*Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (2003), *17*, 31–50. Printed in the USA.

DOI: 10.1017/S0890060403171041

# Configuration knowledge representations for Semantic Web applications

ALEXANDER FELFERNIG,[1] GERHARD FRIEDRICH,[1] DIETMAR JANNACH,[1]
MARKUS STUMPTNER,[2] AND MARKUS ZANKER[1]

[1]Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik, Klagenfurt, Austria
[2]University of South Australia, Advanced Computing Research Centre, Adelaide, Australia

(RECEIVED March 15, 2002; ACCEPTED October 13, 2002)

**Abstract**

Today's economy exhibits a growing trend toward highly specialized solution providers cooperatively offering configurable products and services to their customers. This paradigm shift requires the extension of current standalone configuration technology with capabilities of knowledge sharing and distributed problem solving. In this context a standardized configuration knowledge representation language with formal semantics is needed in order to support knowledge interchange between different configuration environments. Languages such as Ontology Inference Layer (OIL) and DARPA Agent Markup Language (DAML+OIL) are based on such formal semantics (description logic) and are very popular for knowledge representation in the Semantic Web. In this paper we analyze the applicability of those languages with respect to configuration knowledge representation and discuss additional demands on expressivity. For joint configuration problem solving it is necessary to agree on a common problem definition. Therefore, we give a description logic based definition of a configuration problem and show its equivalence with existing consistency-based definitions, thus joining the two major streams in knowledge-based configuration (description logics and predicate logic/constraint based configuration).

**Keywords:** Configuration; Knowledge Representation; Ontologies

## 1. INTRODUCTION

Knowledge-based configuration has a long history as a successful AI application area (e.g., Barker et al., 1989; Mittal & Frayman, 1989; Heinrich & Jüngst, 1991; Wright et al., 1993; Fleischanderl et al., 1998; Mailharro, 1998). Starting with rule-based systems such as R1/XCON (Barker et al., 1989), various higher level representation formalisms have been developed since the late 1980s to exploit the advantages of more concise representation, faster application development, higher maintainability, and more flexible reasoning. Although these representations have proven their applicability in various real-world applications, the heterogeneity of configuration knowledge representation is the major obstacle to incorporating configuration technology in E-commerce environments. The trend toward highly specialized solution providers results in a situation where different configurators of complex products and services must be integrated in order to transparently support distributed configuration problem solving. In such integration scenarios, configurators must share a clear and common understanding of the problem definition and the semantics of the exchanged knowledge. Consequently, it is necessary to agree on the definition of a configuration problem and its solution. Of the two current main streams in representing and solving configuration problems, the first approach is based on predicate logic or various simplified variants thereof, specifically constraint-based systems (including their dynamic and generative variants, e.g., Mittal & Falkenhainer, 1990; Fleischanderl et al., 1998; Mailharro, 1998) and resource balancing methods (e.g., Heinrich & Jüngst, 1991). The second approach uses description logics as knowledge representation and reasoning mechanism (e.g., Wright et al., 1993; McGuinness & Wright, 1998). Clearly, an integration of these approaches is a major milestone for the integration of configuration systems.

Reprint requests to: Alexander Felfernig, Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria. E-mail: alexander.felfernig@uni-klu.ac.at

In order to map the predicate logic based representations into description logic based representations and vice versa, the definition of a common view of a configuration task is needed. A solution for the exchange of knowledge is the provision of a standardized configuration knowledge representation language, which is based on state of the art web technologies that allow easy integration of existing proprietary configuration environments if this common view can be obtained. Languages such as Ontology Inference Layer (OIL; Fensel et al., 2001*b*) or DARPA Agent Markup Language (DAML+OIL; VanHarmelen et al., 2001), which were developed in the context of the Semantic Web (Berners–Lee, 2000), are intended for designing and sharing ontologies. These languages are strongly influenced by description logics and therefore possess clear declarative semantics, an important precondition for the exchange of knowledge. A commonly accepted problem definition with formal semantics on this basis will offer a well-defined interface between configurator implementations and allow a joint provisioning of configuration services, a major step toward the integration goal identified above. Proprietary configuration systems can then be independently implemented following different approaches and still be able to interoperate. We only require that cooperating configurators deliver valid solutions with respect to the common definition of the problem and its solution.

In this paper we give a description logics based definition of a configuration task and show the equivalence of this definition with a consistency-based definition given in Felfernig et al. (2000*b*). The major result of this equivalence is that configuration tasks defined in terms of description logics and predicate logic can be easily transformed into each other and can consequently be represented in ontology representation languages such as OIL or DAML+OIL. Using concepts of OIL,[1] we present the constituting elements of a configuration knowledge representation language by formalizing modeling concepts of *de facto* standard configuration ontologies (Soininen et al., 1998; Felfernig et al., 2000*a*) employed in industrial applications.[2] In addition, we point out extensions that the ontology representation languages need for full fledged configuration knowledge representation. Note that the goal is not simply the representation of configuration knowledge in description logics—this is a subject that has been covered elsewhere as well (Klein et al., 1994; Schröder et al., 1996; Weida, 1996; McGuinness & Wright, 1998). Instead our goal is to compare the requirements of a general configuration ontology with the the logics chosen for the Semantic Web and to describe the specific extensions required for the purpose of communicating configuration knowledge between state of the art configurators via OIL and DAML+OIL.

The rest of the paper is organized as follows. In Section 2 we introduce an example that provides an overview of representative modeling concepts required for building configuration knowledge bases. In Section 3 we give a description logic based definition of a configuration task and show its equivalence to the consistency-based definition given in Felfernig et al. (2000*b*). In Section 4 we describe an OIL-based as well as a corresponding predicate logic based formalization of the modeling concepts presented in Section 2. In Section 5 we summarize the results and analyze the expressiveness of available Semantic Web ontology representation languages with respect to their capability for configuration knowledge representation. In Section 6 the CAWICOMS[3] (Ardissono et al., 2001) environment is presented, which supports personalized distributed configuration problem solving based on the knowledge representation concepts discussed in this paper. Section 7 discusses related work.

## 2. EXAMPLE CONFIGURATION KNOWLEDGE BASE

The Unified Modeling Language (UML; Rumbaugh et al., 1998) is the result of an integration of the object-oriented approaches of Rumbaugh et al. (1991), Jacobson et al. (1992), and Booch (1994), 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 extension of the basic metamodel 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*. UML profiles can be compared with ontologies discussed in the artificial intelligence literature (e.g., Chandrasekaran et al., 1999, defines an ontology as a theory about the sorts of objects, properties of objects, and relationships between objects that are possible in a specified domain of knowledge).

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, that is, 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 that are related to technical restrictions, economic factors, and restrictions according to the production process. The concepts used stem from connection-based (Mittal & Frayman, 1989), resource-based (Heinrich & Jüngst, 1991), and

[1] For presentation purposes we employ OIL text throughout the paper, but this representation can be easily transformed into a corresponding DAML+OIL representation.

[2] Note that these differ from the configuration ontology that was described for demonstration purposes in Gruber et al. (1996).

[3] CAWICOMS is the acronym for Customer-Adaptive Web Interface for the Configuration of Products and Services with Multiple Suppliers.

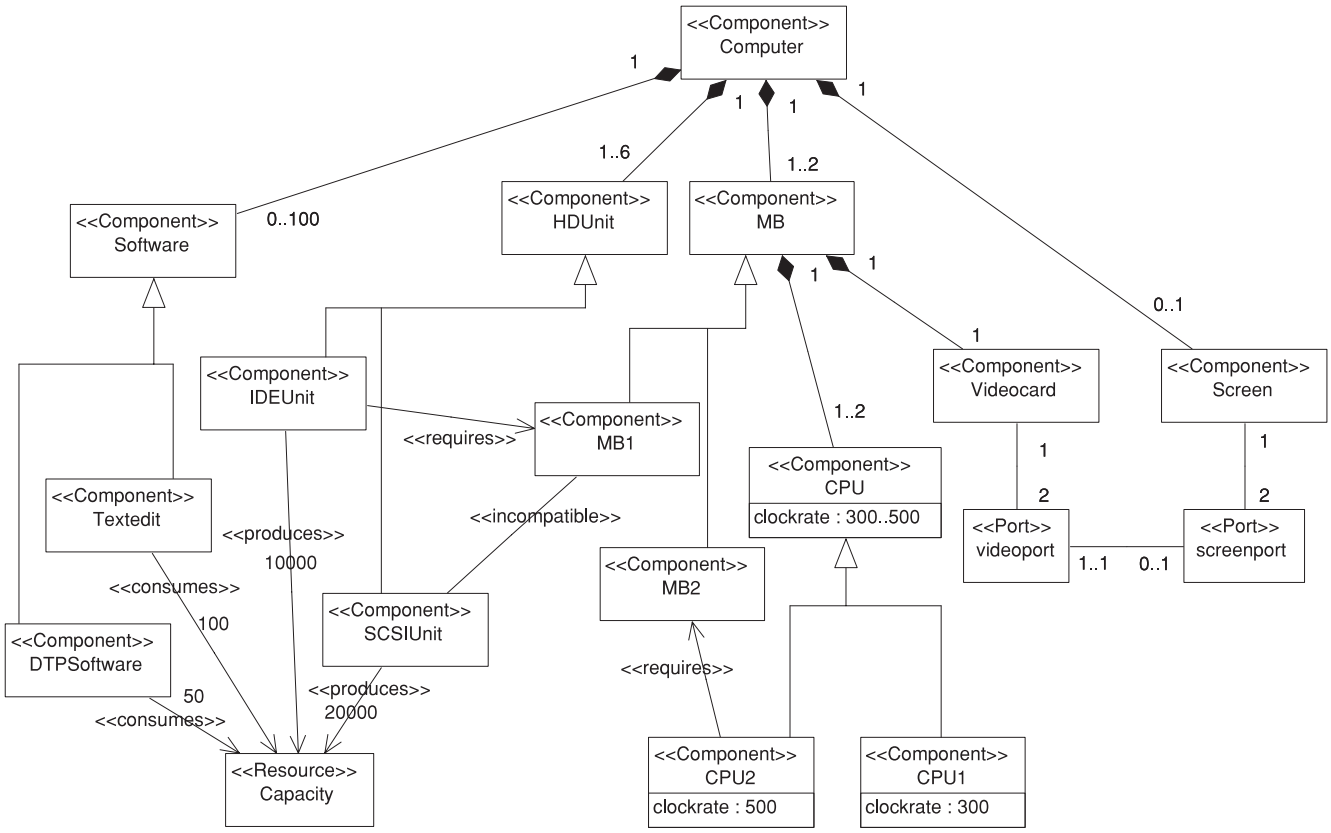


**Fig. 1.** An example of a configuration model.

structure-based (Stumptner, 1997) configuration approaches. These configuration domain-specific concepts represent a basic set useful for building configuration knowledge bases and mainly correspond to those defined in *de facto* standard configuration ontologies (Soininen et al., 1998; Felfernig et al., 2000*a*):

- *Component types*: Component types represent the basic building blocks of which a final product can be built and they are characterized by attributes (e.g., in Fig. 1 the component type CPU is characterized by the attribute *clockrate*).
- *Generalization hierarchies*: Component types with a similar structure are arranged in a generalization hierarchy (e.g., in Fig. 1 the component type 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 (MB)].
- *Incompatibilities and requirements*: Some types of components must not be used together within the same configuration because they are incompatible (e.g., a SCSIUnit is incompatible with a 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 a MB1).
- *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*). Dependencies are introduced for representing the producer/consumer relationships between different component types.
- *Port connections*: In addition to the amount of components and their type specifications, the product topology (i.e., exactly how the components are interconnected) is often of interest as well in the final configuration. The concept of a port is used for this purpose (e.g., see the connection between *Videocard* and *Screen* which is based on the ports *videoport* and *screenport*). Note that in our example we do not apply generalization hierarchies for ports and resources. Such a generalization hierarchy could be defined depending on the application domain and the capabilities of the underlying reasoning system. Our definitions

are general enough to allow these generalization hierarchies.[4]

In the following we give a description logics based definition of a configuration problem and show its equivalence with the consistency-based definition given in Felfernig et al. (2000*b*). Furthermore, we describe OIL[5] based and corresponding predicate logic based representations of the modeling concepts presented in this section.

## 3. DEFINING CONFIGURATION TASKS IN DESCRIPTION LOGIC AND PREDICATE LOGIC

### 3.1. Description logic based configuration task

For the description of a configuration task we employ a description logic language (e.g., OIL) starting from a schema $S = (\mathcal{CN}, \mathcal{RN}, \mathcal{IN})$ of disjoint sets of names for concepts, roles, and individuals (Borgida, 1996). Concepts can be seen as unary predicates defining classes (component types). Roles are used to express relationships between different elements of a domain. Finally, individuals are specific named elements of the domain.

DEFINITION 1 (configuration problem in description logic) In general we assume a configuration problem is described by a triple ($\text{DD}_{\text{DL}}$, $\text{SRS}_{\text{DL}}$, $\text{CLANG}_{\text{DL}}$), where DL is description logic, $\text{DD}_{\text{DL}}$ represents the domain description of the configurable product, and $\text{SRS}_{\text{DL}}$ specifies the particular system requirements defining an individual configuration problem instance. $\text{CLANG}_{\text{DL}}$ comprises a set of concepts $C_{\text{config}} \subseteq \mathcal{CN}$ and a set of roles $R_{\text{config}} \subseteq \mathcal{RN}$ that serve as a configuration language for the description of actual configurations (solutions). ■

One can think of additional restrictions on $\text{SRS}_{\text{DL}}$, $\text{DD}_{\text{DL}}$, for example, that $\text{SRS}_{\text{DL}}$ may only contain concepts and roles that also occur in $\text{DD}_{\text{DL}}$. However, we leave this decision to designers of domain-specific solutions.

In the following we will describe solutions to configuration problems based on the interpretation of concepts and roles. In order to make sure that roles in $\text{CLANG}_{\text{DL}}$ describe relationships only between concepts of $\text{CLANG}_{\text{DL}}$, we require that roles in $\text{CLANG}_{\text{DL}}$ are defined over the domains given in $C_{\text{config}}$; that is, we add for each $R_i \in R_{\text{config}}$ the role descriptions $\text{range}(R_i) = \text{CDom}$ and $\text{dom}(R_i) = \text{CDom}$ for $\text{CDom} \doteq \sqcup_{C_i \in C_{\text{config}}} C_i$ to $\text{DD}_{\text{DL}}$, if such descriptions are not subsumed by other descriptions already contained in the knowledge base. Note that $\text{DD}_{\text{DL}}$ may contain auxiliary concepts or roles that are not actually in $\text{CLANG}_{\text{DL}}$.

[4] Note that UML interfaces can be used as an alternative representation for ports, although in our configuration profile we decided to use specialized classes for this purpose.

[5] In the following we assume that the reader is familiar with the concepts of OIL. See Fensel et al. (2001*b*) for an introductory text.

EXAMPLE 1: In this example we use a part of our computer ontology (see Fig. 1) that comprises CPUs and MBs as component types. On each MB at least one, but at most two, CPUs are mounted (constraint $c_1$ in $\text{DD}_{\text{DL}}$). A CPU must always be mounted on a MB (constraint $c_2$ in $\text{DD}_{\text{DL}}$). A CPU of type CPU2 must be mounted on a MB of type MB2 (constraint $c_3$ in $\text{DD}_{\text{DL}}$).

The domain description $\text{DD}_{\text{DL}}$ is defined as follows. Note that *subclass-of* and *disjoint* are used to express the completeness and disjointness of the generalization hierarchy where each individual must be instantiated to exactly one of the leaf nodes. Note also that to avoid introducing a separate operator we use subclass-of in reverse in one case, when writing CPU subclass-of (CPU1 or CPU2) to specify that the coverage of the subclasses is complete. These requirements together with the assumption that there is a fixed set of component types corresponds to assumptions in other description logic based papers such as Klein et al. (1994) and Weida (1996). We will discuss the implications in Section 5.

$\text{DD}_{\text{DL}} = \{$
class-def MB subclass-of (MB1 or MB2)
  slot-constraint cpu-of-mb min-cardinality 1 CPU
  slot-constraint cpu-of-mb max-cardinality 2 CPU. $[c_1]$
class-def MB1 subclass-of MB.
class-def MB2 subclass-of MB.
disjoint MB1 MB2.
class-def CPU subclass-of (CPU1 or CPU2)
  slot-constraint mb-of-cpu cardinality 1 MB. $[c_2]$
class-def CPU1 subclass-of CPU.
class-def CPU2 subclass-of CPU
  slot-constraint mb-of-cpu cardinality 1 MB2. $[c_3]$
disjoint CPU1 CPU2.
disjoint CPU MB.
slot-def mb-of-cpu
  inverse cpu-of-mb domain CPU range MB.
slot-def cpu-of-mb
  inverse mb-of-cpu domain MB range CPU.$\}$ ■

The customer requirement that there must be (at least) "two CPUs of type CPU1 and one CPU of type CPU2" is expressed by $\text{SRS}_{\text{DL}} = \{$(instance-of *c*1 CPU1), (instance-of *c*2 CPU1), (instance-of *c*3 CPU2)$\}$.

The configuration language $\text{CLANG}_{\text{DL}}$ is defined by $C_{\text{config}} = \{$CPU1, CPU2, MB1, MB2$\}$ and $R_{\text{config}} = \{$mb-of-cpu$\}$.

In our example we do not include the concepts CPU and MB and the role cpu-of-mb in $\text{CLANG}_{\text{DL}}$ because we are only interested in the leaf concepts of a generalization hierarchy and specific relationships (e.g., to manufacture the final system we only need to know the most specific type

for each component and its connections). The semantics of description terms are usually given denotationally using an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a domain (nonempty universe) of values and $(\cdot)^{\mathcal{I}}$ is a mapping from concept descriptions to subsets of the domain and from role descriptions to sets of 2-tuples over the domain. The mapping also associates with every individual name in $\mathcal{IN}$ some distinct value in $\Delta^{\mathcal{I}}$. The reason for this distinctness is the unique name assumption (UNA) we employ in our formalism. We require the UNA for concepts and roles that describe configurations in order to make sure that different identifiers for individuals (e.g., modules of a system) refer to different individuals. The UNA can be lifted, if necessary, for those concepts and roles that are not used to describe configurations. In the following we give a description logic based definition of a valid configuration and show its equivalence with consistency-based definitions given in the literature (Felfernig et al., 2000*b*). This definition serves as a joint foundation of configuration knowledge representation in the Semantic Web.

We use the extensions of concepts and roles that correspond to a logical model to specify valid configurations. Note that, depending on the domain, only specific concepts and roles are considered relevant for describing configurations (i.e., those roles and concepts defined in $\text{CLANG}_{\text{DL}}$).

Our definition of a valid configuration does not make use of extraneous requirements such as minimality. Following our definitions, the domain description joined with the system requirements specification (describing the expectations of a customer) must enforce configuration solutions (described by the configuration language) such that these solutions deliver services which match the expectations of the customer. In practice (Fleischanderl et al., 1998) we have to allow solutions that may even include components that are unnecessary with respect to the current system requirements because some of these components might be needed in later extensions (reconfigurations). Accepting solutions that are satisfactory with respect to costs or to the number of components used is common practice in real-world applications. Although the optimality of such solutions is not shown, these solutions are appreciated because they are as good as or even better than good quality solutions provided by human experts.

DEFINITION 2 (valid configuration in description logic). Given a configuration problem $C = (\text{DD}_{\text{DL}}, \text{SRS}_{\text{DL}}, \text{CLANG}_{\text{DL}})$, let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$ be a model of $\text{DD}_{\text{DL}} \cup \text{SRS}_{\text{DL}}$. $\text{CLANG}_{\text{DL}} = C_{\text{config}} \cup R_{\text{config}}$ is the configuration language of the problem. Let COMPS be a set of tuples $\langle C_i, \text{INDIVS}_{C_i} \rangle$ for every $C_i \in C_{\text{config}}$, where $\text{INDIVS}_{C_i} = \{ci_1, \ldots, ci_{n_i}\} = C_i^{\mathcal{I}}$ is the set of individuals of concept $C_i$ in $\mathcal{I}$. These individuals identify components in an actual configuration. Let $\text{TUPLES}_{R_j} = \{\langle rj_1, sj_1 \rangle, \ldots, \langle rj_{m_j}, sj_{m_j} \rangle\} = R_j^{\mathcal{I}}$ be the set of tuples of role $R_j$, then we define $\text{ROLES} = \{\langle R_j, \text{TUPLES}_{R_j} \rangle | R_j \in R_{\text{config}}\}$.

We call $\text{CONF}_{\text{DL}} = \text{COMPS} \cup \text{ROLES}$ a valid configuration for $C$. ■

EXAMPLE 2. A valid configuration for our example configuration problem is $\text{CONF}_{\text{DL}} = \{\langle \text{CPU1}, \{c1, c2\} \rangle, \langle \text{CPU2}, \{c3\} \rangle, \langle \text{MB1}, \{m1\} \rangle, \langle \text{MB2}, \{m2\} \rangle, \langle \text{mb-of-cpu}, \{\langle m1, c1 \rangle, \langle m1, c2 \rangle \langle m2, c3 \rangle\}\rangle\}$. ■

We also have to describe component parameter settings in addition to components and their connections. Using description logics, such parameter settings of components are modeled by special functional roles (also called *features*) expressing the relation between the component and the data value assigned to a particular attribute. Therefore, component structure and parameter settings can be treated in a uniform manner except that the parameter values come from some data value domain $\text{dom}(D)$ (Sattler, 2000), which is disjunct from the individuals in COMPS.

In addition to the definition of a valid configuration given above, we can provide an equivalent characterization based on checking the consistency of a set of axioms.

REMARK 1. Let $\text{DD}_{\text{DL}}$, $\text{SRS}_{\text{DL}}$, $\text{CLANG}_{\text{DL}} = C_{\text{config}} \cup R_{\text{config}}$ be a configuration problem, and $\text{CONF}_{\text{DL}} = \text{COMPS} \cup \text{ROLES}$ be a description of a configuration.

The concepts $C_i$ are defined by the component axioms

$$\text{AX}_{\text{COMPS}} = \{C_i \doteq \text{one-of}[ci_1, \ldots, ci_{n_i}] | \langle C_i, \text{INDIVS}_{C_i} \rangle \in \text{COMPS} \quad \text{where} \quad \text{INDIVS}_{C_i} = \{ci_1, \ldots, ci_{n_i}\}\}.$$

The roles $R_j$ are defined by the role axioms

$$\text{AX}_{\text{ROLES}} = \left\{ R_j \doteq \bigsqcup_{\langle rj, sj \rangle \in \text{TUPLES}_{R_j}} \text{product}[\text{one-of}[rj], \text{one-of}[sj]] | \langle R_j, \text{TUPLES}_{R_j} \rangle \in \text{ROLES} \quad \text{where} \quad \text{TUPLES}_{R_j} = \{\langle rj_1, sj_1 \rangle, \ldots, \langle rj_{m_j}, sj_{m_j} \rangle\} \right\}.$$

$\text{CONF}_{\text{DL}}$ is a valid configuration iff

$$\text{DD}_{\text{DL}} \cup \text{SRS}_{\text{DL}} \cup \text{AX}_{\text{COMPS}} \cup \text{AX}_{\text{ROLES}}$$

is satisfiable. ■

Note that we use the above notation for defining the complete extension of roles in order to apply the translation function from description logics to predicate logic defined in Borgida (1996). An alternative to the *product* constructor would be a constructor that (similar to the *one-of* constructor for concepts) permits the enumeration of permissible entries for a role. However, computation of the product requires only constant effort because we only apply *product* to singleton arguments.

EXAMPLE 3. In order to verify that a given configuration is valid with respect to our example configuration problem (defined by $DD_{DL}$ and $SRS_{DL}$), we need to add the axioms CPU1 $\doteq$ one-of$[c1, c2]$, CPU2 $\doteq$ one-of$[c3]$, MB1 $\doteq$ one-of$[m1]$, MB2 $\doteq$ one-of$[m2]$, mb-of-cpu $\doteq$ product[one-of$[m1]$, one-of$[c1]] \sqcup$ product [one-of$[m1]$, one-of$[c2]] \sqcup$ product[one-of$[m2]$, one-of$[c3]]$. ■

We now give a consistency-based definition of a configuration problem using predicate logic[6] (corresponding to the definition given in Felfernig et al., 2000*b*) and show the equivalence with the description logic based definition given before.

### 3.2. Predicate logic based configuration task

DEFINITION 3 (configuration problem in predicate logic). In general we assume a configuration problem is described by a triple ($DD_{LOG}$, $SRS_{LOG}$, $CLANG_{LOG}$) where $DD_{LOG}$ and $SRS_{LOG}$ are sets of logical sentences and $CLANG_{LOG}$ is a set of predicate symbols. $DD_{LOG}$ represents the domain description and $SRS_{LOG}$ specifies the particular system requirements. A configuration $CONF_{LOG}$ is described by a set of positive ground literals whose predicate symbols are in the set of $CLANG_{LOG}$. ■

EXAMPLE 4. For our example, $DD_{LOG}$ can be expressed by using monadic and dyadic predicates and numerical quantifiers, that is:

$DD_{LOG} = \{$

$\forall X: \mathrm{MB}(X) \leftrightarrow \mathrm{MB1}(X) \vee \mathrm{MB2}(X).$

$\forall X: \neg\mathrm{MB1}(X) \vee \neg\mathrm{MB2}(X).$

$\forall X: \mathrm{MB}(X) \rightarrow \exists_1^2 Y: \text{cpu-of-mb}(Y, X).$ $[c1]$

$\forall X: \mathrm{CPU}(X) \leftrightarrow \mathrm{CPU1}(X) \vee \mathrm{CPU2}(X).$

$\forall X: \neg\mathrm{CPU1}(X) \vee \neg\mathrm{CPU2}(X).$

$\forall X: \mathrm{CPU}(X) \rightarrow \exists_1^1 Y: \text{mb-of-cpu}(Y, X).$ $[\mathrm{c2}]$

$\forall X: \mathrm{CPU2}(X) \rightarrow \exists_1^1 Y: \text{mb-of-cpu}(Y, X) \wedge \mathrm{MB2}(Y).$ $[\mathrm{c3}]$

$\forall X: \neg\mathrm{MB}(X) \vee \neg\mathrm{CPU}(X).$

$\forall X, Y: \text{mb-of-cpu}(X, Y) \leftrightarrow \text{cpu-of-mb}(Y, X).$

$\forall X, Y: \text{mb-of-cpu}(X, Y) \rightarrow \mathrm{MB}(X) \wedge \mathrm{CPU}(Y)\}.$

$SRS_{LOG} = \{\mathrm{CPU1}(c1).\ \mathrm{CPU1}(c2).\ \mathrm{CPU2}(c3).\}.$

$CLANG_{LOG} = \{\text{CPU1, CPU2, MB1, MB2, mb-of-cpu}\}.$ ■

[6] We employ a notation where variables are all quantified if not explicitly mentioned.

DEFINITION 4 (consistent configuration in predicate logic). Given a configuration problem ($DD_{LOG}$, $SRS_{LOG}$, $CLANG_{LOG}$), a configuration $CONF_{LOG}$ is consistent iff $DD_{LOG} \cup SRS_{LOG} \cup CONF_{LOG}$ is satisfiable. ■

This definition allows determining the consistency of partial configurations but does not guarantee the completeness of configurations (Felfernig et al., 2000*b*). It is necessary that a configuration explicitly includes all needed components (and their connections and attribute values) in order to assemble a correctly functioning system. We need to introduce an explicit formula for each predicate symbol in $CLANG_{LOG}$ to enforce its completeness property. In order to stay within first order logic, we model the property by first order formulae. For our example we have to add the completeness axiom $\forall X: \mathrm{CPU1}(X) \Rightarrow (\bigvee_{Z \in CONF_{LOG}} \mathrm{CPU1}(X) = Z)$ for the predicate CPU1 and similar axioms for CPU2 and mb-of-cpu. Note that $\bigvee_{Z \in CONF_{LOG}} \mathrm{CPU1}(X) = Z$ serves as a macro, which is expanded into a set of formulas by substituting the elements of $CONF_{LOG}$ for $Z$. Also note that if there are no parts of a particular type, the consequent of this rule degenerates to the empty clause (preventing spurious parts of that type from appearing in the configuration). We refer to $CONF_{LOG}$ extended by completeness axioms as $\widehat{CONF}_{LOG}$.

Note that the empty configuration can be a valid configuration. This is in keeping with the standard approach to specifying configuration problems, where it is a problem-dependent explicit specification expressed through key components (Mittal & Frayman, 1989; Heinrich & Jüngst, 1991; Fleischanderl et al., 1998) or some sort of functionality (which can only be satisfied by including components in the configuration, e.g., because of the existence of resource constraints) that prevents empty configurations from being consistent.

EXAMPLE 5. $CONF_{LOG}$ = {CPU1($c1$). CPU1($c2$). CPU2($c3$). MB1($m1$). MB2($m2$). mb-of-cpu($m1, c1$). mb-of-cpu($m1, c2$). mb-of-cpu($m2, c3$).}

The completeness axiom for CPU1 is

$$\forall X: \mathrm{CPU1}(X) \Rightarrow \mathrm{CPU1}(X) = \mathrm{CPU1}(c1) \vee \mathrm{CPU1}(X) = \mathrm{CPU1}(c2),$$

where unsatisfiable literals are deleted. ■

DEFINITION 5 (valid configuration in predicate logic). Let ($DD_{LOG}$, $SRS_{LOG}$, $CLANG_{LOG}$) be a configuration problem. A configuration $CONF_{LOG}$ is valid iff $DD_{LOG} \cup SRS_{LOG} \cup \widehat{CONF}_{LOG}$ is satisfiable. ■

Note that $CONF_{LOG}$ in Example 5 is a valid configuration.

### 3.3. Equivalence

In order to show the equivalence of valid configurations for description logic and predicate logic, we apply a translation

function $\mathcal{T}\langle\cdot\rangle$ that maps description logics to predicate logic, that is, axioms to formulas with no free variables, concepts to formulas with one free variable, and roles to formulas with two free variables.

Borgida (1996, Theorem 1 and subsequent corollaries) provides such a translation function $\mathcal{T}\langle\cdot\rangle$ such that concepts, roles, terms, and axioms of a description language ($\mathcal{DL}$) without transitive closure[7] are translated into equivalent formulas in the first-order predicate logic $\ddot{\mathcal{L}}^3_{\text{CNT}}$. This language $\ddot{\mathcal{L}}^3_{\text{CNT}}$ allows only monadic and dyadic predicates, counting quantifiers and subformulas with at most three free variables. The UNA is assumed from the outset. For a concept $C$ in $\mathcal{CN}$ $\mathcal{T}^X\langle C\rangle = C(X)$, for a role $r$ in $\mathcal{RN}$ $\mathcal{T}^{X,Y}\langle r\rangle$ produces a predicate $r(X,Y)$. Following this approach, $C \sqsubseteq D$ can be translated as $\forall X : \mathcal{T}^X\langle C\rangle \rightarrow \mathcal{T}^X\langle D\rangle$, and $[C, D]$ can be translated as $\mathcal{T}^X\langle C\rangle \wedge \mathcal{T}^X\langle D\rangle$, or $[C, D]$ can be translated as $\mathcal{T}^X\langle C\rangle \vee \mathcal{T}^X\langle D\rangle$, one-of $[b_1, b_2, \ldots, b_m]$ can be translated as $X = b_1 \vee \mathrm{X} = b_2 \vee \cdots \vee X = b_m$, and product$[C, D]$ can be translated as $\mathcal{T}^X\langle C\rangle \wedge \mathcal{T}^Y\langle D\rangle$.

Similar to the above-mentioned translation function (Borgida, 1996, Theorem 2 and subsequent corollaries) defines a translation function that maps sentences of $\ddot{\mathcal{L}}^3_{\text{CNT}}$ into $\mathcal{DL}$. Given function $\mathcal{T}\langle\cdot\rangle$, $\text{DD}_{\text{LOG}} \cup \text{SRS}_{\text{LOG}} = \mathcal{T}\langle\text{DD}_{\text{DL}} \cup \text{SRS}_{\text{DL}}\rangle$ holds for all $\text{DD}_{\text{DL}}$ and $\text{SRS}_{\text{DL}}$, and $\text{DD}_{\text{LOG}} \cup \text{SRS}_{\text{LOG}}$ is satisfiable iff $\text{DD}_{\text{DL}} \cup \text{SRS}_{\text{DL}}$ is satisfiable.

REMARK 2 (equivalence of configuration problems). Let $\text{CLANG}_{\text{LOG}} = \text{CLANG}_{\text{DL}}$ where each concept is interpreted as monadic and each role is interpreted as dyadic predicate. $\text{DD}_{\text{DL}}$ and $\text{SRS}_{\text{DL}}$ are sets of sentences in the description logic language $\mathcal{DL}$ without transitive closure. $\text{DD}_{\text{LOG}}$ and $\text{SRS}_{\text{LOG}}$ are sets of sentences in $\ddot{\mathcal{L}}^3_{\text{CNT}}$. $\text{CONF}_{\text{LOG}}$ describes the actual configuration by two sets of facts $\text{CONF}_{\text{LOG}} = \text{COMP-facts} \cup \text{ROLE-facts}$. The construction of $\text{CONF}_{\text{LOG}}$ is based on $\text{CONF}_{\text{DL}} = \text{COMPS} \cup \text{ROLES}$ where COMP-facts $= \{C_i(ci) | ci \in \text{INDIVS}_{C_i}, C_i \in C_{\text{config}}\}$ and ROLE-facts $= \{R_j(rj, sj) | \langle rj, sj\rangle \in \text{TUPLES}_{R_j}, R_j \in R_{\text{config}}\}$. $\text{DD}_{\text{LOG}} = \mathcal{T}\langle\text{DD}_{\text{DL}}\rangle$ and $\text{SRS}_{\text{LOG}} = \mathcal{T}\langle\text{SRS}_{\text{DL}}\rangle$.

$\text{CONF}_{\text{LOG}}$ is a valid configuration for ($\text{DD}_{\text{LOG}}$,$\text{SRS}_{\text{LOG}}$, $\text{CLANG}_{\text{LOG}}$) iff $\text{CONF}_{\text{DL}}$ is a valid configuration for ($\text{DD}_{\text{DL}}$,$\text{SRS}_{\text{DL}}$,$\text{CLANG}_{\text{DL}}$). ■

Remark 2 follows from Remark 1 and the equivalence property of the translation function. Note that $\text{CLANG}_{\text{LOG}}$ is restricted to exactly the monadic and dyadic predicates that correspond to the concept and role definitions of $\text{CLANG}_{\text{DL}}$ and no others. The equivalence result stated in the last sentence of the remark is made specifically for languages restricted in this manner (i.e., $\mathcal{DL}$ without transitive closure and $\ddot{\mathcal{L}}^3_{\text{CNT}}$), and in this case the equivalence holds. Note also that the completeness axioms correspond exactly to the translation of the axioms $\text{AX}_{\text{COMPS}}$ and $\text{AX}_{\text{ROLES}}$ by applying the translation function proposed in Borgida (1996). For example, by employing the translation function, a $C_i$ in $\text{AX}_{\text{COMPS}}$ is translated to $C_i(X) \leftrightarrow X = ci_1 \vee \cdots \vee X = ci_{ni}$ for $\text{INDIVS}_{C_i} = ci_1, \ldots, ci_{ni}$, which is equivalent to our completeness axioms (predicate logic) for $C_i$. Note that in the formulation of the completeness axioms only one direction of the implication needs to be expressed because the $C_i(c_i)$ facts about the individuals $c_i$ are asserted by definition (they are contained in $\text{CONF}_{\text{LOG}}$). The $\doteq$ corresponds to identity and $\sqcup$ corresponds to a disjunction. The equivalence for $\text{AX}_{\text{ROLES}}$ follows immediately by using the translation rules for the terms *one-of* and *product*.

Note that there is a one to one relationship between $\text{CONF}_{\text{LOG}}$ and $\text{CONF}_{\text{DL}}$. Applying Remarks 1 and 2, every valid configuration in the corresponding restricted predicate logic (i.e., $\ddot{\mathcal{L}}^3_{\text{CNT}}$) corresponds to a valid configuration in description logic. In addition, every valid configuration in description logic corresponds to a valid configuration in predicate logic. It follows that no configuration is missed using either of the two representations.

From the equivalence of configuration problems follow two important consequences. First, the two main streams in solving configuration problems based on description logics on the one hand, and predicate or propositional logic on the other hand can be easily transformed to each other. Second, because description logics without transitive closure are equally expressive to dyadic predicate logic with counting quantifiers and at most three free variables in each subformula (Borgida, 1996), it follows that the predicate logic based approach is strictly more expressive than the description logics based approach, implying that some logic constructions have to be simulated by more complex description logic constructions, and also that certain complex structural restrictions (Immerman, 1982; Cai et al., 1989; Schröder et al., 1996) are not expressible in the language directly but have to be incorporated using a more expressive assertional language.

## 4. STANDARDIZED KNOWLEDGE REPRESENTATION

UML (Rumbaugh et al., 1998) is a well-established modeling language in industrial software development processes. Felfernig et al. (2000*a*) demonstrated the applicability of UML for configuration knowledge representation. In order to bridge the gap between research and development organizations, we think it is indispensable to relate the knowledge representation concepts to UML. In the following, we employ UML as a basis for showing the representation of configuration knowledge in OIL and in predicate logic. For the modeling concepts discussed in Section 2, we present a set of rules for translating those concepts into an OIL based representation, as well as into an equivalent predicate logic based representation. The equivalence of the description logic and the predicate logic based representations is founded in the translation function $\mathcal{T}\langle\cdot\rangle$ proposed in Borgida (1996).

[7] This description language contains a comprehensive list of operators based on an analysis of a set of corresponding survey papers.

UML is based on a graphical notation; therefore, our translation starts from such a graphical description of a configuration domain. 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 and into a predicate logic based representation. Note that we do not discuss the translation of resource constraints in this section because resource constraints cannot be represented using standard predicate logic or standard description logic (i.e., OIL, DAML+OIL). Semantics for resource constraints are presented in Section 5.

RULE 1a (component types). Let $c$ be a component type in GREP, then

- $\text{DD}_{\text{DL}}$ is extended by class-def $c$.

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

- $\text{DD}_{\text{DL}}$ is extended by
  slot-def $a$.
  $c$: slot-constraint $a$ cardinality 1 $d$.
- $\text{DD}_{\text{LOG}}$ is extended by

$$\forall X : c(X) \rightarrow \exists_1^1 Y : a(X,Y).$$

$$\forall X,Y : c(X) \wedge a(X,Y) \rightarrow d(Y). \quad \blacksquare$$

RULE 1b (component types). For those component types $c_i, c_j \in \{c_1,\ldots,c_m\}$ $(c_i \neq c_j)$, which do not have any supertypes in GREP,

- $\text{DD}_{\text{DL}}$ is extended by *disjoint* $c_i, c_j$.
- $\text{DD}_{\text{LOG}}$ is extended by $\forall X : \neg c_i(X) \vee \neg c_j(X)$. $\blacksquare$

EXAMPLE 6a (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.

… $\blacksquare$

EXAMPLE 6b (component type CPU).

$$\forall X : \text{CPU}(X) \rightarrow \exists_1^1 Y : \text{clockrate}(X,Y).$$

$$\forall X,Y : \text{CPU}(X) \wedge \text{clockrate}(X,Y) \rightarrow \min(Y,300) \wedge \max(Y,500).$$

$$\forall X : \neg\text{CPU}(X) \vee \neg\text{MB}(X).$$

$$\forall X : \neg\text{MB}(X) \vee \neg\text{Screen}(X).$$

$$\ldots \quad \blacksquare$$

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 $d_1,\ldots,d_n$ be classes (component types) in GREP, where $u$ is the superclass of $d_1,\ldots,d_n$, then

- $\text{DD}_{\text{DL}}$ is extended by
  $d_1,\ldots,d_n$: subclass-of $u$.
  $u$: subclass-of ($d_1$ or $\cdots$ or $d_n$).
  $\forall d_i, d_j \in \{d_1,\ldots,d_n\}\ (d_i \neq d_j)$ : disjoint $d_i$ $d_j$.
- $\text{DD}_{\text{LOG}}$ is extended by

$$\forall X : u(X) \leftrightarrow d_1(X) \vee \cdots \vee d_n(X).$$

$$\forall X : \neg d_i(X) \vee \neg d_j(X) \quad \text{for } d_i, d_j \in \{d_1,\ldots,d_n\} \ (d_i \neq d_j). \quad \blacksquare$$

EXAMPLE 7a (CPU1, CPU2 subclasses of CPU).

CPU1: subclass-of CPU.

CPU2: subclass-of CPU.

CPU: subclass-of (CPU1 or CPU2).

disjoint CPU1 CPU2. $\blacksquare$

EXAMPLE 7b (CPU1, CPU2 subclasses of CPU).

$$\forall X : \text{CPU}(X) \leftrightarrow \text{CPU1}(X) \vee \text{CPU2}(X).$$

$$\forall X : \neg\text{CPU1}(X) \vee \neg\text{CPU2}(X).$$

$\blacksquare$

Part–whole relationships are important model properties in the configuration domain. Artale et al. (1996), Soininen et al. (1998), and Sattler (2000) pointed out that part–whole relationships have quite variable semantics depending on the 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. Note that we do not require acyclicity because particular domains such as software configuration allow cycles on the type level. In the following we discuss two facets of part–whole relationships that are widely used for configuration knowledge representation (Soininen et al., 1998) and are also provided by UML, namely *composite* and *shared* part–whole relationships. In UML the composite part–whole relationships are denoted by a black diamond and shared part–whole relationships are denoted

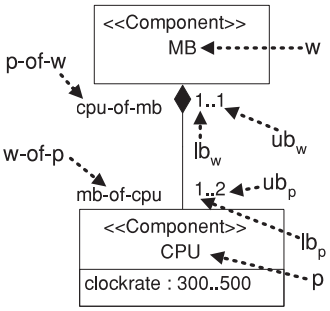


**Fig. 2.** The part–whole relationship (p, part; w, whole).

by a white diamond.[8] If a component is a *compositional part* of another component, then strong ownership is required, that is, it must be part of exactly one component. If a component is a *noncompositional* (*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 among how many aggregates a part can be shared if the aggregation is *noncomposite*. 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 $w$, $\text{ub}_p$ is the upper bound and $\text{lb}_p$ is the lower bound of the multiplicity of the part, and $\text{ub}_w$ is the upper bound and $\text{lb}_w$ is 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, that is, $p$-of-$w \sqsubseteq$ haspart, $w$-of-$p \sqsubseteq \text{Partof}_{\text{mode}}$, where $\text{Partof}_{\text{mode}} \in \{\text{partof}_{\text{composite}}, \text{partof}_{\text{shared}}\}$. A part $p$ can be either a shared part (concept $\text{part}_{\text{shared}}$) or a composite part (concept $\text{part}_{\text{composite}}$). Given a part–whole relationship between $p$ and $w$ in GREP, then

- $\text{DD}_{\text{DL}}$ is extended by
  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 $\text{lb}_w$ *w*.
  *p*: slot-constraint *w-of-p* max-cardinality $\text{ub}_w$ *w*.
  *w*: slot-constraint *p-of-w* min-cardinality $\text{lb}_p$ *p*.
  *w*: slot-constraint *p-of-w* max-cardinality $\text{ub}_p$ *p*.
- $\text{DD}_{\text{LOG}}$ is extended by

$$\forall X, Y: w\text{-of-}p(X,Y) \rightarrow \text{Partof}_{\text{mode}}(X,Y).$$
$$\forall X, Y: p\text{-of-}w(X,Y) \rightarrow \text{haspart}(X,Y).$$
$$\forall X, Y: w\text{-of-}p(X,Y) \rightarrow w(X) \wedge p(Y).$$
$$\forall X, Y: p\text{-of-}w(X,Y) \leftrightarrow w\text{-of-}p(Y,X).$$
$$\forall X: p(X) \rightarrow \text{part}_{\text{shared}}(X) \vee \text{part}_{\text{composite}}(X).$$
$$\forall X: p(X) \rightarrow \exists^{\text{ub}_w}_{\text{lb}_w} Y: w\text{-of-}p(Y,X).$$
$$\forall X: w(X) \rightarrow \exists^{\text{ub}_p}_{\text{lb}_p} Y: p\text{-of-}w(Y,X).$$ ■

[8] Note that in our Computer configuration example we only use composite part–whole relationships. As mentioned in Soininen et al. (1998), 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.

REMARK 3. The following properties have to hold for shared and composite part–whole relationships.

- Each shared part is connected to at least one whole, that is,

($\text{DD}_{\text{DL}}$) *part*$_{shared}$: slot-constraint *partof*$_{shared}$ min-cardinality 1 top.
($\text{DD}_{\text{LOG}}$) $\forall X: \text{part}_{\text{shared}}(X) \rightarrow \exists Y: \text{partof}_{\text{shared}}(Y,X)$.

- Each composite part is connected to exactly one whole, that is,

($\text{DD}_{\text{DL}}$) *part*$_{composite}$: slot-constraint *partof*$_{composite}$ min-cardinality 1 top. slot-constraint *partof*$_{composite}$ max-cardinality 1 top.
($\text{DD}_{\text{LOG}}$) $\forall X: \text{part}_{\text{composite}}(X) \rightarrow \exists^1_1 Y: \text{partof}_{\text{composite}}(Y,X)$.

- A shared part cannot be a composite part at the same time, that is,

($\text{DD}_{\text{DL}}$) *disjoint part*$_{shared}$ *part*$_{composite}$
($\text{DD}_{\text{LOG}}$) $\forall X: \neg\text{part}_{\text{shared}}(X) \vee \neg\text{part}_{\text{composite}}(X)$. ■

EXAMPLE 8a (MB partof Computer).

slot-def computer-of-mb subslot-of $\text{partof}_{\text{composite}}$
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 ($\text{part}_{\text{shared}}$ or $\text{part}_{\text{composite}}$)
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. ■

EXAMPLE 8b (MB partof Computer):

$\forall X,Y$: computer-of-mb$(X,Y) \rightarrow$ partof$_{\text{composite}}(X,Y)$.
$\forall X,Y$: mb-of-computer$(X,Y) \rightarrow$ haspart$(X,Y)$.
$\forall X,Y$: computer-of-mb$(X,Y) \rightarrow$ computer$(X) \wedge$ mb$(Y)$.
$\forall X,Y$: mb-of-computer$(X,Y) \leftrightarrow$ computer-of-mb$(Y,X)$.
$\forall X$: MB$(X) \rightarrow$ part$_{\text{shared}}(X) \vee$ part$_{\text{composite}}(X)$.
$\forall X$: MB$(X) \rightarrow \exists_1^1 Y$: computer-of-mb$(Y,X)$.
$\forall X$: Computer$(X) \rightarrow \exists_1^2 Y$: mb-of-computer$(Y,X)$. ■

### 4.1. Necessary part-of structure properties

In the following we show how the constraints contained in a 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 components involved are parts of the same subconfiguration, that is, the involved components must be connected to the same instance of the component type that represents the common root[9] for these components, meaning that they are within the same mereological context (Soininen et al., 1998). This can be simply 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 satisfied, the meaning of the imposed (graphically represented) constraints becomes ambiguous, because one component can be part of more than one substructure. 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 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.

DEFINITION 6 (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

$$\begin{aligned}\text{path}(c_1,c_n) = \langle &\text{haspart}(c_1,c_2,\text{name}_{c1}),\\ &\text{haspart}(c_2,c_3,\text{name}_{c2}),\ldots,\\ &\text{haspart}(c_{n-1},c_n,\text{name}_{cn-1})\rangle.\end{aligned}$$

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

For the translation into DD$_{\text{LOG}}$ the macro navpath$(c_1,c_n)$ is defined as

$$\exists Y_1,Y_2,\ldots,Y_{n-1},Y_n:$$
$$\text{name}_{c1}(Y_1,X) \wedge \text{name}_{c2}(Y_2,Y_1) \wedge \cdots \wedge \text{name}_{cn-1}(Y_n,Y_{n-1}) \wedge c_n(Y_n),$$

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

EXAMPLE 9a (navpath(Computer,CPU1)).

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

EXAMPLE 9b (navpath(Computer,CPU1)).

$$\exists Y_1,Y_2: \textit{mb-of-computer}(Y_1,X) \wedge \textit{cpu-of-mb}(Y_2,Y_1) \wedge \text{CPU1}(Y_2).$$ ■

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

DEFINITION 7 (nearest common root). A component type $r$ is denoted as the 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[10] of $r$ with paths path$(r',c_1)$, path$(r',c_2)$. ■

When regarding the example configuration 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 it is the nearest common root of CPU1 and IDEUnit (see Fig. 3).

### 4.2. 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 the nearest common root of $c_1$ and $c_2$, then

- DD$_{\text{DL}}$ is extended by $r$: ((not(navpath$(r,c_1)$)) or (navpath$(r,c_2)$)).
- DD$_{\text{LOG}}$ is extended by $\forall X: r(X) \rightarrow ((\text{navpath}(r,c_1)) \rightarrow (\text{navpath}(r,c_2)))$. ■

[9] In Figure 3 the component type Computer is the unique common root of IDEUnit and CPU1.

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

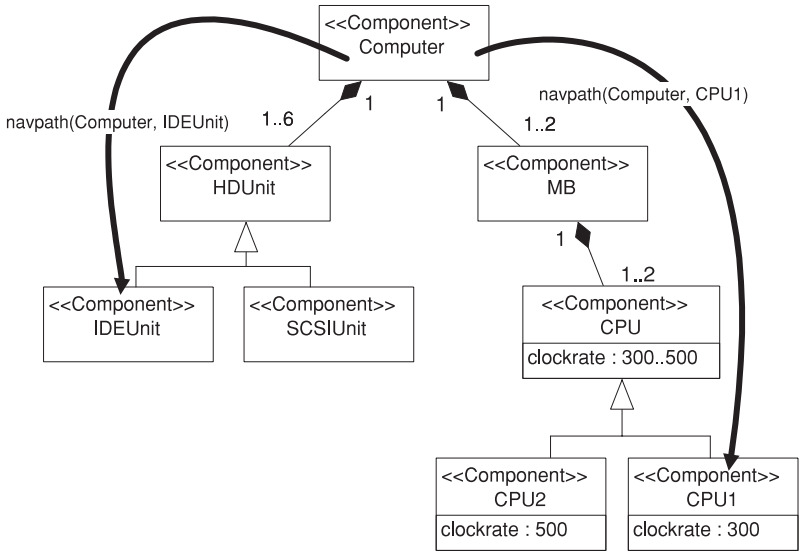


**Fig. 3.** The navigation paths from Computer to CPU1 and IDEUnit.

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 10a (*IDEUnit* requires MB1).

Computer: ((not (slot-constraint *hdunit-of-computer* has-value *IDEUnit*)) or
(slot-constraint *mb-of-computer* has-value MB1)). ■

EXAMPLE 10b (IDEUnit requires MB1).

$$\forall X : \text{Computer}(X) \rightarrow ((\exists Y_1 : \text{hdunit-of-computer}(Y_1, X) \wedge \text{IDEUnit}(Y_1)) \rightarrow (\exists Y_2 : \text{mb-of-computer}(Y_2, X) \wedge \text{MB1}(Y_2))).$$ ■

### 4.3. Incompatibility constraints

An incompatibility constraint between a set of component types $c = \{c_1, c_2, \ldots, c_n\}$ 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 $c = \{c_1, c_2, \ldots, c_n\}$ in GREP with $r$ as the nearest common root of $\{c_1, c_2, \ldots, c_n\}$, then

- $\text{DD}_{\text{DL}}$ is extended by $r$:(not((navpath$(r, c_1)$) and (navpath$(r, c_2)$) and $\cdots$ and (navpath$(r, c_n)$))).
- $\text{DD}_{\text{LOG}}$ is extended by $\forall X : r(X) \rightarrow ((\text{navpath}(r, c_1)) \wedge (\text{navpath}(r, c_2)) \wedge \cdots \wedge (\text{navpath}(r, c_n)) \rightarrow \text{false})$. ■

EXAMPLE 11a (SCSIUnit incompatible with MB1).

Computer: (not ((slot-constraint hdunit-of-computer has-value SCSIUnit) and (slot-constraint mb-of-computer has-value MB1))). ■

EXAMPLE 11b (SCSIUnit incompatible with MB1).

$$\forall X : \text{Computer}(X) \rightarrow ((\exists Y_1 : \text{hdunit-of-computer}(Y_1, X) \wedge \text{SCSIUnit}(Y_1)) \wedge (\exists Y_2 : \text{mb-of-computer}(Y_2, X) \wedge \text{MB1}(Y_2)) \rightarrow \text{false}).$$ ■

### 4.4. Resource constraints

In order to introduce resource constraints, additional expressivity requirements must be fulfilled. This issue will be discussed in Section 5.

### 4.5. Port connections

Ports in the UML configuration model (see Fig. 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.[11] The role *compnt* indicates the component concept that the port belongs to, the role *portname* determines its name, and the role *conn*

[11] 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.

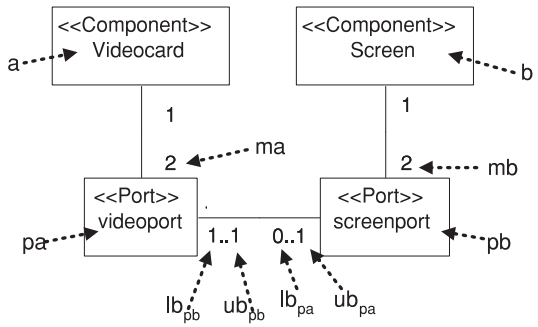


**Fig. 4.** The ports in the configuration model.

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, $\{m_a, m_b\}$ be the multiplicities of the port types with respect to $\{a, b\}$,[12] and $\{\{\text{lb}_{pa}, \text{ub}_{pa}\}, \{\text{lb}_{pb}, \text{lb}_{pb}\}\}$ be the lower and upper bounds of the multiplicities of the port types with respect to $\{pa, pb\}$, then

- $\text{DD}_{\text{DL}}$ is extended by
  class-def *pa* subclass-of *Port*.
  class-def *pb* subclass-of *Port*.
  *pa*: slot-constraint *portname* cardinality 1 (one-of $pa_1 \cdots pa_{ma}$).[13]
  *pa*: slot-constraint *conn* min-cardinality $\text{lb}_{pa}$ *pb*.
  *pa*: slot-constraint *conn* max-cardinality $\text{ub}_{pa}$ *pb*.
  *pa*: slot-constraint *conn* value-type *pb*.
  *pa*: slot-constraint *compnt* cardinality 1 *a*.
  *pb*: slot-constraint *portname* cardinality 1 (one-of $pb_1 \cdots pb_{mb}$).
  *pb*: slot-constraint *conn* min-cardinality $\text{lb}_{pb}$ *pa*.
  *pb*: slot-constraint *conn* max-cardinality $\text{ub}_{pb}$ *pa*.
  *pb*: slot-constraint *conn* value-type *pa*.
  *pb*: slot-constraint *compnt* cardinality 1 *b*.
- $\text{DD}_{\text{LOG}}$ is extended by

$$\forall X: pa(X) \rightarrow (\exists Y: \text{portname}(X,Y) \wedge (Y = pa_1 \vee \cdots \vee Y = pa_{ma})).$$
$$\forall X: pa(X) \rightarrow (\exists^{\text{ub}_{pa}}_{\text{lb}_{pa}} Y: \text{conn}(X,Y) \wedge pb(Y)).$$
$$\forall X,Y: pa(X) \wedge \text{conn}(X,Y) \rightarrow pb(Y).$$
$$\forall X: pa(X) \rightarrow (\exists^1_1 Y: \text{compnt}(X,Y) \wedge a(Y)).$$
$$\forall X: pb(X) \rightarrow (\exists Y: \text{portname}(X,Y) \wedge (Y = pb_1 \vee \ldots \vee Y = pb_{mb})).$$
$$\forall X: pb(X) \rightarrow (\exists^{\text{ub}_{pb}}_{\text{lb}_{pb}} Y: \text{conn}(X,Y) \wedge pa(Y)).$$
$$\forall X,Y: pb(X) \wedge \text{conn}(X,Y) \rightarrow pa(Y).$$
$$\forall X: pb(X) \rightarrow (\exists^1_1 Y: \text{compnt}(X,Y) \wedge b(Y)).$$ ■

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

[13] In this context $pa_i$ denotes one *pa* port.

EXAMPLE 12a (*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 $screenport_1$ $screenport_2$).

*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*. ■

EXAMPLE 12b (Videocard connected to Screen).

$$\forall X: \text{videoport}(X) \rightarrow (\exists Y: \text{portname}(X,Y) \wedge (Y = \text{videoport}_1 \vee Y = \text{videoport}_2)).$$
$$\forall X: \text{videoport}(X) \rightarrow (\exists^1_0 Y: \text{conn}(X,Y) \wedge \text{screenport}(Y)).$$
$$\forall X,Y: \text{videoport}(X) \wedge \text{conn}(X,Y) \rightarrow \text{screenport}(Y).$$
$$\forall X: \text{videoport(X)} \rightarrow (\exists^1_1 \text{Y}: \text{compnt}(X,Y) \wedge \text{Videocard}(Y)).$$
$$\forall X: \text{screenport}(X) \rightarrow (\exists Y: \text{portname}(X,Y) \wedge (Y = \text{screenport}_1 \vee Y = \text{screenport}_2)).$$
$$\forall X: \text{screenport}(X) \rightarrow (\exists^1_1 Y: \text{conn}(X,Y) \wedge \text{videoport}(Y)).$$
$$\forall X,Y: \text{screenport}(X) \wedge \text{conn}(X,Y) \rightarrow \text{videoport}(Y).$$
$$\forall X: \text{screenport}(X) \rightarrow (\exists^1_1 Y: \text{compnt}(X,Y) \wedge \text{Screen}(Y)).$$ ■

Using the port connection structure defined above, the constraint "a Videocard must be connected via $\text{videoport}_1$ with a Screen via $\text{screenport}_1$" can be written as follows.

EXAMPLE 13a.

*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 $screenport_1$)))))). ■

EXAMPLE 13b.

$$\forall X : \text{Videocard}(X) \rightarrow (\exists Y_1 : \text{videoport-of-videocard}(Y_1, X) \wedge$$
$$(\exists Y_2 : \text{portname}(Y_1, Y_2) \wedge (Y_2 = \text{videoport}_1)) \wedge$$
$$(\exists Y_3 : \text{conn}(Y_1, Y_3) \wedge$$
$$(\exists Y_4 : \text{compnt}(Y_3, Y_4) \wedge \text{Screen}(Y_4)) \wedge$$
$$(\exists Y_5 : \text{portname}(Y_3, Y_5) \wedge$$
$$(Y_5 = \text{screenport}_1)))).$$ ■

## 5. ANALYSIS

Although the basic frame structure and formal basis of description logics based languages makes them one of the natural choices for configuration representation, certain demands on expressiveness must be met. This section deals with the functions that are not currently expressible in OIL and therefore require extensions to the language: aggregation functions and built-in predicates, representation of resources, structural properties and *n*-ary relationships, and the possible incorporation of specific assertional languages. These will be considered in the following subsections.

### 5.1. Aggregation functions and built-in predicates

The current versions of OIL (Fensel et al., 2001*b*) and DAML+OIL (VanHarmelen et al., 2001) do not support aggregation functions (e.g., sum or avg) which are fundamental representation concepts frequently used in the configuration domain. Baader and Sattler (1998) provided concrete domains and aggregation functions over them as extensions to the basic description logic $\mathcal{ALC}$. In addition to aggregation functions, built-in predicates must be allowed in order to support comparisons on the results of aggregation functions as well as on local features. In configuration knowledge representation aggregation functions are used for designing resource constraints enforcing the balance between produced and consumed resources.

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 (see Fig. 1). Resource balancing tasks (Heinrich & Jüngst, 1991) are defined within a (sub)tree (context) of the configuration model. To map a resource balancing task into DD, additional attributes ($\text{res}_p$ and $\text{res}_c$ in the following) have to be defined for the component types acting as producers and consumers. In addition, we have to introduce aggregation functions as representation concepts, which are currently supported neither in OIL nor DAML+OIL. However, proposals exist (Baader & Sattler, 1998) to extend description logics by concepts that allow the modeling of such resource constructs. The following representation of aggregation functions is based on the formalism presented in Baader and Sattler (1998), where a set of predicates $P$ associated with binary relations (e.g., $\leq$, $\geq$, $<$, $>$) over a value domain $\text{dom}(D)$ and a set of aggregation functions $\text{agg}(D)$ (e.g., count,min,max,sum) are defined. Let $\phi$ be the path leading from the nearest common root to the concept whose features are aggregated. Then the definitions of Baader and Sattler (1998) require that all but the last one of the roles in $\phi$ must be features, that is, functional relations (see Section 3). Note that for our navpath expressions functional restrictions are also required.

RULE 7 (resource constraints). Let $p = \{p_1, p_2, \ldots, p_n\}$ be producing component types and $c = \{c_1, c_2, \ldots, c_m\}$ be consuming component types of resource *res* in GREP. Furthermore, let $\text{res}_p$ be a feature common to all component types in $p$, and $\text{res}_c$ be a feature common to the types in $c$, where the values of $\text{res}_p$ and $\text{res}_c$ are defined by the tagged values of the consumes and produces dependencies in GREP.

Using the notation of Baader and Sattler (1998), a resource constraint for $\text{DD}_{\text{DL}}$ can be expressed as

$$r\text{: } P(r_1^1 r_2^1 \cdots r_{n-1}^1 \textstyle\sum(r_n^1 \circ \text{res}_p), r_1^2 r_2^2 \cdots r_{m-1}^2 \sum(r_m^2 \circ \text{res}_c)),$$

where $r$ represents the nearest common root of the elements in $c$ and $p$, $P$ is a binary relation, and $< r_1^1, r_2^1, \ldots, r_n^1 >, < r_1^2, r_2^2, \ldots, r_m^2 >$ represent navigation paths from $r$ to the elements of $p$ and $c$.

A resource constraint for $\text{DD}_{\text{LOG}}$ can be expressed as

$$r(R) \wedge \text{allconsumers}(R, \text{Consumer}) \wedge$$
$$\text{allproducers}(R, \text{Producer}) \rightarrow$$
$$\sum_{s \in \text{consumer} \wedge \text{res}_c(s,V)} V \leq \sum_{t \in \text{producer} \wedge \text{res}_p(t,W)} W,$$

where $P$ is set to $\leq$ in this case.

The predicates *allconsumers* and *allproducers* define sets of all objects that appear as role fillers at the end of a navigation path and are (respectively) either consumers or producers. We use normal set notation (Hella et al., 2001) to express the grouping.

- $\text{allconsumers}(R, \{\text{Consumer}\}) \leftarrow \text{navpath}'(R, \text{Consumer}, r, c_1) \vee \cdots \vee \text{navpath}'(R, \text{Consumer}, r, c_m)$.
- $\text{allproducers}(R, \{\text{Producer}\}) \leftarrow \text{navpath}'(R, \text{Producer}, r, p_1) \vee \cdots \vee \text{navpath}'(R, \text{Producer}, r, p_n)$.

In the context of resources we define the macro $\text{navpath}'(X_1, X_n, c_1, c_n)$ as

$c1(X_1) \wedge \text{name}_{c1}(X_2, X_1) \wedge \text{name}_{c2}(X_3, X_2) \wedge \ldots \wedge \text{name}_{cn-1}(X_n, X_{n-1}) \wedge c_n(X_n)$, where $\text{name}_{ci}$ represents a haspart role in the path from the component type $c_1$ to component type $c_n$. ■

EXAMPLE 14a (capacity needed by *Software* ≤ 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* ∘ *Capacity*), sum(*hdunit-of-computer* ∘ *Capacity*)). ■

EXAMPLE 14b (needed Software capacity ≤ HDUnit capacity).

$$\text{Computer}(R) \wedge \text{allconsumers}(R, \text{Consumer}) \wedge \text{allproducers}(R, \text{Producer}) \rightarrow \sum_{s \in \text{Consumer} \wedge \text{Capacity}(s,V)} V \leq \sum_{t \in \text{Producer} \wedge \text{Capacity}(t,W)} W.$$

$$\text{allconsumers}(R, \{\text{Consumer}\}) \leftarrow \text{Computer}(R) \wedge \text{sw-of-computer}(\text{Consumer}, R) \wedge \text{Textedit}(\text{Consumer}) \vee \text{Computer}(R) \wedge \text{sw-of-computer}(\text{Consumer}, R) \wedge \text{DTPSoftware}(\text{Consumer}).$$

$$\text{allproducers}(R, \{\text{Producer}\}) \leftarrow \text{Computer}(R) \wedge \text{hdunit-of-computer}(\text{Producer}, R) \wedge \text{SCSIUnit}(\text{Producer}) \vee \text{Computer}(R) \wedge \text{hdunit-of-computer}(\text{Producer}, R) \wedge \text{IDEUnit}(\text{Producer}).$$ ■

### 5.2. Complex structural properties

Because trivial structural conditions lead to definitional overhead in DAML+OIL (e.g., when defining restrictions on port connections), additional concepts must be provided allowing the definition of roles with arity greater than 2. In effect, this would result in the definition of a separate constraint language or a relational language permitting arbitrary *n*-ary relations and disjuncts of positive literals, (e.g., to express alternative ports). Also, the description of more complex structural properties would be supported by permitting the usage of variables (Immerman, 1982; Cai et al., 1989; Schröder et al., 1996).

The utility of knowledge-based configurators in some domains crucially depends on the ability to configure structural (topological) properties such as the connecting of components by cables (see Fig. 5 for an example). In fact, the configuration of the cable connections is one of the most difficult parts in a configuration process (e.g., when configuring telecommunication switches; Fleischanderl et al., 1998). It is exactly at this stage that successful knowledge-based configuration has to show its advantage over traditional procedural configuration systems. In order to truly gain advantage over conventional approaches in domains where structural knowledge must be expressed, knowledge representation must support the formulation of these properties.

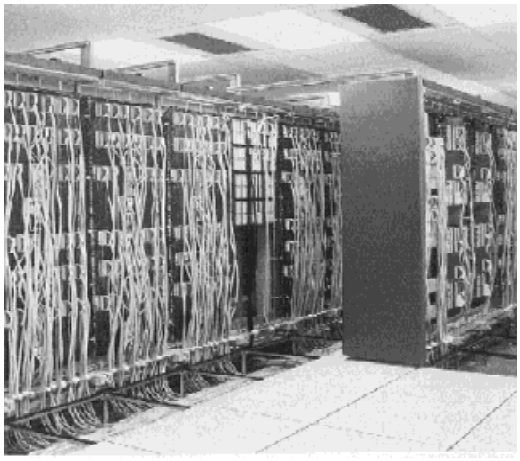

**Fig. 5.** The cable connections of a Siemens EWSD telecommunication system.

### 5.3. Decidability

Happily, most of the required means of expression are already available in the description logic designer's toolbox; however, the degree of expressivity required also leads to problems with respect to decidability of basic properties such as satisfiability or subsumption (Baader & Sattler, 1998). State of the art configurators achieve decidability by putting a predefined limit on the number of individuals and allowing only finite domains of values for features (constraint variables). Furthermore, only fixed concept hierarchies are allowed (as part catalogs are typically considered unchangeable), which reduces the importance of subsumption versus that of A-Box (assertion box, i.e., instance level) reasoning.

From a practical point of view, the systems occurring in configuration domains (such as telecom systems, railway safety systems, etc.) are designed to be understood (and, theoretically, configured) by human experts. The language these human experts use to describe the configuration constraints and to exchange knowledge is far more expressive than decidability considerations would allow. In the configuration process, simple heuristics are used to guide the construction process of a configuration. It is interesting that the configuration knowledge allows a quick conclusion as to whether a customer request can be fulfilled. In the rare cases where it is not easily decidable, organizations tend toward default decisions (e.g., if we are not sure that a valid configuration is possible, we just refuse the customer request). COCOS (Fleischanderl et al., 1998) and almost any practical applications of configurators crucially depend on the application of search heuristics. Relating this to our concepts, it must be possible to guide the model search

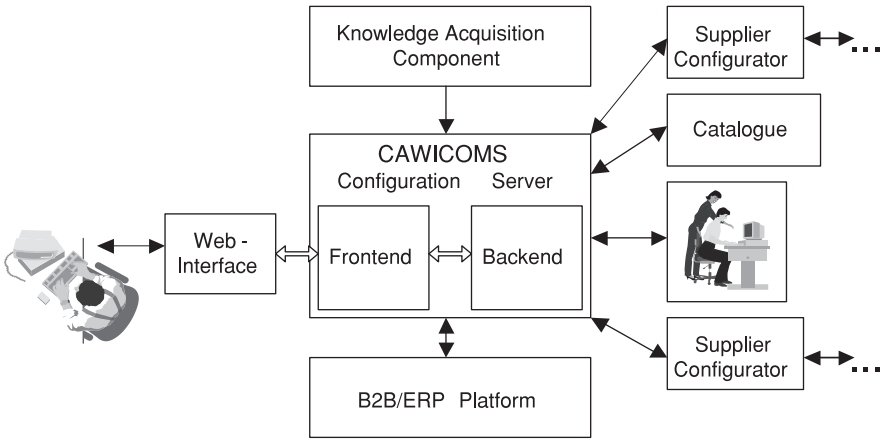


**Fig. 6.** The overall CAWICOMS architecture.

process and even to stop this process if certain thresholds are exceeded. Such heuristics include orderings about the types of individuals (e.g., which type a newly generated individual should take) or about assignments of values to features and filling of roles. These types of heuristics have proved quite sufficient and highly effective in the configuration of such large-scale systems as the EWSD (a digital switching system, Siemens), which can comprise more than 100 racks, 1,000 frames, 30,000 modules, and 10,000 cables.

## 6. THE CAWICOMS ENVIRONMENT

When selling complex products and services over the web, the shortcomings of current configuration technology become obvious. Web-based commerce places additional requirements on the interaction with configuration systems: customers with different needs, skill levels, or organizational backgrounds interact with the system. Therefore, interfaces must be provided that dynamically adapt to the needs of the customers (i.e., a *personalized* presentation of the configurable product is needed). Another shortcoming of current configuration technology concerns the cooperation between separate configurators. There is no central point of knowledge, and therefore a single-configurator approach is not appropriate (see the application scenario in Section 6.3). In addition to the distribution of configuration knowledge, *distributed problem solving* processes have to be supported in order to allow the calculation of solutions for distributed configuration tasks. In order to address the shortcomings of current configuration systems, we have developed an infrastructure for the creation of web-based, user-adaptive configuration systems that can perform distributed configuration of products and services by interacting with remote suppliers. This infrastructure is a result of the IST research project CAWICOMS. The concepts presented in this paper are the basis for the CAWICOMS Knowledge Acquisition Component (see Section 6.2). OIL representations can be generated from a UML configuration model and be used for configuration knowledge interchange. The knowledge acquisition component has been successfully tested using the scenario of distributed virtual private network (VPN) configuration (see Section 6.3).

### 6.1. Architecture

The overall architecture of the CAWICOMS (Ardissono et al., 2001) environment is shown in Figure 6. The central component of this architecture is the CAWICOMS Configuration Server, which is responsible for the integration of remote configuration systems (also product catalogs), the coordination of a distributed configuration process, and the personalized presentation of the configuration results to the customer. In this context the CAWICOMS Frontend provides personalization functionality and the CAWICOMS Backend provides distributed problem solving functionality. The configuration server has an interface to underlying B2B/ERP Platforms, which allows it to forward the result of a configuration session to subsequent processes (e.g., procurement transactions).

### 6.2. Knowledge acquisition

The concepts presented in this paper are implemented in the CAWICOMS Knowledge Acquisition Component. This workbench enables distributed configuration processes by supporting standardized configuration knowledge base development and interchange between different configuration environments; knowledge interchange is supported in UML/Extensible Markup Language (UML/XML; OMG, 1999) and OIL (Fensel et al., 2001*b*) using the OilEd ontology editor presented in Bechhofer et al. (2001). Constraints on the product structure can be modeled by directly annotating the UML configuration model; this functionality is provided by a Configuration Add-in for the CASE tool Rational Rose.[14] Based on the interchange of functional

[14] See www.rational.com.

architectures (Mittal & Frayman, 1989) of configurable products, a distributed configuration process can be started where distributed problem solving is based on standard distributed constraint satisfaction algorithms (e.g., Yokoo et al., 1998; Silaghi et al., 2000). The core reasoning mechanism is implemented by extending a commercial domain-independent configuration engine (ILOG JConfigurator[15]). These extensions were done without changing the core mechanisms of the configurator engine but only by using the built-in extensibility features. Note that the architecture of the knowledge acquisition component allows configuration knowledge base design for different configuration environments. The precondition for this is the provision of the corresponding translation routines.

### 6.3. Application scenario

One of the guiding application scenarios for the CAWICOMS project is the distributed configuration of VPNs. A VPN is an extension of an enterprise's private network that provides network services based on a public network infrastructure such as the Internet. A secure communication environment can be provided for defined communities of interest. The major advantage of such networks is that no expensive maintenance for company-wide area networks is necessary. The infrastructure for VPNs is provided by specialized solution providers who offer different subcomponents (services) for the VPN (e.g., telephony services, leased lines, firewalls, or computers). These subcomponents are integrated by resellers (integrated solution providers), who subsequently offer complete VPN solutions to their customers. In many cases the subcomponents from specialized solutions providers are configurable (i.e., the integration of such a set of components into a complete VPN solution can be interpreted as a distributed configuration task). Using the CAWICOMS Knowledge Acquisition Component, resellers can integrate functional product descriptions (functional architectures, Mittal & Frayman, 1989) into an integrated configuration model. This integrated model is the basis for the CAWICOMS Configuration Server, which starts a distributed configuration process and tries to calculate a solution for the given distributed configuration task.

## 7. RELATED WORK

### 7.1. Description logic based approaches

In the past, a number of authors have suggested general description logic based frameworks for configuration. First among these was the work by Owsnicki–Klewe (1988), which used a KL-ONE like language for automatically determining the most specific concept for each entity in a (fixed) configuration structure. The result of a configuration corresponds to a deductive closure starting from object instances and reaching a fix point. These deductions compute facts about objects such as concept membership or role fillers. The configuration result does not describe a complete configuration in case of disjunctions or existential quantification. As pointed out by Schröder et al. (1996), unlike other description logic approaches, this required defining concepts in terms of both necessary and sufficient terms. No new objects could be created. Compared to our definitions the approach of Owsnicki–Klewe (1988) computes consistent configurations, but (depending on the formulation of the knowledge base) it is not guaranteed that these configurations are valid because they may be incomplete.

The best known description logic based configuration system is the PROSE system used by AT&T (McGuinness & Wright, 1998). PROSE used a knowledge management and reasoning system based on the tractable description logic CLASSIC for checking the consistency of configuration solutions and to perform deductions. Configurations were computed (expanded) using a forward chaining production rule interpreter as also described by Weida (1996). From the view of our definitional framework, PROSE searches for consistent solutions where a completeness check can be achieved by closing roles. Note that CLASSIC does not apply the closed world assumption.

When configuring, many deductions are drawn which are not relevant for describing configurations. This was solved in PROSE by filters which restrict the output of the configuration process to the relevant deductions. This filter mechanism is achieved in our approach by defining the relevant concepts and roles needed to describe solutions to a configuration problem. An approach similar to PROSE was followed in the system developed by Weida (1996), which directly incorporated the so-called *closed terminology assumption* (CTA). As in PROSE, the description logic reasoner was used for consistency checking, with problem specific reasoning components producing the actual configuration. The CTA means that, as in CPS (see below) or most non-description logic configuration systems, the set of concepts and the subsumption hierarchy were assumed to be fixed; the parts in a final solution have to belong to the leaves in the concept hierarchy, which represent concrete types. The CTA limits the concepts an individual can instantiate. Roughly speaking, an individual (e.g., a particular component of a configuration) is only accepted to be part of a configuration if it can be monotonically extended to match an explicitly defined (concrete) concept. For example, if we learn from a component that its color is red but our domain knowledge base is ignorant about colors, then this component cannot be part of any configuration. The closed terminology assumption adds implicit additional constraints to a knowledge base. In order to achieve the same effect, we need additional constraints in our framework to exclude such components from valid configurations. We argue that both approaches (open vs. closed terminologies)

[15] See www.ilog.com.

have their advantages, depending on the application domain. Note that this is closely related to the distinction made in the diagnosis community between the consistency-based and the abductive-based approaches; that is, one has to decide whether diagnoses that do not explain all observations should be ruled out or not. A further aspect of such assumptions concerns the exchange of knowledge bases. The content of the knowledge base depends on the semantics and assumptions of the underlying reasoning system. Consequently, the exchange of knowledge bases eventually needs a transformation, if we submit a knowledge base written under the closed terminology assumption to a reasoning system employing open terminologies.

Generally, the most formally refined of the description logic approaches was the CPS approach that used a sorted feature logic to provide a formal model for configuration tasks (Klein et al., 1994; Buchheit et al., 1995). Configuration is defined as an abductive task using a domain-specific implication operator that provides a model $\tilde{C}$ based on $C \models_D \exists.S, C \models_D I$, where $D$ is the definitional knowledge; $C$ is the configuration, $\exists.S$ is the existential closure of the specification $S$; $I$ is a set of integrity constraints; and $S$ and $I$ hold in the extension $\tilde{C}$ of $C$, which is defined by $D$. As most logic-based configuration approaches (e.g., systems based on constraint satisfaction formalisms) and all configuration solvers in industrial use known to the authors use a consistency-based approach, we have taken that route instead.

Finally, Schröder et al. (1996) used a description logic based approach for a theoretical analysis of the the well-known PLAKON/KONWERK configurator (Cunis et al., 1989; Günther & Cunis, 1992; Günther, 1995) and its expressive means. The language defined for this study was designed to possess as many properties as possible of the original heterogeneous PLAKON/KONWERK representation (including explicit cover axioms that fixed the concept hierarchy and guaranteed disjunction between concepts). We found that a large amount of the expressiveness could be captured, although not all (PLAKON/KONWERK did, for example, support arbitrary *n*-ary constraints). Our view of the configuration problem and its solution is compatible with this approach.

### 7.2. Other approaches

In parallel to the description logic based work, a number of other approaches were recently developed to provide broad coverage of configuration domains with different properties.

The approach of Simons et al. (2002) is based on stable model semantics. As a consequence, unjustified components in a configuration solution are excluded. More formally speaking, some of the logical models of a set of sentences are not accepted if the stable model property is not fulfilled. Compared to our approach, such an approach of restricting the set of allowed models (one can also think of minimal model semantics) rejects configurations that would be valid in our case. The reason why we are more liberal is based on practical observations. For example, in the telecommunications domain it is common practice to add components to a configuration that are not necessary in the current situation but might be necessary in a next expansion step. Such configurations would not be accepted under stable model semantics because of the unjustified components, and explicit justifications would have to be entered into the knowledge base.

One of the most prominent methods to solve configuration problems is to apply constraint-based reasoning and its variants, such as dynamic and generative constraint satisfaction (e.g., Mittal & Frayman, 1989; Mittal & Falkenhainer, 1990; Heinrich & Jüngst, 1991; Fleischanderl et al., 1998; Mailharro, 1998) just to name some examples of the vast number of applications. In constraint satisfaction we are searching for an assignment to all variables in order to satisfy all constraints. This corresponds to our consistency-based definition of a configuration problem, where the configuration result is described by the set of variables and their assignments. Completeness of the configuration is automatically achieved in the case where the set of all variables is known to be static because all variables receive a value assignment. Note that dynamic CSPs can be transformed to static ones. Generative constraint satisfaction is different insofar that the number of variables is not known initially. However, the solution described in Stumptner et al. (1998) assures that the configuration is valid by checking that all attributes are assigned and no additional components and connections exist. Likewise, the resource-based (Heinrich & Jüngst, 1991) approach fits our definitions nicely because in this paradigm we are searching for a set of resources such that all resource constraints are satisfied.

The goal of Aldanondo et al. (2000) is to propose a set of modeling concepts enabling a noncomputer specialist to describe generic configuration models. Based on a set of requirements stemming from different classes of configuration problems a set of modeling concepts is presented which is based on the dynamic CSP approach (Mittal & Falkenhainer, 1990). This approach to configuration knowledge representation has its advantages when modeling configuration problems for dynamic CSP solving. Compared to the approach presented in this paper, the problem of representing configuration knowledge on a more abstract level is solved by providing graphical representations for problem variables. In Ramachandran and Gil (1999) the EXPECT (Swartout & Gil, 1995) approach for knowledge acquisition is applied to the configuration domain. The EXPECT configuration knowledge acquisition approach is based on a proprietary representation, whereas the approach presented in this paper supports knowledge acquisition and knowledge interchange on the basis of a common foundation for configuration knowledge representation; the use of UML as the configuration knowledge representation language permits the integration of configuration technology into industrial software development processes. Smith et al.

(1988) present an approach based on the idea of configuration knowledge representation using dependency diagrams. These diagrams depict the dependencies between different parameters of the configurable product. The graphical notation is used as the communication basis between the domain expert and the knowledge engineer but does not allow the automated derivation of configuration knowledge bases.

The definition of a common representation language to support knowledge interchange between and integration of different knowledge-based systems is an important issue in the configuration domain. In Soininen et al. (1998) one approach to collect relevant concepts for modeling configuration knowledge bases is presented. The defined ontology is based on Ontolingua (Gruber, 1992) and represents a synthesis of resource-based, function-based, connection-based, and structure-based configuration approaches. This ontology is a kind of metaontology that is similar to the UML profile for configuration models presented in this paper. Conforming to the definition of Chandrasekaran et al. (1999), a UML configuration model is an ontology, that is, it restricts the sort of objects relevant for the domain and defines the possible properties of objects and the relationships between objects. Felfernig et al. (2000*a*) present an approach to automatically translate UML configuration models into a corresponding consistency-based definition of a configuration problem. The work presented in this paper is an extension of the work of Felfernig et al. (2000*a*) in the sense that a joint foundation for the representation of configuration problems is established by giving a description logic based definition of a configuration problem and showing the equivalence to existing consistency-based definitions (Felfernig et al., 2000*b*).

The work of Cranefield (2001) 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 translation rules allowing the automatic derivation of executable configuration knowledge bases. We show the correspondence between Semantic Web ontology languages and UML 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 we present the corresponding representation in OIL.

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 means of expression must be provided in terms of new operators or relaxed restrictions on the language, as discussed in Section 5. Finally, we assumed the unique name assumption from the start, as this is a necessity in each problem domain that exhibits repetitive subcomponents (i.e., tasks where the same type of component can occur repeatedly but the topology of the connections plays a role).

Within the Semantic Web community there are ongoing efforts to increase the expressiveness of web ontology languages. DAML-L (McIlraith et al., 2001) is a language that builds on the basic concepts of DAML. XML Rules (Grosof, 2001) and CIF (i.e., Constraint Interchange Format; Gray et al., 2001) are similar approaches with the goal to provide rule languages for the Semantic Web.

Solutions already exist for the integration of product catalogs within marketplace environments (Fensel et al., 2001*a*). The basic approach is to provide a standard representation language and to provide a set of transformation concepts for integrating proprietary product representations. However, standard representation languages for simple products do not consider the basic properties of configurable products; exactly these properties were discussed in this paper and corresponding solutions for the integration of configurable products and services have been developed in the CAWICOMS project (Ardissono et al., 2001).

## 8. CONCLUSION

In this paper we have shown how to apply Semantic Web ontology languages for configuration knowledge representation. We have given a description logic based definition of a configuration problem and shown its equivalence with corresponding consistency-based definitions. A consequence of this equivalence is that configuration problems represented in standard description logics (OIL or DAML+OIL or Borgida, 1996) can be transformed into configuration problems represented in predicate logic (dyadic predicate logic with counting quantifiers and at most three free variables in each subformula) and vice versa. Consequently, we provide a common foundation that enables joint research activities and exploration of results. With respect to ongoing efforts to extend DAML+OIL, our paper contributes a set of criteria that must be fulfilled in order to use such a language for full-fledged configuration knowledge representation. It follows that DAML+OIL and OIL must be extended in order to cover requirements imposed by *de facto* standard configuration ontologies. By using UML for configuration knowledge representation, we support effective sharing and integration of configuration knowledge on a graphical level, which has become one of the major issues in the context of distributed configuration problem solving.

## ACKNOWLEDGMENTS

We thank the reviewers for their suggestions that certainly improved this paper.

## REFERENCES

Aldanondo, M., Moynard, G., & Hamou, K. (2000). General configurator requirements and modeling elements. *Proc. Workshop on Configuration, ECAI 2000*, pp. 1–6.

Ardissono, L., Felfernig, A., Friedrich, G., Jannach, D., Zanker, M., & Schäfer, R. (2001). Customer-adaptive and distributed online product configuration in the CAWICOMS Project. *Proc. Workshop on Configuration, in Conjunction with the 17th International Conf. Artificial Intelligence (IJCAI-2001)*, pp. 8–14.

Artale, A., Franconi, E., Guarino, N., & Pazzi, L. (1996). Part–whole relations in object-centered systems: An overview. *Data & Knowledge Engineering 20(3)*, 347–383.

Baader, F., & Sattler, U. (1998). Description logics with concrete domains and aggregation. *Proc. 13th European Conf. Artificial Intelligence (ECAI '98)*, pp. 336–340.

Barker, V., O'Connor, D., Bachant, J., & Soloway, E. (1989). Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM 32(3)*, 298–318.

Bechhofer, S., Horrocks, I., Goble, C., & Stevens, R. (2001). OilEd: A reasonable ontology editor for the Semantic Web. *Proc. Joint German/Austrian Conf. on AI*, pp. 396–408.

Burners–Lee, T. (2000). *Weaving the Web*. New York: Harper Business.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications. Object Technology Series*. Reading, MA: Addison–Wesley.

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

Buchheit, M., Klein, R., & Nutt, W. (1995). *Constructive Problem Solving: A Model Construction Approach Towards Configuration*. Technical Report TM-95-01. Saarbrücken, Germany: DFKI.

Cai, J., Fürer, M., & Immerman, N. (1989). An optimal lower bond on the number of variables for graph identification. *Proc. 30th IEEE Symposium on FOCS*, pp. 612–617.

Chandrasekaran, B., Josephson, J., & Benjamins, R. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems 14(1)*, 20–26.

Cranefield, S. (2001). UML and the Semantic Web. *Semantic Web Working Symposium*.

Cunis, R., Günter, A., Syska, I., Peters, H., & Bode, H. (1989). PLAKON—An approach to domain independent construction. *Proc. Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 298–318.

Felfernig, A., Friedrich, G., & Jannach, D. (2000*a*). UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering 10(4)*, 449–469.

Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2000*b*). Consistency-based diagnosis of configuration knowledge bases. *Proc. 14th European Conf. Artificial Intelligence (ECAI 2000)*, pp. 146–150.

Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M., & Fett, A. (2001*a*). Product data integration in B2B E-commerce. *IEEE Intelligent Systems 16(4)*, 54–59.

Fensel, D., VanHarmelen, F., Horrocks, I., McGuinness, D., & Patel–Schneider, P. (2001*b*). OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems 16(2)*, 38–45.

Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H., & Stumptner, M. (1998). Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems 13(4)*, 59–68.

Gray, P., Hui, K., & Preece, A. (2001). An expressive constraint language for Semantic Web applications. *Proc. IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pp. 46–53.

Grosof, B. (2001). Standardizing XML rules. *Proc. IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pp. 2–3.

Gruber, T. (1992). *Ontolingua: A Mechanism to Support Portable Ontologies*. Technical Report KSL 91-66. Stanford, CA: KSL.

Gruber, T., Olsen, R., & Runkel, J. (1996). The configuration design ontologies and the VT elevator domain theory. *International Journal of Human–Computer Studies 44(3/4)*, 569–598.

Günther, A., & Cunis, R. (1992). Flexible control in expert systems for construction tasks. *Journal of Applied Intelligence 2(4)*, 369–385.

Günther, A. (1995). *Wissensbasiertes Konfigurieren: Ergebnisse aus dem Projekt PROKON*. Sankt Augustin, Germany: Infix.

Heinrich, M., & Jüngst, E. (1991). A resource-based paradigm for the configuring of technical systems from modular components. *Proc. 7th IEEE Conf. AI Applications (CAIA)*, pp. 257–264.

Hella, L., Libkin, L., Nurmonen, J., & Wong, L. (2001). Logics with aggregate operators. *Journal ACM 48(4)*, 880–907.

Immerman, N. (1982). Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences 25*, 76–98.

Jacobson, I., Christerson, M., & Övergaard, G. (1992). *Object-oriented Software Engineering—A Use-Case Driven Approach*. Reading, MA: Addison–Wesley.

Klein, R., Buchheit, M., & Nutt, W. (1994). Configuration as model construction: The constructive problem solving approach. *Proc. 3rd Int. Conf. Artificial Intelligence in Design, AID'94*, pp. 201–218.

Mailharro, D. (1998). A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4)*, 383–397.

McGuinness, D., & Wright, J. (1998). Conceptual modeling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4)*, 333–344.

McIlraith, S., Son, T., & Zeng, H. (2001). Mobilizing the Semantic Web with DAML-enabled web services. *Proc. IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pp. 29–39.

Mittal, S., & Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. *Proc. National Conf. Artificial Intelligence (AAAI 90)*, pp. 25–32.

Mittal, S., & Frayman, F. (1989). Towards a generic model of configuration tasks. *Proc. 11th Int. Joint Conf. Artificial Intelligence*, pp. 1395–1401.

OMG. (1999). XMI Specification. Available on-line at www.omg.org.

Owsnicki–Klewe, B. (1988). Configuration as a consistency maintenance task. *Proc. GWAI-88—The 12th German Workshop on Artificial Intelligence*, pp. 77–87.

Ramachandran, S., & Gil, Y. (1999). Knowledge acquisition for configuration tasks: The EXPECT approach. *Proc. AAAI Workshop on Configuration*, Technical Report WS-99-05, pp. 29–34.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice–Hall.

Rumbaugh, J., Jacobson, I., & Booch, G. (1998). *The Unified Modeling Language Reference Manual*. Reading, MA: Addison–Wesley.

Sattler, U. (2000). Description logics for the representation of aggregated objects. *Proc. 14th Eur. Conf. Artificial Intelligence (ECAI 2000)*, pp. 239–243.

Schröder, C., Möller, R., & Lutz, C. (1996). A partial logical reconstruction of PLAKON/KONWERK. *Proc. Workshop on Knowledge Representation and Configuration*, DFKI Memo D-96-04, pp. 55–64.

Silaghi, M., Sam–Haroud, D., & Faltings, B. (2000). Asynchronous search with aggregations. *Proc. 17th National Conf. Artificial Intelligence (AAAI)*, pp. 917–922.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence 138(1–2)*. 181–234.

Smith, C., Inder, R., & Chung, P. (1988). Knowledge acquisition and representation for product configuration. *Proc. 1st Int. Conf. Industrial and Engineering Applications of AI and Expert Systems (IEA/AIE'88)*, pp. 805–811.

Soininen, T., Tiihonen, J., Männistö, T., & Sulonen, R. (1998). Towards a general ontology of configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4)*, 357–372.

Stumptner, M. (1997). An overview of knowledge-based configuration. *AI Communications 10(2)*, 111–125.

Stumptner, M., Friedrich, G., & Haselböck, A. (1998). Generative constraint-based configuration of large technical systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4)*, 307–320.

Swartout, B., & Gil, Y. (1995). EXPECT: Explicit representation for flexible acquisition. *Proc. 9th Knowledge Acquisition for Knowledge-Based Systems Workshop* (KAW-95).

VanHarmelen, F., Patel–Schneider, P., & Horrocks, I. (2001). *A Model–Theoretic Semantics for DAML+OIL*. Available on-line at www.daml.org.

Weida, R. (1996). Closed terminologies in description logics. *AAAI Fall Symposium 1996*, pp. 11–18.

Wright, J., Weixelbaum, E., Vesonder, G., Brown, K., Palmer, S., Berman, J., & Moore, H. (1993). A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T Network Systems. *AI Magazine 14(3)*, 69–80.

Yokoo, M., Durfee, E., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem. *IEEE Transactions on Knowledge and Data Engineering 10(5)*, 673–685.

**Alexander Felfernig** is a Research Assistant in computer science at the University of Klagenfurt, Austria, where he obtained his MS and PhD degrees in applied informatics. His research interests are in the fields of knowledge-based configuration, knowledge representation, and the Semantic Web. Within the European Union project CAWICOMS, Dr. Felfernig focused on design and implementation of the knowledge acquisition component.

**Gerhard Friedrich** is full Professor of computer science at the University of Klagenfurt, Austria, where he directs the Computer Science and Manufacturing Research Group. His research interests include configuration, knowledge representation, and diagnosis. Dr. Friedrich received his MS and PhD degrees in computer science from the Vienna University of Technology.

**Dietmar Jannach** is a Research Assistant in computer science at the University of Klagenfurt, Austria, where he obtained his MS and PhD degrees in applied informatics. His current research areas include knowledge-based configuration and the integration of model-based diagnosis techniques into the configurator development and deployment process. Within the CAWICOMS project Dr. Jannach focused on the development and implementation of distributed problem solving algorithms.

**Markus Stumptner** is full Professor of computer science at the University of South Australia, where he directs the Advanced Computing Research Centre. His research interests include object-oriented modeling, knowledge representation, and model-based reasoning in areas such as configuration and diagnosis. Dr. Stumptner received his MS and PhD degrees in computer science from the Vienna University of Technology.

**Markus Zanker** is a Research and Teaching Assistant in computer science at the University of Klagenfurt, Austria, where he obtained MS degrees in applied informatics and business administration and a PhD in applied informatics. His research interests are in the field of configuration, distributed CSP, and the Semantic Web. He participated in the CAWICOMS project and focused on distributed problem solving.