., Tiihonen, J., M¨annist¨o, T., Sulonen, R.: Towards a General Ontology of Configuration. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM), Special Issue on Configuration Design **12** (1998) 357–372

[17] Borgida, A.: On the relative expressive power of description logics and predicate calculus. Artificial Intelligence **82** (1996) 353–367

[18] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., Zanker, M.: A Joint Foundation for Configuration in the Semantic Web. 15^{th} European Conference on Artificial - Configuration Workshop (2002)

[19] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-Based Diagnosis of Configuration Knowledge Bases. In: Proceedings of the 14^{th} European Conference on Artificial Intelligence (ECAI), Berlin, Germany (2000) 146–150

[20] Artale, A., Franconi, E., Guarino, N., Pazzi, L.: Part-Whole Relations in Object-Centered Systems: An Overview. Data & Knowledge Engineering **20** (1996) 347–383

[21] Sattler, U.: Description Logics for the Representation of Aggregated Objects. In: Proceedings of the 14^{th} European Conference on Artificial Intelligence (ECAI). (2000) 239–243

[22] Warmer, J., Kleppe, A.: The Object Constraint Language - Precise Modeling with UML. Addison Wesley Object Technology Series (1999)

[23] Felfernig, A., Friedrich, G., Jannach, D.: 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, Chicago, IL (2000) 284–293

[24] McIlraith, S., Son, T., Zeng, H.: Mobilizing the Semantic Web with DAML-Enabled Web Services. In: 17^{th} International Joint Conference on Artificial Intelligence (IJCAI) - Workshop on E-Business and the Intelligent Web. (2001) 29–39

[25] Junker, U.: QuickXPlain: Conflct Detection for Arbitrary Constraint Propagation Algorithms. In: 17^{th} International Joint Conference on Artificial Intelligence (IJCAI) - Workshop on Modelling and Solving problems with constraint, Seattle, WA (2001)