# A Joint Foundation for Configuration in the Semantic Web

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

**Abstract.** Product configuration is a major commercial application of knowledge-based systems, and joint configuration by multiple business partners is becoming a key application in today's highly specialized economy. The required integration of configuration knowledge is a challenging task due to the variety of knowledge representation formalisms used in commercial configurators. Ontology languages such as DAML+OIL provide an infrastructure for the Semantic Web with the goal of intelligent information integration. The aim of this paper is to show the applicability of such languages for building configuration knowledge bases. We join the two major streams in knowledge-based configuration (description logics on one hand and predicate logic, including constraint-based and resource-balancing techniques on the other) by giving a description logic based definition of a configuration task and showing its equivalence with existing consistency-based definitions. We show that Semantic Web ontology languages can be applied to configuration by formalizing language elements relevant for building configuration knowledge bases and discuss extensions needed in order to provide full fledged configuration knowledge representation. The result is a common basis for current configuration approaches on the Semantic Web that is necessary for the provision of joint configuration services.

## 1 Introduction

Knowledge-based configuration has a long history as a successful AI application area (e.g., [3, 10, 12, 13, 15, 19]). Starting with rule-based systems such as R1/XCON [3], various higher level representation formalisms were developed 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 eCommerce environments. The trend towards 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.

For this integration, configurators must have a clear 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., [10, 13, 14]) and resource balancing methods (e.g., [12]). The second approach uses description logics as knowledge representation and reasoning mechanism (e.g., [19]). Clearly, an integration of these approaches is a major milestone for configurator integration.

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 allowing easy integration of existing proprietary configuration environments. Ontology representation languages such as OIL [9] or DAML+OIL [18] developed in the context of the Semantic Web [4] are well suited for designing and sharing ontologies. These languages are strongly influenced by description logics and therefore possess clear declarative semantics, providing one important precondition for the exchange of knowledge. Nonetheless, their roots in description logics reinforces the need for the definition of a common view of a configuration task, so that predicate logic based representations can be mapped to them.

The practical consequence of a commonly accepted problem definition and knowledge semantics in joint provisioning of configuration services is a well defined interface between configurator implementations. Proprietary configuration systems can be independently implemented following different approaches and are still able to interoperate. We only require that cooperating configurators deliver valid solutions w.r.t. the common definition of the problem, the solution, and the semantics of the exchanged knowledge.

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 [8] - the major result of this equivalence is that configuration tasks defined in terms of description logics and predicate logic can easily be transformed into each other and consequently be represented in ontology representation languages such as DAML+OIL. Using concepts of OIL[3], we present the constituting elements of a configuration knowledge representation language by formalizing modeling concepts of de facto standard configuration ontologies [7, 17] employed in industrial applications[4]. In addition, we point out extensions needed to apply OIL and DAML+OIL for full fledged configuration knowledge representation. As a result, we provide a common basis for knowledge representation in configuration problem solving, thus enhancing the applicability of configuration technology to Web-based environments.

The rest of the paper is organized as follows. In Section 2 we intro-

[1] Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria, email: {*felfernig,friedrich,jannach,zanker*}@ifit.uni-klu.ac.at.

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

[3] For presentation purposes we employ OIL text - this representation can easily be transformed into a corresponding DAML+OIL representation.

[4] Note that these differ somewhat from the configuration ontology that was described for demonstration purposes in [11].

duce an example that provides an overview of the modeling concepts required for building configuration knowledge bases. In Section 3 we give a description logics based definition of a configuration task and show its equivalence to the consistency-based definition given in [8]. In Section 4 we describe an OIL-based formalization of the modeling concepts presented in Section 2 and summarize the results. Section 5 closes with conclusions.

## 2 Configuration domain specific modeling concepts

In the following we discuss a set of relevant modeling concepts for building configuration knowledge bases. These concepts are extracted from the configuration ontologies defined in [7, 17]. Figure 1 shows the simplified structure of a configurable *Computer* system which is composed of the following representation concepts.

*Component types* represent the parts, a final product is built of - they are characterized by attributes (e.g., the component type *CPU* is characterized by the attribute *clockrate*). Component types with a similar structure are arranged in a *generalization* hierarchy and represent choices for the configurable product (e.g., in the final configuration an instance of *CPU* can be either a *CPU1* or a *CPU2*). *Part-whole* relationships can be considered as bill-of-material representations semantically enriched with multiplicities, stating a range of how many subparts an aggregate can consist of (e.g., a *MB* must contain at least one *CPU* and at most two *CPUs*). In addition to the number and types of different components, the product topology may be important in a final configuration as well, i.e., how the components are interconnected to each other (e.g., which *videoport* is used in the configuration to connect the *Videocard* with the *Screen*).

Additionally, a set of constraints specifies allowed combinations of component- and attribute settings in the final configuration. Some component types cannot be used in the same final configuration (they are *incompatible*). E.g., we can impose the constraint on the product structure of Figure 1 that a *CPU1* is incompatible with a motherboard *MB2*. In some cases, the existence of a component of a certain type *requires* the existence of an instance of another specific type. Regarding the product structure of Figure 1, we can impose the constraint that the existence of a *CPU2* requires the existence of a *MB*2. Finally, parts of a configuration problem can be interpreted as a resource balancing task, where some of the components are *producers* and others are *consumers*. In the final configuration, consumers and producers must be balanced w.r.t. some resource balancing criteria - e.g., the amount of installed hard-disk capacity is the upper bound for the capacity requirements of the installed software. In Section 4 we will show how to represent these concepts in extended OIL [9].

## 3 Defining configuration tasks

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 [5]. 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[5].

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

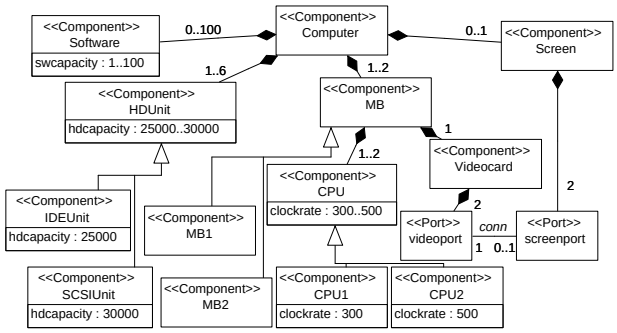


**Figure 1.** Example configuration model

In the following we will define a solution of a configuration problem based on the interpretation of concepts and roles. In addition we require that roles in $CLANG_{DL}$ are defined over the domains given in $C_{config}$, i.e., we add for each $R_i \in R_{config}$ the role descriptions $range(R_i) = CDom$ and $dom(R_i) = CDom$ for $CDom \doteq \bigsqcup_{C_i \in C_{config}} C_i$ to the knowledge base $KB_{DL}$ if such descriptions are not subsumed by other descriptions already contained in the knowledge base.

**Example 1:** In this example we use a part of our *Computer* ontology (see Figure 1) that comprises *CPUs* and *MBs*. On each *MB* at least one but at most two *CPUs* are plugged in (constraint $c_1$). A *CPU* must always be mounted on a *MB* (constraint $c_2$). A *CPU* of type *CPU2* must be mounted on a *MB* of type *MB2* (constraint $c_3$). The domain description $DD_{DL}$={
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 "two CPUs of type *CPU1* and one CPU of type *CPU2*" is expressed by $SRS_{DL}$ = { (*instance-of c1 CPU1*), (*instance-of c2 CPU1*), (*instance-of c3 CPU2*) }. The configuration language $CLANG_{DL}$ is defined by $C_{config} = \{CPU1, CPU2, MB1, MB2\}$ and $R_{config} = \{$*mb-of-cpu*$\}$.

In our example we do not include the concepts $CPU$ and $MB$ and the role *cpu-of-mb* in $CLANG_{DL}$ since we are only interested in the

[5] In the following we assume that the reader is familiar with the concepts of OIL. See [9] for an introductory text.

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 (non-empty universe) of values, and $(\cdot)^{\mathcal{I}}$ 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 which describe configurations in order to make sure that different identifiers for individuals (e.g., modules of a system) refer to different individuals. This UNA can be lifted if necessary for concepts and roles which are not used to describe configurations. In the following we give a description logic based definition of a configuration problem and show its equivalence with consistency-based definitions given in the literature [8]. This definition serves as a joint foundation of configuration knowledge representation in the Semantic Web.

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

**Example 2:** A valid configuration for our example knowledge base is $CONF_{DL}$ = $\{\langle CPU1, \{c1, c2\}\rangle$, $\langle CPU2, \{c3\}\rangle$, $\langle MB1, \{m1\}\rangle$, $\langle MB2, \{m2\}\rangle$, $\langle$ *mb-of-cpu*, $\{\langle c1, m1\rangle, \langle c2, m1\rangle$ $\langle c3, m2\rangle\}\rangle\}$. □

We also have to describe component parameter settings in addition to components and their connections. Using description logics, 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 $dom(D)$ [16] 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 $KB_{DL}$ be a configuration knowledge base, $CLANG_{DL} = C_{config} \cup R_{config}$ a configuration language, and $CONF_{DL} = COMPS \cup ROLES$ a description of a configuration.

The concepts $C_i$ are defined by the component axioms $AX_{COMPS} =$

$$\{C_i \doteq \text{one-of}[ci_1, \ldots, ci_{n_i}] | \langle C_i, INDIVS_{C_i} \rangle \in COMPS \text{ where}$$

$$INDIVS_{C_i} = \{ci_1, \ldots, ci_{n_i}\}\}.$$

The roles $R_j$ are defined by the role axioms $AX_{ROLES} =$

$$\{R_j \doteq \bigsqcup_{\langle rj, sj \rangle \in TUPLES_{R_j}} \text{product}[\text{one-of}[rj], \text{one-of}[sj]] |$$

$$\langle R_j, TUPLES_{R_j} \rangle \in ROLES \text{ with}$$

$$TUPLES_{R_j} = \{\langle rj_1, sj_1 \rangle, \ldots, \langle rj_{m_j}, sj_{m_j} \rangle\}\}.$$

$CONF_{DL}$ is a valid configuration *iff* $KB_{DL} \cup AX_{COMPS} \cup AX_{ROLES}$ 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 [5]. An alternative to the *product* constructor would be a constructor that (in a similar manner to the *one-of* constructor for concepts) permits the enumeration of permissible entries for a role. However, the usual complexity problems with the *product* constructor do not actually arise since we only apply *product* to singleton arguments.

**Example 3:** In order to verify that a given configuration is valid w.r.t. our example $KB_{DL}$, 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*$[c1]$, *one-of*$[m1]$] $\sqcup$ *product*[*one-of*$[c2]$, *one-of*$[m1]$]$\sqcup$ *product*[*one-of*$[c3]$, *one-of*$[m2]$].□

We now give a consistency-based definition of a configuration problem using predicate logic (this definition corresponds to the definition given in [8]) and show the equivalence with the description logic based definition given before.

**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 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, i.e.,

$DD_{LOG} = \{$
$\forall X : MB(X) \leftrightarrow MB1(X) \vee MB2(X).$
$\forall X : \neg MB1(X) \vee \neg MB2(X).$
$\forall X : MB(X) \rightarrow \exists_1^2 Y :$ *mb-of-cpu*$(X, Y).$
$\forall X : CPU(X) \leftrightarrow CPU1(X) \vee CPU2(X).$
$\forall X : \neg CPU1(X) \vee \neg CPU2(X).$
$\forall X : CPU(X) \rightarrow \exists_1^1 Y :$ *mb-of-cpu*$(Y, X).$
$\forall X : CPU2(X) \rightarrow \exists_1^1 Y :$ *mb-of-cpu*$(Y, X) \wedge MB2(Y).$
$\forall X : \neg MB(X) \vee \neg CPU(X).$
$\forall X, Y :$ *mb-of-cpu*$(X, Y) \leftrightarrow$ *cpu-of-mb*$(Y, X).$
$\forall X, Y :$ *mb-of-cpu*$(X, Y) \rightarrow$ *mb*$(X) \wedge$ *cpu*$(Y)\}.$
$SRS_{LOG} = \{CPU1(c1).\ CPU1(c2).\ CPU2(c3).\}.$
$CLANG_{LOG} = \{CPU1, CPU2, MB1,$
*MB2*, *mb-of-cpu*$\}$.□

**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 [8]. 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 $\forall X : CPU1(X) \Rightarrow (\bigvee_{Z \in CONF_{LOG}} CPU1(X) = Z)$ for the predicate $CPU1$ and similar axioms for $CPU2$ and *mb-of-cpu*.

Note that $\bigvee_{Z \in CONF_{LOG}} CPU1(X) = Z$ serves as a macro which is expanded based on the elements in $CONF_{LOG}$. We refer to $CONF_{LOG}$ extended by completeness axioms as $\widehat{CONF}_{LOG}$.

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

The completeness axiom for $CPU1$ is $\forall X : CPU1(X) \Rightarrow CPU1(X) = CPU1(c1) \vee CPU1(X) = CPU1(c2)$, 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.

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, i.e., axioms to formulas with no free variables, concepts to formulas with one free variable $x$, and roles to formulas with two free variables $x$ and $y$.

Borgida [5] provides such a translation function $\mathcal{T}\langle\cdot\rangle$ such that concepts, roles, terms, and axioms are translated into equivalent formulas in predicate logic. $KB_{LOG} = \mathcal{T}\langle KB_{DL}\rangle$ where $KB_{LOG}$ is satisfiable *iff* $KB_{DL}$ is satisfiable.

**Remark 2 (Equivalence of configuration problems):**

Let $CLANG_{LOG} = CLANG_{DL}$ where each concept is interpreted as monadic and each role is interpreted as dyadic predicate. $CONF_{LOG}$ describes the actual configuration by two sets of facts $CONF_{LOG} =$ *COMP-facts* $\cup$ *ROLE-facts*. The construction of $CONF_{LOG}$ is based on $CONF_{DL} = COMPS \cup ROLES$ where *COMP-facts* $= \{C_i(ci) | ci \in INDIVS_{C_i}, C_i \in C_{config}\}$ and *ROLE-facts* $= \{R_j(rj, sj) | \langle rj, sj\rangle \in TUPLES_{R_j}, R_j \in R_{config}\}$. $DD_{LOG} = \mathcal{T}\langle DD_{DL}\rangle$, and $SRS_{LOG} = \mathcal{T}\langle SRS_{DL}\rangle$.

$CONF_{LOG}$ is a valid configuration for $(DD_{LOG}, SRS_{LOG}, CLANG_{LOG})$ *iff* $CONF_{DL}$ is a valid configuration for $(DD_{DL}, SRS_{DL}, CLANG_{DL})$. □

Remark 2 follows from Remark 1 and the equivalence property of the translation function. Note that the completeness axioms correspond exactly to the translation of the axioms $AX_{COMPS}$ and $AX_{ROLES}$ by applying the translation function proposed in [5].

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 first order predicate or propositional logic on the other hand can be easily transformed to each other. Second, since description logics without transitive closure are equally expressive to dyadic predicate logic with at most three (counting) quantifiers [5] it follows that the predicate logic based approach is strictly more expressive than the description logics based approach, implying that certain logic constructions have to be simulated by more complex description logic constructions, and also that certain complex structural restrictions [6] will not be expressible in the language directly but will have to be incorporated using a more expressive assertional language.

## 4 Building configuration knowledge bases for the Semantic Web

In the following we show how to represent the configuration domain specific modeling concepts discussed in Section 2 using OIL.

**Component types.** The representation of component types is straightforward and has already been shown in Section 3. Component types are transformed into corresponding concepts, attributes into role definitions constraining datatype and cardinality (the cardinality of attribute roles is set to 1). Attributes as well as abstract roles are inherited by the defined subconcepts. In most configuration environments the semantics of a generalization hierarchy are disjunctive (*disjoint* concept) and complete by default (each instance of a supertype is also an instance of exactly one of its subtypes).

**Part-whole relationships.** Part-whole relationships play an important role in many application domains having quite different semantics (see [1], [16]). Basically, a part-whole relationship can be expressed using the two roles *PartOf* and *HasPart*, where *PartOf* is the inverse role of *HasPart*. In OIL or DAML+OIL we can define - depending on the application domain - different semantics for part-whole relationships by imposing restrictions on the usage of the corresponding roles. Sattler [16] presents an extension of the basic description logic $\mathcal{ALC}$ with concepts for adequate representation of part-whole relationships - in this context a categorization of different facets of part-whole relationships is given. In our working example (Figure 1) we only allow part-whole relationships where a component is part-of exactly one other component (this is also denoted as exclusive part-whole relationship). Such exclusivity restrictions can be introduced by restricting the cardinality of the corresponding role to 1, e.g., *slot-constraint mb-of-cpu cardinality 1 MB*, where the role *mb-of-cpu* must be interpreted as a subslot of the role *PartOf*. Furthermore, restrictions concerning the cardinality of parts must be added to the concept definition of the whole, e.g., *slot-constraint cpu-of-mb min-cardinality 1 CPU* (*max-cardinality 2 CPU*).

**Port connections.** As mentioned above, certain predicate logic constructs must be simulated by more complex description logic arrangements [5]. In terms of predicate logic, port connections can be represented using a predicate *conn*$(C_1, P_1, C_2, P_2)$, i.e., component $C_1$ is connected via port $P_1$ with component $C_2$ via port $P_2$ - e.g., *conn*$(videocard1, videoport2, screen1, screenport1)$ describes a connection between $videoport2$ of $videocard1$ and $screenport1$ of $screen1$. In order to represent port-based connections, we have to introduce a *Port* concept, which is characterized by a role indicating the related component concept (role *compnt*) and a role which indicates the used portname of the connection (role *portname*) - additionally, a role *conn* indicates the connection to the second involved *Port* concept. Although a representation of port connections is possible with OIL or DAML+OIL, the original knowledge of domain experts is split into multiple pieces of knowledge which are hard for these domain experts to understand. Also, note that the connections via such a connection object must be unique and therefore the roles must be bidirectionally defined using the *inverse* role constructor. The constraint that a *Videocard* must be connected via $videoport2$ with a *Screen* via $screenport1$ can be formulated as

**Example 6:** *Videocard:*(*slot-constraint videoport has-value*((*slot-constraint portname has-value* (*one-of videoport2*)) *and* (*slot-constraint conn has-value* ((*slot-constraint compnt has-value Screen*) *and* (*slot-constraint portname has-value* (*one-of screenport1*)))))). □

The constituting elements of such port constraints are *navigations* representing role compositions between connected concepts.

**Navigation in product structure.** In the following we discuss a set of representative constraint concepts which are frequently used in the configuration domain [7, 17]. The constraints consist of navigation expressions over concepts which are represented by complex paths over abstract roles.

**Definition 6 (Navigation expression):** *Given two concepts $C_1$ and $C_2$, a navigation expression from $C_1$ to $C_2$ is formulated as a*

*sequence of existential role quantifications*

*(slot-constraint $r_1$ has-value(slot-constraint $r_2$ has-value ... (slot-constraint $r_n$ has-value $C_2$)))*,

*where <$r_1$, $r_2$, ..., $r_n$> denotes a sequence of roles along the navigation path from $C_1$ to $C_2$ ($r_1$ is a role of $C_1$ and $C_2$ is in the range of $r_n$). In the following we use the expression navpath($C_1$, $C_2$) as short hand notation for a navigation path concept from $C_1$ to $C_2$.* □

**Definition 7 (Common root):** *A concept $C_R$ is denoted as common root of a set of concepts $C = \{C_1, ..., C_n\}$, if there exists a navigation path from $C_R$ to each concept of C.* □

**Incompatibilities.** An incompatibility constraint between concepts $C_1$, $C_2$, ..., $C_n$ can be expressed as $C_R$*: not*(*navpath*($C_R$, $C_1$) *and navpath*($C_R$, $C_2$) *... and navpath* ($C_R$,$C_n$)), where $C_R$ is the common root of $C_1$, $C_2$, ..., $C_n$ in $DD_{DL}$.

**Example 7 (IDEUnit incompatible with MB2):** *Computer: not*((*slot-constraint hdunit-of-computer has-value IDEUnit*) *and* (*slot-constraint mb-of-computer has-value MB2*)) □

**Requirements.** A *requires* dependency between concepts $C_1$ and $C_2$ ($C_1$ requires $C_2$) can be expressed as $C_R$*: not*(*navpath*($C_R$, $C_1$)) *or navpath*($C_R$, $C_2$) with the common root $C_R$ of $C_1$, $C_2$ in $DD_{DL}$. Similar to incompatibilities, requirements can be extended by introducing further navigation paths on the LHS and RHS of a requires dependency, e.g. $C_1 \wedge C_3$ *requires* $C_2 \vee C_4$.

**Example 8 (SCSIUnit requires MB1):** *Computer:*((*not*(*slot-constraint hdunit-of-computer has-value SCSIUnit*)) *or* (*slot-constraint mb-of-computer has-value MB1*)) □

**Resource balancing.** Resource constraints can be formulated using aggregation functions as proposed in [2]. We assume the existence of a value domain $dom(D)$, a set of predicate symbols $P_i$ associated with binary relations (typically $<, \leq, =, \geq, >$) over $dom(D)$, and a set of aggregation functions $agg(D)$ (typically $sum$, $avg$, $count$, $id$; the last being *identity* in the case where we want to compare to a single fixed value instead of to an aggregate). Concerning the path leading to the concept whose feature values are aggregated, the definition of [2] requires that all but the last one of the roles in this path must be features.

Let <$r_1^c$, $r_2^c$, ..., $r_{n-1}^c$, $r_n^c$> and <$r_1^p$, $r_2^p$, ..., $r_{m-1}^p$, $r_m^p$> be navigation expressions with $C_R$ as common root leading to the consumer (producer) concepts $C_c$ ($C_p$). Furthermore, let $f_c$, $f_p$ be features of $C_c$ and $C_p$ and $\Sigma \in agg(D)$ be an aggregation function. We can express a resource constraint as $C_R$: $P(r_1^c\ r_2^c$ ... $r_{n-1}^c\ \Sigma(r_n^c \circ f_c), r_1^p\ r_2^p$ ... $r_{m-1}^p\ \Sigma(r_m^p \circ f_p))$ according to [2].

Generally, if two aggregates are compared, one interprets the set that has to produce a smaller value as the set of *consumers*, and the set that has to produce a larger value as the set of *producers*.

Now we can define a concept *Computer* that comes equipped with a set of *HDUnits* that provide a corresponding hard-disk capacity and a set of software components that need hard-disk capacity. We express the requirement that the total hard-disk capacity consumed by software cannot exceed the installed hard-disk capacity. In this case navigation expressions only consist of two elements, consequently *hdcapacity* must be a feature, while *hdunit-of-computer* is a general role. The last line is the actual resource constraint.

**Example 9 ($\Sigma$ swcapacity $\leq \Sigma$ hdcapacity):**
*Software: slot-def swcapacity range* (*min 0*) *properties functional*
*HDUnit: slot-def hdcapacity range* (*min 0*) *properties functional*
*Computer: slot-def hdunit-of-computer range HDUnit*
*slot-def sw-of-computer range Software*
*lesseq* (*sum*(*sw-of-computer*∘*swcapacity*),
*sum*(*hdunit-of-computer*∘*hdcapacity*)) □

**Analysis.** While the basic frame structure and formal basis of description logics based languages makes them one of the natural choice for configuration representation, certain demands on expressiveness must be met. The current versions of OIL and DAML+OIL do not support aggregation functions which are fundamental representation concepts frequently used in the configuration domain. Sattler and Baader [2] 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. Since trivial structural conditions lead to definitional overhead (e.g., when defining restrictions on port connections), additional concepts must be provided allowing the definition of roles with arity greater than two; the description of more complex structural properties (see [6]) would be supported by permitting the usage of variables. As far as the definition of rule languages for expressing detailed configuration knowledge on top of DAML+OIL is concerned, we must stress that such a language must allow disjuncts of positive literals when writing the constraint in clausal form, e.g., to express alternative port connections.

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 w.r.t. to decidability of basic properties such as satisfiability or subsumption [2]. 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 catalogues are typically considered unchangeable), which reduces the importance of subsumption versus that of A-Box reasoning.

## 5 Conclusions

In this paper we have shown how to apply Semantic Web ontology representation languages for configuration knowledge representation and integration. We gave a description logic based definition of a configuration problem and showed its equivalence with corresponding consistency-based definitions. A consequence of this equivalence is that configuration problems represented in description logics can easily be transformed into configuration problems represented in predicate logic or simplified variants thereof (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 which must be fulfilled in order to use such a language for full-fledged configuration knowledge representation. Thus DAML+OIL has the opportunity for being a standard knowledge representation language for the configuration domain.

## References


[1] A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-Whole Relations in Object-Centered Systems: An Overview. *Data & Knowledge Engineering*, 20(3):347–383, 1996.

[2] F. Baader and U. Sattler. Description Logics with Concrete Domains and Aggregation. In *Proceedings of the* $13^{th}$ *European Conference on Artificial Intelligence (ECAI '98)*, pages 336–340, Brighton, UK, 1998.

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

[4] T. Berners-Lee. *Weaving the Web*. Harper Business, 2000.

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

[6] J. Cai, M. Furer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *Proceedings of* $30^{th}$ *IEEE Symposium on FOCS*, pages 612–617, 1989.

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

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

[9] D. Fensel, F. vanHarmelen, I. Horrocks, D. McGuinness, and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[10] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.

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

[12] E.W. Jüngst M. Heinrich. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings of the* $7^{th}$ *IEEE Conference on AI applciations (CAIA)*, pages 257–264, Miami, FL, USA, 1991.

[13] D. Mailharro. A classification and constraint-based framework for configuration. *AI Engineering Design Analysis and Manufacturing Journal, Special Issue: Configuration Design*, 12(4):383–397, 1998.

[14] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 90)*, pages 25–32, Boston, MA, 1990.

[15] S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings* $11^{th}$ *International Joint Conf. on Artificial Intelligence*, pages 1395–1401, Detroit, MI, 1989.

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

[17] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a General Ontology of Configuration. *AI Engineering Design Analysis and Manufacturing Journal, Special Issue: Configuration Design*, 12(4):357–372, 1998.

[18] F. vanHarmelen, P.F. Patel-Schneider, and I. Horrocks. A Model-Theoretic Semantics for DAML+OIL. *www.daml.org*, March 2001.

[19] J.R. Wright, E. Weixelbaum, G.T. Vesonder, K.E. Brown, S.R. Palmer, J.I. Berman, and H.H. Moore. A Knowledge-Based Configurator that supports Sales, Engineering, and Manufacturing at AT&T Network Systems. *AI Magazine*, 14(3):69–80, 1993.