# Analyzing configuration systems with description logics: A case study

Ralf Möller Carsten Schröder
Carsten Lutz
University of Hamburg, Computer Science Department,
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany,
{moeller,schroeder,lutz}@kogs.informatik.uni-hamburg.de

November 14, 1997

Corresponding author: Ralf Möller,
Tel: ++49 40 5494 2571
Fax: ++49 40 5494 2572

# Analyzing configuration systems with description logics: A case study

**Ralf Möller, Carsten Schröder and Carsten Lutz**
University of Hamburg, Computer Science Department,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{moeller,schroeder,lutz}@kogs.informatik.uni-hamburg.de

**Abstract**

*In this contribution we argue that the methods of formal knowledge representation, especially description logics, first, are valuable tools for analyzing existing configuration systems and open problems in configuration systems research, second, should be used by developers in order to make clear statements about the performance of their systems, and third, can even directly be used for building practical systems. As a case study we analyze two languages dealing with knowledge-based configuration in technical domains. The paper demonstrates that specialized languages (e.g. object-oriented languages developed for configuration problems) can be interpreted as a special purpose description logic. It is demonstrated that the construction or configuration process can be "simulated" by a model-constructing satisfiability prover. The constructed logical model represents the artifact to be designed. We also show a methodology for describing the meaning of specialized languages by applying syntactical transformations from language constructs to description logic formulae. Considering these transformations, a language designer can easily estimate the computational costs of intended constructs.*

**Keywords**— Methods for configuration, Configuration representations, Description logics.

## 1 Introduction

In this contribution we argue that the methods of formal knowledge representation, especially description logics, first, are valuable tools for analyzing existing configuration systems and open problems in configuration systems research, second, should be used by developers in order to make clear statements about the performance of their systems, and third, can even directly be used for building practical systems. As a case study we analyze two languages dealing with knowledge-based configuration in technical domains. The systems are called PLAKON and KONWERK and have been developed at the Dept. of Computer Science, University of Hamburg as part of a research project to develop a knowledge-based methodology for solving configuration problems (see [Neumann 1988, Cunis et al. 1991, Günter 1995b] for an overview and [Kopisch & Günter 1992] for an application example). The representation language of the PLAKON and KONWERK systems is called BHIBS and is used to define and structure a configuration space for a specific domain. Strategies for actually finding a solution in the configuration space can be defined in a domain-dependent way.

In the following we present a methodology for deriving a (partial) logical reconstruction of configuration languages such as PLAKON and KONWERK. Section 2 introduces the view that the methodology used in configuration systems for solving configuration tasks can be seen as a special instance of a satisfiability test for terms of a logical language. Section 3 shows how most of the concept definitions and some of the constraint definitions of the BHIBS representation language can be transformed to terminological axioms of a description logic. This section also explains the consequences with respect to estimating the complexity of a configuration process. The presentation is complementary to [Baader et al. 1996] who develop a specialized description logic for solving specific configuration problems rather than a general language for building configuration system development environments [Günter 1995b]. Section 4 discusses how the use of formal methods based on description logic helps in understanding open problems of the configuration domain.

## 2 The Configuration Methodology

As part of our case study we give an introduction to the configuration language BHIBS and to the way the configuration space is defined in PLAKON and KONWERK. After dicussing how a given configuration task is solved in these systems by repeated application of four basic configuration steps we give a formal interpretation of the configuration process in terms of a description logic and a specific method for testing satisfiability of description logic terms based on a semantics given for BHIBS expressions.

### 2.1 Defining the Configuration Space

Similar to other many other configuration systems (e.g. those based on CLASSIC [Borgida et al. 1989, McGuiness & Resnick 1995, McGuiness et al. 1995]), PLAKON's as well as KONWERK's approach to configuration of technical devices is a model-based one. The main idea of the configuration methodology of both systems is to use a conceptual model to describe the space of possible configurations of the devices in a certain domain. For defining the conceptual domain model, a frame-based language is used. A configuration task is given as a goal object (defined by instantiating a certain concept of the domain model) and optionally a set of additional objects (components) which must be part of the goal object in the final configuration. The construction process of PLAKON and KONWERK proceeds by applying the following four basic *configuration steps* until the goal object is completely specified (operational semantics).

1. Determine a slot (or parameter) value for a construction object (either a concrete value or a value restriction).

2. Specialize a construction object by asserting (i.e. *hypothesizing*) that it is an instance of one of the explicitly given subconcepts of its current concept.

3. Aggregate a set of construction objects, i.e. create a new object by instantiating the concept the construction objects to be aggregated must be parts of, or add an object to an existing aggregate.

4. Decompose a construction object and configure the parts.

Obviously, more than one step might be applicable in a certain state during the configuration process and, in turn, with each step different possibilities are available. For instance, there might exist several ways to decompose an object into its parts. PLAKON and KONWERK provide an explicit control module to structure the configuration search space (as usual, applying a construction step is called a "heuristic decision"). The control module can be adapted to the problem using an explicit model with "strategies" for traversing the construction space (see [Günter & Cunis 1992]). The construction of the goal object, i.e. the configuration of the required device, is finished either if none of the four construction steps can be applied any more, or if there does not exist a consistent solution. Thus, each of the construction objects contained in the solution is specialized as much as possible, i.e., it is an instance of a leaf node of the concept hierarchy given by the domain model, and all required parts and parameter values of each of the construction objects are determined.

In the following section we will present a logical interpretation of this process.

## 2.2 Using description logics for configuration problems

### 2.2.1 The Abstract Domain

In a description logic (DL) a factual world consists of named individuals and their relationships that are asserted through binary relations. Descriptions for different sets of individuals form the terminological knowledge. Descriptions (or terms) about sets of individuals are called *concepts* and binary relations are called *roles*. Descriptions consist of identifiers denoting concepts, roles, and individuals, and of description constructors. Concepts or roles may be either *primitive* or *defined*. A specification of a primitive concept is denoted with the declaration operator '$\sqsubseteq$' and represents membership conditions that are necessary but *not* sufficient. The specification of a defined concept is denoted by '$\doteq$' and represents conditions that are both necessary *and* sufficient. For any individual $x$ the set $\{y|\mathsf{r}(x, y)\}$ is called the set of *fillers* of role $\mathsf{r}$ for the individual $x$.

Concept specifications may consist of concept terms and concept names. Unary operators (e.g. $\neg$) are used as modifiers and binary operators (e.g. $\sqcap$, $\sqcup$) are used as connectives. A concept term can also be given as a restriction for role fillers. *Value restrictions* constrain the range of roles, i.e. fillers are constrained to be individuals of a specific concept (e.g. ($\forall$ $\mathsf{has\_father}$ $\mathsf{male}$)). *Number restrictions* specify the maximum or minimum number of allowed fillers (e.g. ($\leq$ 3 $\mathsf{has\_child}$), ($\geq$ 1 $\mathsf{has\_father}$)). Roles with an implicit '$\leq$1' number restriction are called *attributes* or *features*. These concept specifications are only a subset of all possible specifications. Section 2.2.3 lists the model-theoretic semantics of DL elements mentioned in this paper. The reasoning services a DL inference engine provides are defined with respect to the semantics of the representation language. In most description logics, the terminology must not contain cyclic definitions because the semantics of cycles cause theoretical and practical difficulties. Furthermore, on the left-hand side of a definition only concept names may apprear and a concept name must occur only once on the left-hand side in the definitions of a terminology. The expressiveness of a DL and the tractability of reasoning algorithms for a particular DL depends on the type and possible combinations of connectives and restrictions (see e.g. [Woods & Schmolze 1992]).

DL systems (i.e. implementations of a DL) usually distinguish two separate reasoning components. The *terminological reasoner* or *classifier* classifies concepts with respect to subsumption relationships between these concepts and organizes them into a taxonomy. The TBox language is designed to facilitate the construction of concept expressions describing classes (types) of individuals. The classifier automatically performs consistency checking of concept definitions and offers retrieval facilities about the classification hierarchy. The (forward-chaining) *assertional reasoner* or *realizer* recognizes and maintains the "type" (i.e. concept membership) of individuals. The purpose of the ABox language is to state constraints or facts (usually restricted to unary or binary predications) that apply to a particular world model. Assertional reasoners support a query language in order to access stated and deduced constraints. Some query languages offer the expressiveness of the full first-order calculus.

### 2.2.2 The Concrete Domain

Baader and Hanschke [Baader & Hanschke 1991, Hanschke 1993] have explored the idea of separating the domain of a description logic into an *abstract* and a *concrete* part. A DL with a concrete domain extends a standard DL by adding predicates and individuals for the concrete domain. The predicates can be used to define new concepts in the abstract domain. Attribute fillers can be restricted by predicates of the concrete domain (see below).

Basically, (i) the set of predicate names defined by a concrete domain has to be closed under negation and has to contain a predicate name for domain membership, and (ii) the satisfiability problem for finite conjunctions of corresponding predicates has to be decidable (see [Hanschke 1993] for a detailed definition).

Another approach for integrating other domains into description logics has been proposed by [Borgida et al. 1996]. This approach is specific to the description logic CLASSIC [Borgida et al. 1989] which—as we will see in the following sections—is not expressive enough to model the facilities offered by the configuration systems Plakon and Konwerk considered in this contribution.

### 2.2.3 Semantics of DL Elements

Let $\mathcal{C}$ be the set of concepts, $\mathcal{R}$ the set of roles, and $\mathcal{P}$ the set of concrete predicates in a DL theory. The model-theoretic semantics of a DL is based on the notion of an *interpretation* which is defined as a pair $\langle \mathcal{D}, \xi \rangle$ where the domain $\mathcal{D}$ is subdivided into two subsets, the abstract and the concrete objects (e.g. real numbers), $\mathcal{D} = \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{C}, \mathcal{D}_\mathcal{A} \cap \mathcal{D}_\mathcal{C} = \emptyset$. Furthermore, we have two disjoint sets, the set of concept names $\mathcal{C}$ (atomic concepts) and the set of role names $\mathcal{R}$ (the elements of a subset $\mathcal{F}$ of the roles are defined to be features, i.e. partial functions. $\xi$ is an assignment function such that $\xi : \mathcal{C} \longrightarrow 2^{\mathcal{D}_\mathcal{A}}$, $\xi : \mathcal{R} \longrightarrow 2^{\mathcal{R}'}$ where $\mathcal{R}' = (\mathcal{D}_\mathcal{A} \times (\mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{C}))$. $\xi$ must satisfy the following conditions for mapping syntactical terms to semantical entities (concept names are denoted by $\mathsf{c}$, possibly with index, role names by $\mathsf{r}$, features by $\mathsf{f}$, and concrete predicate names by $\mathsf{p}$). We only list the semantics for the DL elements mentioned in this paper (for a detailed introduction to description logics see [Woods & Schmolze 1992]).

$$
\begin{aligned}
\xi[(\top)] &= \mathcal{D}_\mathcal{A} \\
\xi[(\bot)] &= \emptyset \\
\xi[\mathsf{concept\ name}] &\subseteq \mathcal{D}_\mathcal{A} \\
\xi[\mathsf{feature\ name}] &\subseteq \mathcal{D}_\mathcal{A} \times \mathcal{D}_\mathcal{A} \text{ or} \\
\xi[\mathsf{feature\ name}] &\subseteq \mathcal{D}_\mathcal{A} \times \mathcal{D}_\mathcal{C} \\
\xi[\mathsf{role\ name}] &\subseteq \mathcal{D}_\mathcal{A} \times \mathcal{D}_\mathcal{A} \\
\xi[\mathsf{predicate\ name}] &\subseteq \mathcal{D}_\mathcal{C} \\
\xi[\neg\mathsf{c}] &= \mathcal{D}_\mathcal{A} \backslash \xi[\mathsf{c}] \\
\xi[(\mathsf{c}_1 \sqcap \ldots \sqcap \mathsf{c}_n)] &= \cap_{i=1}^{n} \xi[\mathsf{c}_i] \\
\xi[(\mathsf{c}_1 \sqcup \ldots \sqcup \mathsf{c}_n)] &= \cup_{i=1}^{n} \xi[\mathsf{c}_i] \\
\xi[(\geq n\ \mathsf{r}\ \mathsf{c})] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[\mathsf{r}] \Rightarrow y \in \xi[\mathsf{c}]\}\| \geq n\} \\
\xi[(\leq n\ \mathsf{r}\ \mathsf{c})] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[\mathsf{r}] \Rightarrow y \in \xi[\mathsf{c}]\}\| \leq n\} \\
\xi[(\exists\ \mathsf{f}\ \mathsf{p})] &= \{x |\ \exists y : (x,y) \in \xi[\mathsf{f}] \Rightarrow y \in \xi[\mathsf{p}]\}
\end{aligned}
$$

Note that other known operators can be seen as syntactic abbreviations for special qualifying number restrictions (we also use the operator $=$ for a conjunction of the respective formulas with $\leq$ and $\geq$):

$$\begin{aligned}(\forall\ \mathsf{r}\ \mathsf{c}) &:= (\leq 0\ \mathsf{r}\ \neg c),\\ (\exists\ \mathsf{r}\ \mathsf{c}) &:= (\geq 1\ \mathsf{r}\ c),\\ (\geq\ n\ \mathsf{r}) &:= (\geq n\ \mathsf{r}\ \top),\\ (\leq\ n\ \mathsf{r}) &:= (\leq n\ \mathsf{r}\ \top)\end{aligned}$$

In the TBox the two special symbols '$\doteq$' and '$\sqsubseteq$' are used for introducing defined and primitive concepts, respectively. The definitions are mapped onto set-inclusion axioms.

- $\mathsf{Cname} \doteq \mathsf{C}$ is mapped onto $\xi[\mathsf{Cname}] = \xi[\mathsf{C}]$
- $\mathsf{Cname} \sqsubseteq \mathsf{C}$ is mapped onto $\xi[\mathsf{Cname}] \subseteq \xi[\mathsf{C}]$

The semantics of ABox assertions is defined analogously:

- $\mathsf{Iname} : \mathsf{C}$ is mapped onto $\xi[\mathsf{Iname}] \in \xi[\mathsf{C}]$
- $(\mathsf{Iname}_1, \mathsf{Iname}_2) : \mathsf{Rname}$ is mapped onto $(\xi[\mathsf{Iname}_1], \xi[\mathsf{Iname}_2]) \in \xi[\mathsf{Rname}]$

An interpretation that satisfies all axioms in a terminology is called a *model*. The notion of a model is used to define the reasoning services a DL inference engine has to provide: subsumption and consistency checking which are closely related. A term $\mathsf{A}$ *subsumes* another term $\mathsf{B}$ if and only if for every model $\langle \mathcal{D}, \xi \rangle$ $\xi[\mathsf{B}] \subseteq \xi[\mathsf{A}]$ holds. A term $\mathsf{A}$ is *coherent* if and only if there exists a model $\langle \mathcal{D}, \xi \rangle$ such that $\xi[\mathsf{A}] \neq \emptyset$.

### 2.2.4 A Formal Interpretation of the Configuration Process

One of the first formal approaches to configuration problems was given by Owsnicki-Klewe [1988]. He used the terminological language of a KL-ONE-like description logic for defining a domain model and the corresponding assertional language for specifying the device to be configured (the goal object). Given a knowledge base of his logic he then used the object classification service (i.e. *realization*) provided by description logics for computing the most special concepts of the objects given in the specification. These concepts were defined to be the solution of the configuration problem: They provide a description of all the properties of the given objects. However, this process only generates interesting solutions, if the concepts of the domain model are properly *defined* by giving necessary as well as sufficient conditions, for object classification is a purely deductive process. If only necessary conditions are given, no new information can be generated (except the detection of inconsistent specifications, of course). In addition, note that no new objects are generated by this process. Neither does it aggregate objects to a new one nor does it construct the required parts of an object. It is quite obvious that this formal approach to configuration does not explain the approach taken by PLAKON and KONWERK: although deductive reasoning is clearly needed, hypothetical reasoning[1] is needed as well.

However, the methodology used in our case study Plakon and Konwerk as well as in other configuration system for generating solutions of a configuration task described above can be seen to be a special instance of the model construction approach of Buchheit et al. [1995] tailored to the peculiarities of the respective representation language (see also the work of Baader et al. [1996]). Following this approach, a solution of a configuration task is defined to be a *logical model* of the given knowledge base containing both the conceptual domain model as well as the task specification. A logical model consists (i) of a set of objects (the *domain of discourse*), (ii) of an interpretation function which maps object names to the objects of the domain of discourse and concepts and slot names to unary and binary relations, respectively, and (iii) it is required to satisfy the formulas of the given knowledge base. A more thorough analysis reveals that the set of objects and relations represented by slots which are usually constructed by configuration processes is a representation of a logical model of itself as well as the domain model containing the concept descriptions. This model maps each object to itself, each concept to the set of instances of this concept contained in the constructed configuration and each slot to the set of object/filler tuples.

Although the classification services usually provided by description logics are not the central mechanisms needed for configuration, the tableau calculi which became popular for realizing these services can be directly used as a basis for configuration systems [Buchheit et al. 1995]. The algorithm for the satisfiability test provided by these calculi tries to *contruct a logical model of the given knowledge base.* When a logical model can be constructed, a knowledge base is satisfiable. Therefore, extended by suitable control mechanisms, tableau calculus algorithms can be used for emulating configuration techniques such as those found in our case study Plakon and Konwerk. Note, however, that the language proposed by Buchheit et al. [1995] which is based on a feature logic is not suitable for the configuration domain. One of its central notions, the *part-whole* relation, cannot be represented using functional roles (i.e. features).[2]

Using our example systems Plakon and Konwerk, the next chapter explains the main idea of describing specialized configuration systems with description logic theory.

## 3 The Language

As explained before, Plakon provides a language called Bhibs which can be used for modeling a domain by defining concepts [Cunis et al. 1989]. In this section we present the main ideas behind Bhibs and illustrate how the language constructs can be transformed to description logics. We would like to emphasize

[1]We hesitate to call it *abductive* reasoning, because configuration is not a task of generating explanations.

[2]Many reports on whole-part relations have been published. In particular, from a description logic point of view, see e.g. [Artale et al. 1996, Lutz 1996, Sattler 1995].

that Bhibs is used as an example here. For other object-oriented (configuration) languages known today, similar reconstructions can be provided.

## 3.1 Concept Descriptions

Bhibs is a frame language using single inheritance which allows one to describe the properties of instances by specifying restrictions for the required values of named slots. The values can be either single objects or sets and sequences of objects, and the restrictions can be specified extensionally by directly giving concrete values like numbers, symbols or instances of concepts, or by intensionally describing sets and sequences of objects. The following example of an expression of the Bhibs-language describes the concept of a cylinder:

```
(is! (a Cylinder)
     (a Motorpart
        (part-of (a Motor))
        (capacity [1ccm 1000ccm])
        (has-parts
           (:set #[(a Cylinderpart)   4 6] :=
                 #[(a Piston)         1 1]
                 #[(a Connecting-Rod) 1 1]
                 #[(a Valve)          2 4]))))
```

A Cylinder is required to be a Motorpart, to be part-of a Motor, to have a capacity of 1 to 1000ccm, and to have a set of 4 to 6 parts (has-parts) which are all Cylinderparts and it consists of exactly 1 Piston, exactly 1 Connecting-Rod, and 2 to 4 Valves. This expression can be transformed to a terminological inclusion axiom of a description logic providing concrete domains as follows (the term $\lambda_{\mathrm{Vol}}\, c. (\ldots)$ is a unary predicate of a numeric concrete domain for the dimension *Volume* with base unit $m^3$):

$$
\begin{aligned}
\mathsf{Cylinder} \sqsubseteq\ & \mathsf{Motorpart} \sqcap \\
& (= 1\ \mathsf{part\text{-}of}) \sqcap (\forall\, \mathsf{part\text{-}of}\ \mathsf{Motor}) \sqcap \\
& (= 1\ \mathsf{capacity}) \sqcap \\
& (\exists\, \mathsf{capacity}\ \lambda_{\mathrm{Vol}}\, c. (0.001 \le c \wedge c \le 1)) \sqcap \\
& (\forall\, \mathsf{has\text{-}parts}\ (\mathsf{Cylinderpart} \sqcap (\mathsf{Piston} \sqcup \mathsf{Connecting\text{-}Rod} \sqcup \mathsf{Valve}))) \sqcap \\
& (\ge 4\ \mathsf{has\text{-}parts}\ \mathsf{Cylinderpart}) \sqcap \\
& (\le 6\ \mathsf{has\text{-}parts}\ \mathsf{Cylinderpart}) \sqcap \\
& (= 1\ \mathsf{has\text{-}parts}\ \mathsf{Piston}) \sqcap \\
& (= 1\ \mathsf{has\text{-}parts}\ \mathsf{Connecting\text{-}Rod}) \sqcap \\
& (\ge 2\ \mathsf{has\text{-}parts}\ \mathsf{Valve}) \sqcap \\
& (\le 4\ \mathsf{has\text{-}parts}\ \mathsf{Valve})
\end{aligned}
$$

Note that the given restrictions are only necessary conditions for a Cylinder. This was not at all clear on first sight, but was deduced from the procedural semantics of BHIBS defined by the system PLAKON. This example shows the importance of defining a logical semantics for a configuration system.

In an effort to provide a formal declarative semantics for BHIBS we found that all concept definitions except those containing sequence descriptions can be transformed to terminological inclusion axioms. Figure 1 specifies a set of transformation rules. Read the functions TTA and TSD as *Transform TBox Axiom* and *Transform Slot Description*, respectively. A *Measure* is a number either with or without a unit for a specific dimension, e.g. `42` or `25km`. The function DIM returns the dimension of a "measure", e.g. *Vol* for `1000ccm`, and the function VALUE returns the value of a given "measure", 1000 in this example.

| | | |
|---|---|---|
| TTA((is! (a *ConceptName*)<br>(a *SuperConceptName*<br>*SlotDescription1*<br>*SlotDescription2*<br>...)) | $\rightarrow$ | $\mathit{ConceptName} \sqsubseteq \mathit{SuperConceptName} \sqcap$<br>$\textsc{tsd}(\mathit{SlotDescription1}) \sqcap$<br>$\textsc{tsd}(\mathit{SlotDescription2}) \sqcap$<br>$\ldots$ |
| TTA((def-relation :name *SlotName1*<br>:inverse *SlotName2*)) | $\rightarrow$ | $\mathit{SlotName1} \doteq \mathit{SlotName2}^{-1}$ |
| TSD((*SlotName* (a *ConceptName*))) | $\rightarrow$ | $(= 1\ \mathit{SlotName}) \sqcap$<br>$(\forall\ \mathit{SlotName}\ \mathit{ConceptName})$ |
| TSD((*SlotName*<br>{*ObjectName1* *ObjectName2* ...})) | $\rightarrow$ | $(= 1\ \mathit{SlotName}) \sqcap$<br>$(\forall\ \mathit{SlotName}\ \{\mathit{ObjectName1}\ \mathit{ObjectName2}\ \ldots\})$ |
| TSD((*SlotName* [*Measure1* *Measure2*])) | $\rightarrow$ | $(= 1\ \mathit{SlotName}) \sqcap$<br>$(\exists\ \mathit{SlotName}$<br>$\lambda_{\textsc{dim}(\mathit{Measure1})}\ x.$<br>$(\textsc{val}(\mathit{Measure1}) \leq x \wedge x \leq \textsc{val}(\mathit{Measure2})))$ |
| TSD((*SlotName*<br>(:some (a *ConceptName*) $m$ $n$))) | $\rightarrow$ | $(\geq m\ \mathit{SlotName}\ \mathit{ConceptName}) \sqcap$<br>$(\leq n\ \mathit{SlotName}\ \mathit{ConceptName})$ |
| TSD((*SlotName*<br>(:set (:some (a *ConceptName1*) $m_1$ $n_1$) :><br>(:some (a *ConceptName2*) $m_2$ $n_2$)<br>(:some (a *ConceptName3*) $m_3$ $n_3$)<br>...))) | $\rightarrow$ | $(\forall\ \mathit{SlotName}\ \mathit{ConceptName1}) \sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName1*) $m_1$ $n_1$))) $\sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName2*) $m_2$ $n_2$))) $\sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName3*) $m_3$ $n_3$))) $\sqcap$<br>$\ldots$ |
| TSD((*SlotName*<br>(:set (:some (a *ConceptName1*) $m_1$ $n_1$) :=<br>(:some (a *ConceptName2*) $m_2$ $n_2$)<br>(:some (a *ConceptName3*) $m_3$ $n_3$)<br>...))) | $\rightarrow$ | $\forall\ \mathit{SlotName}\,.$<br>$(\mathit{ConceptName1} \sqcap$<br>$(\mathit{ConceptName2} \sqcup \mathit{ConceptName3} \sqcup \ldots)) \sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName1*) $m_1$ $n_1$))) $\sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName2*) $m_2$ $n_2$))) $\sqcap$<br>TSD((*SlotName*<br>(:some (a *ConceptName3*) $m_3$ $n_3$))) $\sqcap$<br>$\ldots$ |

Figure 1: Rules for transforming a BHIBS terminology.

There are several things to point out in this transformation. First, we are using more than one concrete domain—one for each dimension—although all of them are numeric. This helps in seperating the dimensions from each other, they can be handled independently. Second, what we have called a *SlotDescription* (in accordance with one of the developers of the system KONWERK) is transformed to a concept term of a description logic, for it intensionally describes a set of objects of the domain. Third, in PLAKON as well as in KONWERK the slots of an object are assumed to have only one filler. This might be either a single object (a number, a symbol, or an instance of a concept) or a set of objects. We transform slots to *roles* of a description logic which may have more than one filler. Slots are not transformed to *features* (i.e. roles with an implicit "at least one" number restriction). Therefore, objects having a set of objects as a slot filler are seen as objects having multiple fillers of a *role* in our transformation, so, in transformed expressions, there is no reification of a set of objects.

After transforming a BHIBS knowledge base by applying the rules shown in Figure 1 some additional axioms must be added in order to retain the intended meaning. In PLAKON as well as in KONWERK, the domain model given by a knowledge base is assumed to be *complete* in the sense that all the different types of objects (i.e. concepts) are known and explicitly given (see [Cunis et al. 1991, Günter 1995b]). Therefore, concepts are assumed to be completely covered by its direct subconcepts, and the direct subconcepts are assumed to be pairwise disjoint. In both systems these assumptions manifest themselves in configuration step 2 shown in Section 2.1. Objects are specialized to a leaf node of the concept hierarchy. In our transformation these assumptions must be made explicit by adding a number of cover and disjointness axioms (see also Buchheit et al. [1995]). If, for example, a concept $\mathsf{C0}$ has the direct subconcepts $\mathsf{C1}$, $\mathsf{C2}$, and $\mathsf{C3}$, then the following axioms must be added to the TBox.

$$\mathsf{C0} \sqsubseteq \mathsf{C1} \sqcup \mathsf{C2} \sqcup \mathsf{C3}$$
$$\mathsf{C1} \sqsubseteq \neg\mathsf{C2} \qquad \mathsf{C1} \sqsubseteq \neg\mathsf{C3} \qquad \mathsf{C2} \sqsubseteq \neg\mathsf{C3}$$

After adding cover and disjointness axioms, "specialization to leaf concepts" is done by a model construction process as well.

The basic PLAKON and KONWERK systems support only incomplete reasoning services for checking the domain model. Let us assume the following declarations impose restrictions on $\mathsf{B}$, $\mathsf{C}$ and $\mathsf{D}$.

$$\begin{aligned}
\mathsf{A} &\sqsubseteq (\geq 10\ \mathsf{r}) \sqcap (\leq 60\ \mathsf{r})\\
\mathsf{B} &\sqsubseteq \mathsf{A} \sqcap (\geq 15\ \mathsf{r}) \sqcap (\leq 20\ \mathsf{r})\\
\mathsf{C} &\sqsubseteq \mathsf{A} \sqcap (\geq 20\ \mathsf{r}) \sqcap (\leq 30\ \mathsf{r})\\
\mathsf{D} &\sqsubseteq \mathsf{A} \sqcap (\geq 30\ \mathsf{r}) \sqcap (\leq 50\ \mathsf{r})
\end{aligned}$$

The generated cover axiom $\mathsf{A} \sqsubseteq \mathsf{B} \sqcup \mathsf{C} \sqcup \mathsf{D}$ imposes the following additional restrictions on $\mathsf{A}$:

$$\mathsf{A} \sqsubseteq (\geq 15\ \mathsf{r}) \sqcap (\leq 50\ \mathsf{r})$$

Thus, there is more to TBox reasoning than only consistency checking. The KONWERK system tries to support these inferences with an extension module called TAX [Günter 1995a]. The main idea of using TAX is to reduce the search space for constructing objects. For instance, if a construction object is specialized to an $\mathsf{A}$, it will be known in advance that there is no need to try whether e.g. only ten role fillers for $\mathsf{r}$ are sufficient for an $\mathsf{A}$.

In our reconstruction of PLAKON and KONWERK using description logics with the model construction view of realizing the satisfiability test we used the following language constructs:

- conjunction,
- negation and disjunction with atomic concepts,
- value restrictions,
- qualifying number restrictions (see [Hollunder & Baader 1991]),
- inverse roles (e.g. for `has-parts`),
- `one-of` (or sets [Schaerf 1994], see the fourth translation rule in Figure 1),
- concrete domains over $\Re$ (see [Baader & Hanschke 1991]).

It should be noted that due to the special form of the BHIBS syntax, the description logic formulas are not arbitrarily nested, i.e. in principle we use a limited kind of description logic. Considering the formal semantics for BHIBS we defined in this paper, it is obvious that reasoning would be incomplete if the TAX module were not loaded into the KONWERK system. In a specific configuration system that is implemented with BHIBS, this kind of incompleteness would result in a larger configuration space to be explored (possibly searching for solutions where no solutions can be found). The list of construct required for the BHIBS semantics (see above) reveals that the complexity of the satisfiability test will at least be exponential (see e.g. [Baader et al. 1996]). Severe problems concerning decidability can be expected by the circularity of TBox axioms introduced by the implicit cover axioms (unrestricted terminological axioms with generalized concept inclusions (GCIs), see also the language $\mathcal{ALCQ}$ with cardinality restrictions on concepts [Baader et al. 1996]). As Baader [1991] has shown, with respect to the greatest fixpoint semantics the concept defining facilities of a language with cycles are also available in a language with transitive closure of roles. This relationship is important because in another contribution Baader & Hanschke [1991] have shown that in languages with concrete domains (e.g. $\mathcal{ALC}(\mathcal{D})$) the introduction of transitive roles leads to undecidability of the satisfiability problem (i.e. $\mathcal{ALC}(\mathcal{D})$ with transitive closure of roles is undecidable). Although we do not have a formal proof, it is extremely likely that Baader's PCP reduction proof technique can also be applied with the constructs offered by the BHIBS language. Thus, checking a BHIBS concept for consistency with a given (cyclic) TBox might be undecidable, i.e. the configuration process might not terminate.

The analysis of the Bhibs language that we have presented as a case study in this section reveals the necessity of a formal analysis of the semantics of a specific language for defining the configuration space. In the following we will discuss some of the additional knowledge modeling features found in configuration systems.

## 3.2 Mixins and Views

Plakon's and Konwerk's concept languages are restricted to single inheritance. The restriction to single inheritance can easily be understood when Plakon's and Konwerk's technique used for generating solutions of a configuration task is seen from the operational semantics point of view (see Section 2.1). If multiple inheritance were used, construction step 2 would not be sufficient to traverse the configuration space. When a concept is specialized to a certain subconcept with multiple predecessors it must also be specialized to subconcepts of these superconcepts, i.e., in general, there would be no single leaf concept to describe a configuration object. Furthermore, since in the basic Bhibs inheritance scheme the subconcepts of a concept are defined to be pairwise disjoint (see the semantics of Bhibs), declaring two concepts A and B as a superconcept for a concept C would result in an inconsistency (we assume that, implictly, every concept is a subconcept of the central root concept Domain-Object).

However, single inheritance causes modeling redundancy in many domains. In order to provide a more flexible modeling language, Hotz & Vietze [1995a] extended the concept language of Konwerk by introducing the notion of *mixins* (see Figure 2 for an example). Mixins are not instantiated but they provide a restricted form of multiple inheritance and can be seen as macro definitions. The restrictions defined for a mixin are inserted where the name of a mixin appears in a concept definition.

The control mechanism of Konwerk does not attempt to specialize objects to any subconcept of a mixin because mixins are expanded like macros. In the description logic translation, mixins can be transformed to terminological axioms as well, but in contrast to normal concepts no cover and disjointness axioms are created for subconcepts of a mixin. Mixins are translated to terminological inclusion axioms.

To support the knowledge acquisition phase, Plakon suggests the notion of a *view*. The main idea of using views is to provide a structured way to use multiple inheritance while preserving a domain model skeleton with single inheritance. A view is used to describe aspects of an object that can be separated from other aspects. For instance, the mode of operation of a vehicle (Gas-driven, Diesel-driven, Electricity-driven) can be separated from the medium the vehicle is constructed for (land, water). In Figure 3 the Mode-of-Operation mixin concept tree from Figure 2 is presented as a view.

A view is a separate concept hierarchy with single inheritance that is coupled to the main hierarchy. In Figure 3 the nodes for Mode-of-Operation and Operation-Medium are linked to Vehicle and the concept Motorized is linked to

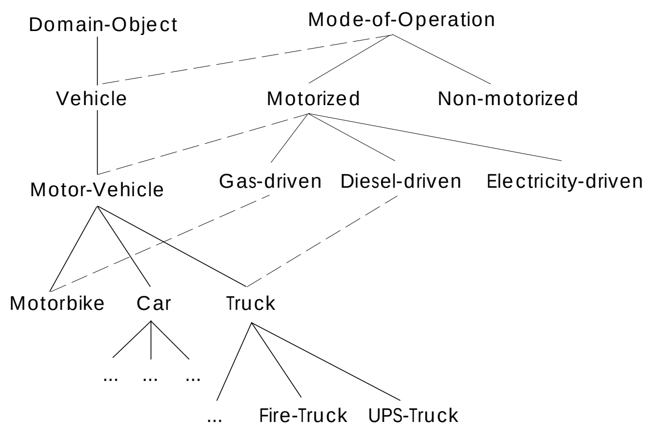

Figure 2: Example for a concept hierarchy with mixins. The main inheritance links (single inheritance) are indicated with normal lines, mixin links are shown with dashed lines.

Motor-Vehicle. In the following we will consider the Mode-of-Operation view only.

The operational semantics of view links as given by Hotz & Vietze is defined as follows. For each main concept C that is linked to a view concept V, two sets are constructed. The first set (C-Set) contains the leaf subconcepts of the main concept C that can be reached by traversing the subclass inheritance hierarchy *without* touching a concept that is also linked to a view concept. The second set (V-Set) contains the leaf concepts that can be found by traversing the view subconcept hierarchy starting from V *without* touching a view that is also linked to a main concept. As conjunctions, the elements of the cross-product C-Set $\times$ V-Set define new subconcepts of C. In Figure 4 the new subconcepts for the main and view hierarchy of Figure 3 are presented: For Vehicle an additional subconcept Non-motorized-Vehicle and for Motor-Vehicle three new subconcepts Gas-driven-Motor-Vehicle, Diesel-driven-Motor-Vehicle and Electricity-driven-Motor-Vehicle. The new concepts are created to avoid multiple inheritance. For each of these new concepts, the view concept of the corresponding cross-product tuple is used as a mixin, i.e. the concept definition is expanded like a macro and only a single superconcept remains. In order to avoid a combinatorial explosion, the new concepts are created on demand, i.e. a concept Diesel-driven-Motor-Vehicle is only created when an object is known to be Motor-Vehicle.

The declarative semantics is much simpler. With description logics no restructuring of the inheritance graph is necessary. View links (dotted lines in Figure 4) are treated as ordinary superconcept links. A view concept V connected to a main concept C via a view link is simply included in the concept definition of C as an additional restriction. Similar to the approach presented

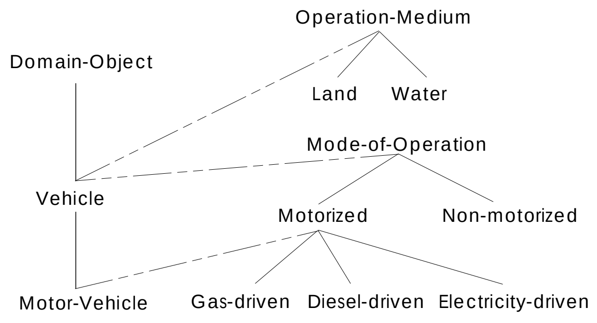

Figure 3: Example for a concept hierarchy with views.The main inheritance links (single inheritance) are indicated with normal lines, view links are shown with dashed lines.

above, for each concept in the view hierarchy cover and disjointness axioms are generated. However, only the *view* subconcepts are combined in a disjunction (or cover) term. For instance, for the main concept Motor-Vehicle and for the view concept Motorized (see Figure 4) the following inclusion axioms are generated:

$$\begin{aligned}
\mathsf{Motor\text{-}Vehicle} &\sqsubseteq \mathsf{Vehicle} \sqcap \mathsf{Motorized} \\
\mathsf{Motorized} &\sqsubseteq \mathsf{Gas\text{-}driven} \sqcup \mathsf{Diesel\text{-}driven} \sqcup \\
&\quad \mathsf{Electricity\text{-}driven} \\
\mathsf{Gas\text{-}driven} &\sqsubseteq \neg\mathsf{Diesel\text{-}driven} \\
\mathsf{Gas\text{-}driven} &\sqsubseteq \neg\mathsf{Electricity\text{-}driven} \\
\mathsf{Diesel\text{-}driven} &\sqsubseteq \neg\mathsf{Electricity\text{-}driven}
\end{aligned}$$

Considering the model construction process of the description logic reasoner, the axioms ensure that a Motor-Vehicle will be either Gas-driven, Diesel-driven or Electricity-driven. Using the facilities of description logics, there is no need to create additional concepts (see the cross-products mentioned above). It becomes clear that mixins and views as defined in BHIBS are important for knowledge engineers but—from a logical point of view—do not enhance the power of the inference systems.

## 3.3 Object Descriptions

During the configuration process, instances are created (see the configuration steps in Section 2.1). These instances are then manipulated by the control system of PLAKON or KONWERK.

In a description logic, assertions about concrete instances are gathered in the so called ABox. The assertional language of a description logic can be used for specifying a device to be constructed in a configuration task as well as for

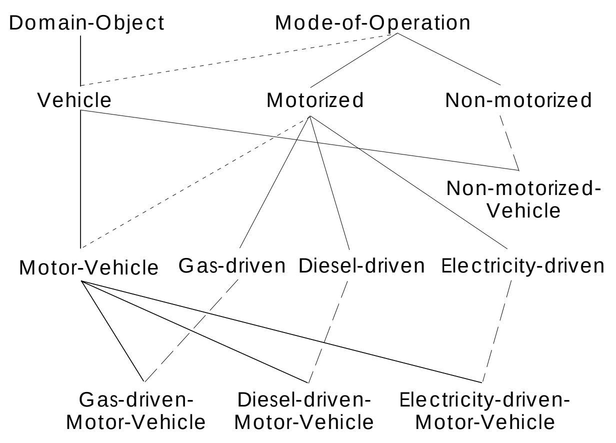

Figure 4: Expanded concept hierarchy.

representing the solutions of the configuration task. The configuration steps mentioned in Section 2.1 generate the following kinds of ABox assertions:

- Creation of instances (construction steps 3 and 4)
- Asserting primitive concepts for instances (construction step 1)
- Asserting concrete fillers for roles (construction step 2)
- Asserting restrictions for role fillers for a specific instance (construction step 2).

In PLAKON and KONWERK there does not exist a simple language for making these assertions. Making assertions about instances is explicitly done by using the functions `slot-value` and `(setf slot-value)` of the underlying implementation language CLOS as well as a number of other functions. For the sake of a simple description we invented a language with a single construct (`set-slot`) and provide a formal declarative symantics for it by showing how it can be transformed to the assertional language of a description logic. Figure 5 specifies the set of transformation rules. Read the function TAA as *Transform ABox Axiom*.

As mentioned earlier, PLAKON's and KONWERK's concept languages are frame languages based on the idea of *slots*. From a logical point of view this has no effect on the interpretation of the languages. It does have an effect on the expressivity of the assertional language, however. If, for example, a Motor-Vehicle and its subconcept Truck (see Figure 2) are not required to have a color, while the subconcept Fire-Truck is required to have the color RED and, for instance, a UPS-Truck is required to have the color BROWN, then in PLAKON as well as KONWERK it is not possible to construct any Motor-Vehicle with

$$
\begin{array}{lcl}
\textsc{taa}(\texttt{(set-slot} & & \\
\qquad \mathit{ObjectName1}\ \ \mathit{SlotName}\ \ \mathit{ObjectName2}\texttt{)}) & \rightarrow & (\mathit{ObjectName1}, \mathit{ObjectName2}) \colon \mathit{SlotName} \\
 & & \\
\textsc{taa}(\texttt{(set-slot} & & \mathit{ObjectName} \colon (\exists\ \mathit{SlotName}\ . \\
\qquad \mathit{ObjectName}\ \ \mathit{SlotName}\ \ \mathit{Measure}\texttt{)}) & \rightarrow & \qquad\qquad \lambda_{\textsc{dim}(\mathit{Measure})}\, x.\, (x = \textsc{val}(\mathit{Measure}))) \\
 & & \\
\textsc{taa}(\texttt{(set-slot} & & \\
\qquad \mathit{ObjectName}\ \ \mathit{SlotName}\ \ \mathit{ObjectDescriptor}\texttt{)}) & \rightarrow & \mathit{ObjectName} \colon \textsc{tsd}(\texttt{(}\mathit{SlotName}\ \ \mathit{ObjectDescriptor}\texttt{)})
\end{array}
$$

Figure 5: Rules for transforming assertions.

color RED other than a Fire-Truck, and worse, when specifying a device to be configured, it is not possible to specify a Motor-Vehicle with color RED. The absence of a color slot must not be confused with the requirement of not having a color, however, for a Fire-Truck clearly is a Motor-Vehicle. The assertional language simply does not allow to express something like this. This anomaly of the language must be taken into account when modeling a domain, and it clearly prevents something like innovative configuration (see Section 4).

This feature of the assertional language of Plakon and Konwerk has an additional effect: Whenever a slot which is defined to be the inverse of another slot is used in a *SlotDescription* of a concept, its inverse must be used in a *SlotDescription* of the concept of the fillers of the slot. In order to provide adequate restrictions for the configuration space, value restrictions must be declared for the corresponding slots. Note that this might result in cyclic concept definitions.

The control system of Plakon or Konwerk can be configured to use different strategies for traversing the configuration space (chronological backtracking, TMS-based construction of a single version of an artifact with knowledge-based backtracking, ATMS-based construction of multiple versions of an artifact). Different strategies can also be implemented for the model construction system for testing satisfiability (see Section 2.2).

### 3.4 Constraints

Plakon's constraint language [Cunis et al. 1991, Chapter 6] can be used to express $n$-ary constraints on the fillers of role chains of objects. These include equality as well as inequality constraints, which in some cases are identical to the well known *role value maps*, as well as numeric constraints.

Role value maps are important for describing has-parts relations. For instance, in the following TBox we define graph structures. A graph consists of vertices and edges which also are set into relation to one another.

$$\begin{aligned}
\mathsf{has\text{-}vertex} &\doteq \mathsf{vertex\text{-}of}^{-1} \\
\mathsf{has\text{-}parts} &\doteq \mathsf{part\text{-}of}^{-1} \\
\mathsf{Graph\text{-}Object} &\sqsubseteq (= 1\ \mathsf{part\text{-}of}) \\
\mathsf{Vertex} &\sqsubseteq \mathsf{Graph\text{-}Object} \\
\mathsf{Edge} &\sqsubseteq \mathsf{Graph\text{-}Object} \sqcap \\
&\quad (\forall\, \mathsf{has\text{-}vertex}\ \mathsf{Vertex}) \sqcap \\
&\quad (= 2\ \mathsf{has\text{-}vertex}) \sqcap \\
&\quad \neg \mathsf{Vertex} \\
\mathsf{Graph\text{-}Object} &\sqsubseteq \mathsf{Vertex} \sqcup \mathsf{Edge} \\
\mathsf{Graph} &\doteq (\forall\, \mathsf{has\text{-}parts}\ \mathsf{Graph\text{-}Object}) \sqcap \\
&\quad ((\mathsf{has\text{-}parts}|_{\mathsf{Edge}} \circ \mathsf{has\text{-}vertex}) = \\
&\quad\ \ \mathsf{has\text{-}parts}|_{\mathsf{Vertex}}) \sqcap \\
&\quad ((\mathsf{has\text{-}parts}|_{\mathsf{Vertex}} \circ \mathsf{vertex\text{-}of}) = \\
&\quad\ \ \mathsf{has\text{-}parts}|_{\mathsf{Edge}})
\end{aligned}$$

Role value maps are required to ensure that if an edge is part of a graph, then the vertices that are set into relation to an edge are part of the same graph. The example uses additional constructs such as range restriction and role composition. In general, checking satisfiability (and subsumption) of concept terms containing role value maps (agreements) and role compositions is undecidable (see [Hanschke 1992, Hanschke 1993]). However, in KONWERK and PLAKON, constraints are not evaluated at the concept level, i.e. constraints are checked only for ABox objects. This might result in a larger search space during the configuration phase. The constraint satisfaction algorithm for checking consistency of ABox structures used in PLAKON and KONWERK is incomplete in general, it uses local propagation techniques. Furthermore, constraint solving can be explicitly postponed by defining a certain control strategy (see Section 2.1). This design decision has been made to keep constraint reasoning tractable.

In case of a numeric constraint, if the arguments of an $n$-ary constraint are specified by $n$ differently named slots, then this can be transformed to a predicate of a concrete domain. In general, however, the constraint language of BHIBS is much too expressive to be transformed to description logics; it allows to quantify over more than one or two variables (for a discussion of the relationship between description logics and predicate logic see [Borgida 1996]).

## 3.5 Defaults

PLAKON's and KONWERK's concept languages provide a means for specifying defaults for the slots of certain objects, but their intended meaning is not quite clear. They are used for focusing the search mechanism, but there is no notion of quality of solutions in PLAKON and KONWERK. By using the approach of Quantz & Royer [1992] ("Preferential Default Description Logics") defaults can be used for defining a preference relation on the set of solutions. However, it can be shown that PLAKON's and KONWERK's use of defaults for focusing search does not guarantee the generation of the optimal solution with respect to this preference relation. Due to space restrictions we cannot discuss this in detail in this contribution.

## 4 Innovation and Creativity in Configuration Tasks

A formal, logical approach to configuration as advocated in this contribution might be very helpful for analyzing open problems, e.g. the intended meaning of notions like *innovative* or even *creative* configuration [Hotz & Vietze 1995b]. In this paper we define innovation in the context of configuration problems in terms of in description logics as a process of dynamic classification. The definition is motivated by an example.

Let us assume there exists a domain model with concepts for various real world objects, for instance, ships, houses etc. Maybe houses of different kinds are represented using *defined* concepts (i.e. concepts with necessary and *sufficient* conditions) and houses and ships are not disjoint. In our example we assume the initial construction task is to design a Ship that satisfies certain restrictions (e.g. number of persons, number of bedrooms as well as convenience or luxury criteria). Let us further assume that a certain ship s1 has been designed. Due to the cover axioms in the TBox (see above), the ABox instance s1 is subsumed by a leaf subconcept of Ship. After the design has been completed, the customer is asked whether he is satisfied with the result. Maybe the customer adds additional constraints to the designed artifact s1 using the relations defined in the domain model. The additional restrictions might cause the sufficient conditions for a House concept to be satisfied. If this happenes, the construction process will try to further specialize the ship s1 using the house concepts (see the cover and disjointness axioms). Thus, the designed Ship can also be used as House. The fact that the ABox discovers that House (a sibling of the initial concept Ship) also holds and the subsequent specialization of the sibling concept can be interpreted as the task of designing a houseboat. The House concept (or a subconcept of House) serves as a dynamically instantiated view in this respect that imposes additional constraints because of the associated cover axioms. The new artifact might better satisfy certain optimization criteria.

In this case, innovative design is possible because additional restrictions are asserted for a *single* ABox instance s1 (innovative design by imposing additional restrictions). Note that there is no concept definition for a Houseboat in the domain model. If there had been such a concept definition as a subconcept of Ship (with the same additional restrictions), the TBox classification process would have inferred in advance that the defined concept House is a superconcept of Houseboat. Thus, there would be no innovation at all. Innovation can be defined to be a *task reformulation by adding restrictions after the configuration has been comppleted in order to find additional defined concepts* to hold together with the subsequent specialization of these defined concepts. When the concept term describing the instance s1 is computed and inserted into the TBox, a new concept Houseboat is created (of course, the name would have to be computed by additional processes).

Note that this is impossible when storage-oriented slots are used as a basis for expressing ABox restrictions. With PLAKON's and KONWERK's limited ABox expressibility (see Section 3.3), additional restrictions that trigger the derivation of House cannot be expressed without knowing in advance that a Ship s1 is also a House.

Innovation can also require *goal-directed relaxation of restrictions*. For instance, minimum cardinality restrictions for certain roles might be relaxed such that more restricted maximum cardinalities can be asserted (either explicitly or by applying the closed world assumption by "closing" a role). In our example, the "goal" would be to relax the constraints of s1 such that a defined concept (like House) can be proved to hold. This concept will again be subclassified to leaf concepts etc. Other kinds of innovation might require the automatic definition of new ontological vocabulary. How this can be achieved is still an open question.

## 5 Conclusion of the Case Study

Using the languages PLAKON and KONWERK as a case study, the paper demonstrates that specialized languages (e.g. developed for configuration problems) can be interpreted as a special purpose description logic. The construction or configuration process as defined by PLAKON and KONWERK can be "simulated" by a model-constructing satisfiability prover. The constructed logical model represents the artifact to be designed. We also demonstrate a methodology for describing the meaning of specialized languages by applying syntactical transformations to description logic formulae. Considering these transformations, a language designer can easily estimate the computational costs of intended constructs by exploiting the rich research literature on description logics.

With the implementation of KONWERK, several prototype applications have been built. In this paper, we cannot discuss all aspects of this large system. Especially, we do not claim that the usual syntax for description logics is adequate for all users. Maybe the syntax and modeling philosophy of BHIBS (with object descriptors, see Figure 1) is better suited to engineers. With this paper however, we hope to provide a basis for defining an integrated semantics for application-oriented configuration development environments. The results hold also for distributed or Web-based systems. The article shows that both approaches – practical and theoretical approaches – are valuable contributions to AI research and both can complement each other. The semantics for PLAKON and KONWERK we gave in this paper indicates what kinds of term constructors are required for BHIBS and its extensions (see Section 3.1 and Section 3.4). The term constructs used in the semantics make clear that at least EXPTIME algorithms are required to check consistency of BHIBS concepts. We have seen that in some cases it is difficult to verify that the resulting language is decidable at all. The analysis reveals that efficiency (or tractability) is not a question of using a description logic or not but it is a question of how complete a solution to a configuration problem is expected to be wrt. a formally defined semantics.

## Acknowledgments

We thank Lothar Hotz and Bernd Neumann for interesting discussions and for explaining details about the systems PLAKON and KONWERK.

## References

[Artale et al. 1996] A. Artale, E. Franconi, N. Guarino, L. Pazzi. Part-Whole Relations in Object-Centered Systems: An Overview. In Data and Knowledge Engineering, 20 (1996) 347-383, North-Holland, Elsevier.

[Baader 1991] Franz Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. DFKI-Research Report RR-90-13, also published in Proc. IJCAI-91.

[Baader & Hanschke 1991] Franz Baader, Philipp Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. DFKI-Research Report RR-91-10, also published in Proc. IJCAI-91.

[Baader et al. 1996] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. Artificial Intelligence, 88 (1996), 195–213, shorter version published in: Proc. KI-94: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence 861, Springer 1994.

[Borgida et al. 1989] Alex Borgida, Ron Brachman, Deborah McGuiness, Lori Alperin Resnick. CLASSIC: A Structural Data Model for Objects. In Proc. ACM SIGMOD-89 International Conference on Management of Data, pp. 59–67, 1989.

[Borgida 1996] Alex Borgida. On the relative expressiveness of description logics and predicate logics. Artificial Intelligence, 82 (1996), 353–367.

[Borgida et al. 1996] Alex Borgida. Reasoning with Black Boxes: Handling Test Concepts in CLASSIC. In Proc. DL-96, Boston, 1996.

[Buchheit et al. 1995] Martin Buchheit, Rüdiger Klein, and Werner Nutt. Constructive Problem Solving: A Model Construction Approach towards Configuration. DFKI Technical Memo TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, January 1995.

[Cunis et al. 1991] Roman Cunis, Andreas Günter, and Hellmut Strecker, editors. *Das PLAKON-Buch – Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen*, volume 266 of *Informatik-Fachberichte*. Springer-Verlag, Berlin – Heidelberg – New York, 1991.

[Cunis et al. 1989] Roman Cunis, Andreas Günter, Ingo Syska, Heino Peters, H. Bode. PLAKON–An approach to domain-independent construction. In Proc. 2. IEA/AIE, Tennesse, USA, ACM-Press, pp. 866–874, 1998.

[Günter & Cunis 1992] Andreas Günter, Roman Cunis. Flexible Control in Expert Systems for Construction Tasks. In Int. Journal Applied Intelligence, Kluwer Acadmic Press Vol. 2, 1992.

[Günter 1995a] Andreas Günter. Ein pragmatischer Ansatz zur Auswertung von taxonomischen Relationen bei der Konfigurierung. In [Günter 1995b], chapter 18.

[Günter 1995b] Andreas Günter, editor. *Wissensbasiertes Konfigurieren – Ergebnisse aus dem Projekt PROKON.* Infix, Sankt Augustin, 1995.

[Hanschke 1992] Philipp Hanschke. Specifying Role Interactions in Concept Languages. In [KR 1992], pages 318–329.

[Hanschke 1993] Philipp Hanschke. A declarative integration of terminological, constraint-based, data-driven, and goal-directed reasoning. Deutsches Forschungszentrum für Künstliche Intelligenz: Research Reports; RR-93-46.

[Hollunder & Baader 1991] Bernhard Hollunder, Franz Baader. Qualifying Number Restrictions in Concept Languages. DFKI-Research Report RR-91-03.

[Hotz & Vietze 1995a] Lothar Hotz and Thomas Vietze. Erweiterung der Begriffshierarchie um Sichten und Mehrfachvererbung. In Günter [1995b], chapter 11.

[Hotz & Vietze 1995b] Lothar Hotz and Thomas Vietze. Innovatives Konfigurieren als Erweiterung des modellbasierten Ansatzes. In Günter [1995b], chapter 4.

[KR 1992] Bernhard Nebel, Charles Rich, and William Swartout, editors. *Principles of Knowledge Representation and Reasoning – Proc. of the Third International Conference KR'92*, Cambridge, Mass., October 25–29, 1992. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1992.

[Kopisch & Günter 1992] Manfred Kopisch, Andreas Günter. Knowledge-based Support for the Layout Development of the AIRBUS-A340 Passenger Cabin. Proc. 12th International Conference on Artificial Intelligence, Expert Systems and Natural Language, Avignon 1992, Vol. 2, pp. 507–517.

[Lutz 1996] Carsten Lutz. Untersuchungen zu Teil-Ganzes-Relationen – Modellierungsanforderungen und Realisierung in Beschreibungslogiken. Memo FBI-HH-M-258/96, Fachbereich Informatik, Universität Hamburg, April 1996.

[McGuiness & Resnick 1995] Deborah McGuiness, Lori Alperin Resnick. Description Logic-based Configuration for Consumers. In: Proc. DL-95, 1995, pp. 109–111.

[McGuiness et al. 1995] Deborah McGuiness, Lori Alperin Resnick, Charles Isbell. Description Logic in Practice: A CLASSIC Application. In the Proc. IJCAI'95, Montreal, Canada, 1995.

[Neumann 1988] Bernd Neumann. Configuration Expert Systems: a Case Study and Tutorial. In: Artificial Intelligence in Manufacturing, Assembly and Robotics, H. Bunke (Ed.), Oldenbourg, Munich, 1988.

[Owsnicki-Klewe 1988] Bernd Owsnicki-Klewe. Configuration as a Consistency Maintenance Task. In Wolfgang Hoeppner, editor, *GWAI-88 12th German Workshop on Artificial Intelligence*, Eringerfeld, September 1988, volume 181 of *Informatik-Fachberichte*, pages 77–87. Springer-Verlag, Berlin – Heidelberg – New York, 1988.

[Quantz & Royer 1992] Joachim Quantz and Véronique Royer. A Preference Semantics for Defaults in Terminological Logics. In [KR 1992], pages 294–305.

[Sattler 1995] Ulrike Sattler. A Concept Language for an engeneering application with part-whole relations. In Proc. of the International Workshop on Description Logics, pp. 119–123, Rome, 1995.

[Schaerf 1994] Andrea Schaerf. Reasoning with individuals in concept languages. Data & Knowledge Engineering Vol. 13, No. 2, 1994, pp. 141–176.

[Woods & Schmolze 1992] W.A. Woods and J.G. Schmolze. The KL-ONE Family. In F. Lehmann, editor, Semantic Networks in Artificial Intelligence, pages 133–177. Pergamon Press, Oxford, England, 1992.