A Partial Logical Reconstruction of PLAKON/KONWERK

Carsten Schröder and **Ralf Möller** and **Carsten Lutz**
Universität Hamburg, Fachbereich Informatik,
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany,
{schroeder,moeller,lutz}@kogs.informatik.uni-hamburg.de

1 Introduction

The main goal of the projects TEX-K and PROKON, carried out at the Dept. of Computer Science, University of Hamburg in the years of 1986–1990 and 1991–1995, respectively, and dealing with knowledge-based configuration in technical domains, was to develop suitable representation languages tailored to the needs of the configuration domain and a methodology for actually solving configuration tasks [Cunis et al. 1991, Günter 1995b]. The emphasis of these efforts was on building practical, useful software tools instead of formal methods and culminated in the system PLAKON and its successor KONWERK.

In this contribution we argue that the methods of formal knowledge representation, especially description logics, first, are valuable tools for analyzing existing systems and open problems, 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 systems. In the following we present a partial logical reconstruction of PLAKON and KONWERK. Section 2 introduces the view that the methodology used in PLAKON and KONWERK for solving configuration tasks can be seen as a special instance of a process resulting from a precise definition of the configuration problem in logical terms. Section 3 shows how most of the concept definitions and some of the constraint definitions using PLAKON's representation languages can be transformed to terminological axioms of a description logic and explains some of the peculiarities of the languages. Section 4 discusses how the use of formal methods helps in understanding open problems of the configuration domain. The paper ends with a conclusion.

2 The Configuration Methodology

In this section we give an introduction 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 satisfiability testing.

2.1 Defining the Configuration Space

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 domain 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.

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 (applying a construction step is called a "heuristic decision" in PLAKON's and KONWERK's terminology). The control module can be adapted to the problem using an explicit model with "strategies" for traversing the construction space (see [Günter 1991]). 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 hierachy 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 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 PLAKON and KONWERK 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 BHIBS representation language. 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 of a set of objects (the *domain of discourse*) as well as an interpretation function which maps object names to the objects of the domain of discourse and concepts as well as slot names to unary and binary relations on the domain of discourse, respectively, and it is required to satisfy the formulas of the given knowledge base.

A bit of analysis reveals that the set of objects and relations represented by slots which are constructed by the configuration process in PLAKON and KONWERK 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.

Interestingly, although the classification services usually provided by description logics are not the central mechanisms needed for configuration (as noted by Günter [1995a]), 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 the configuration technique used in 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 *whole-part* relation [Lutz 1996], cannot be represented using functional roles (features).

3 The Language

PLAKON provides a language called BHIBS which can be used for modeling a domain by defining concepts [Cunis 1991]. In this section we present the main ideas behind BHIBS and illustrate how the language constructs can be transformed to description logics or, if this is not possible, to First-Order Predicate Logic.

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] :=
```

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

```
#[(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 [Hanschke 1992] 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):

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

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

In an effort to provide a formal declarative semantics for BHIBS we found that all concept definitions except those containing sequence description 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.

There are a few things to note 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* which are interpreted as partial functions. 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, 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 themself 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 Buchheit et al. [1995]). If, for example, a concept A has the direct subconcepts B, C, and D, then the following axioms must be added to the TBox:

$$\mathsf{A} \sqsubseteq \mathsf{B} \sqcup \mathsf{C} \sqcup \mathsf{D}$$

$$\mathsf{B} \sqsubseteq \neg\mathsf{C} \qquad \mathsf{B} \sqsubseteq \neg\mathsf{D} \qquad \mathsf{C} \sqsubseteq \neg\mathsf{D}$$

After adding cover and disjointness axioms, "specialization to leaf concepts" is done by a model construction process as well. Note that this formalization of the original assumption of a complete domain model easily shows that it does not correspond to a *closed world assumption* as claimed by Cunis et al. [1991] and Günter [1995a].

The basic PLAKON and KONWERK systems support only incomplete reasoning services for checking the domain model. For instance, the cover axioms, might implicitly add additional restrictions to A. Let us assume the following declarations impose restrictions on B, C and 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 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

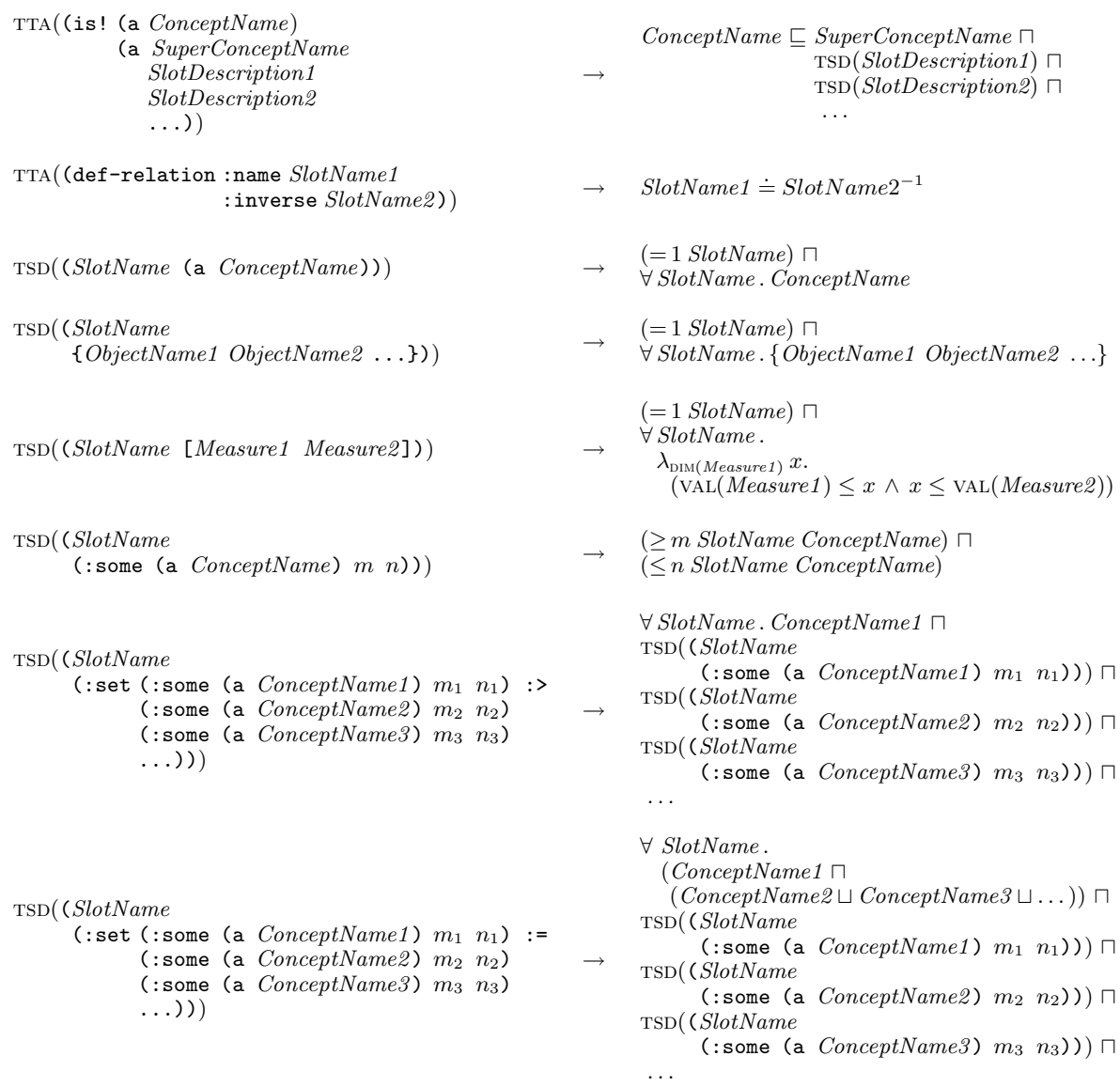

Figure 1: Rules for transforming a BHIBS terminology.

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 A, it will be known in beforehand that there is no need to try whether e.g. only ten role fillers for r are sufficient for an A. [Günter 1995a] uses an example with intervals to demonstrate the facilities of TAX.

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,
- Qualified number restrictions,
- Qualified existential restriction,
- Negation and disjunction with primitive concept names and
- Concrete domains over $\Re$.

Furthermore, it should be noted that formulas are not arbitrarily nested, i.e. 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 was not loaded into the KONWERK system. Currently, it is still not clear whether the inference services of BHIBS together with TAX are complete with respect to the semantics we defined in this paper

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 a logical point of view. If multiple inheritance were used, construction step 2 (see Section 2.1) would not be sufficient to traverse the configuration space.

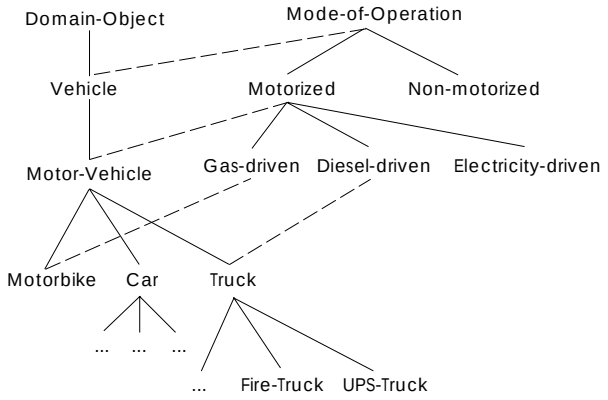

Figure 2: Example for a concept hierarchy with mixins.

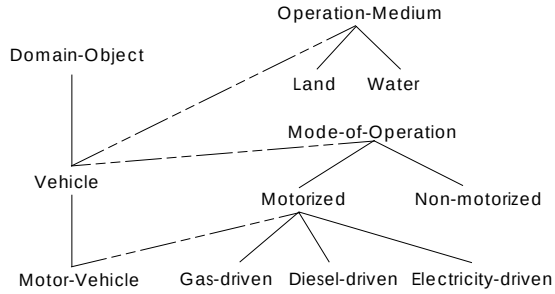

Figure 3: Example for a concept hierarchy with views.

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 no single leaf concept to describe a configuration object. Furthermore, since 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 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 equality axioms because the semantics for using a mixin name in a concept definition and for directly including the mixin definition term (right side of the concept definition) should be identical.

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. The semantics of views was not very well understood and quite confusing when first specified by Cunis [1991]. Recently, Hotz & Vietze [1995a] gave an interpretation of this notion in terms of a restricted form of multiple inheritance with mixins.

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

The semantics of view links is different to that of mixin links (see Figure 2). The procedural 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. 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.

With description logics no restructuring of the in-

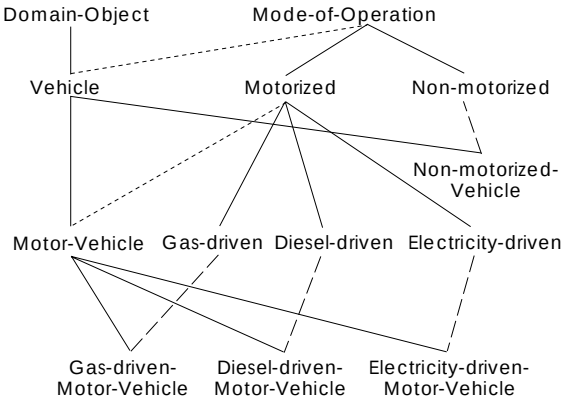

Figure 4: Expanded concept hierarchy.

heritance 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 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 axioms are generated:

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

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).

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 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 exists 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 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 arti-

$$\text{TAA}(\texttt{(set-slot } \textit{ObjectName1 SlotName ObjectName2}\texttt{)}) \rightarrow (\textit{ObjectName1}, \textit{ObjectName2}) : \textit{SlotName}$$

$$\text{TAA}(\texttt{(set-slot } \textit{ObjectName SlotName Measure}\texttt{)}) \rightarrow \textit{ObjectName} : (\exists\ \textit{SlotName}\,.\ \lambda_{\text{DIM}(\textit{Measure})}\, x.\, (x = \text{VAL}(\textit{Measure})))$$

$$\text{TAA}(\texttt{(set-slot } \textit{ObjectName SlotName ObjectDescriptor}\texttt{)}) \rightarrow \textit{ObjectName} : \text{TSD}(\texttt{(}\textit{SlotName ObjectDescriptor}\texttt{)})$$

Figure 5: Rules for transforming assertions.

$$\text{TSD}(\texttt{(has-parts (:ct (:kk-menge (a Vertex) } m_1\ n_1 \texttt{ (an Edge) } m_2\ n_2\texttt{))))}) \rightarrow \begin{array}{l} \forall\, \mathsf{has\text{-}parts}\,.\, (\mathsf{Vertex} \sqcup \mathsf{Edge}) \sqcap \\ (\geq m_1\ \mathsf{has\text{-}parts}\ \mathsf{Vertex}) \sqcap \\ (\leq n_1\ \mathsf{has\text{-}parts}\ \mathsf{Vertex}) \sqcap \\ (\geq m_2\ \mathsf{has\text{-}parts}\ \mathsf{Edge}) \sqcap \\ (\leq n_2\ \mathsf{has\text{-}parts}\ \mathsf{Edge}) \end{array}$$

$$\text{TSD}(\texttt{(has-parts (:ct (Vertex Vertex Vertex Vertex) ((Edge 1 2) (Edge 2 3) (Edge 3 4)))))}) \rightarrow \begin{array}{l} \{\, a \mid \exists\ v_1, v_2, v_3, v_4, e_1, e_2, e_3 : \\ \quad \mathsf{has\text{-}parts}(a, v_1) \wedge \cdots \wedge \mathsf{has\text{-}parts}(a, v_4) \wedge \\ \quad \mathsf{has\text{-}parts}(a, e_1) \wedge \cdots \wedge \mathsf{has\text{-}parts}(a, e_3) \wedge \\ \quad \mathsf{Vertex}(v_1) \wedge \cdots \wedge \mathsf{Vertex}(v_4) \wedge \\ \quad \mathsf{Edge}(v_1) \wedge \cdots \wedge \mathsf{Edge}(v_3) \wedge \\ \quad \mathsf{has\text{-}vertex}(e_1, v_1) \wedge \mathsf{has\text{-}vertex}(e_1, v_2) \wedge \\ \quad \mathsf{has\text{-}vertex}(e_2, v_2) \wedge \mathsf{has\text{-}vertex}(e_2, v_3) \wedge \\ \quad \mathsf{has\text{-}vertex}(e_3, v_3) \wedge \mathsf{has\text{-}vertex}(e_3, v_4)\ \} \end{array}$$

Figure 6: Rules for transforming graph structure specifications.

fact). 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.

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

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.

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 is much too expressive to be transformed to description logics; it allows to quantify over more than one or two variables. The constraint reasoner of PLAKON and KONWERK is incomplete in general, it uses local propagation. Furthermore, constraint solving can be explicitly postponed by defining a certain control strategy (see Section 2.1).

In this section we have used general graph structures as an example for the use of constraints. More specific graph structures are discussed in the next section.

3.5 Configuration of Graph Structures

In KONWERK special modeling constructs have been added to BHIBS to represent the construction space for graph structures (see [Bartuschka 1995]). In a similar way as the object descriptors presented above, special constructors for vertex and edge structures are supported. Figure 6 defines the mapping for slot descriptions that contain graph structure specifications. While the first descriptor can be mapped to description logic constructs, the second descriptor is mapped to First-Order Predicate Logic. In this description, the parts are explicitly named (see the existential quantifier). The second

graph (polyline with three edge elements) requires seven parts to be named. Thus, in general, the construction or configuration space for graph structures cannot be represented in description logics.

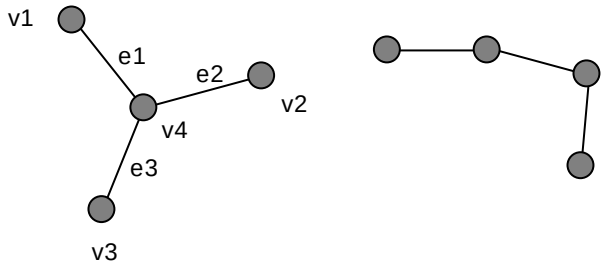

Figure 7: Two examples for configurations of vertices and edges: a star and a polyline.

In Figure 7 we present a few examples for graph structures. From a data representation point of view, graph structures (e.g. a star) can easily be represented in the ABox. Furthermore, it is not very difficult to define a TBox that can be used to "recognize" a certain graph structure. In this paper, we discuss a small TBox for recognizing the star of Figure 7:

$$
\begin{aligned}
\mathsf{End\text{-}Vertex} \doteq\ & \mathsf{Vertex} \sqcap \\
& (\leq 1\ \mathsf{vertex\text{-}of}) \\
\mathsf{Middle\text{-}Vertex} \doteq\ & \mathsf{Vertex} \sqcap \\
& (= 3\ \mathsf{vertex\text{-}of}) \sqcap \\
& \forall\ \mathsf{vertex\text{-}of}\ .\ \mathsf{End\text{-}Edge} \sqcap \\
& \neg \mathsf{End\text{-}Vertex} \\
\mathsf{Star} \doteq\ & \mathsf{Graph} \sqcap \\
& (= 7\ \mathsf{has\text{-}parts}) \sqcap \\
& (= 3\ \mathsf{has\text{-}parts}\ \mathsf{Edge}) \sqcap \\
& (= 3\ \mathsf{has\text{-}parts}\ \mathsf{End\text{-}Edge}) \sqcap \\
& (= 4\ \mathsf{has\text{-}parts}\ \mathsf{Vertex}) \sqcap \\
& (= 3\ \mathsf{has\text{-}parts}\ \mathsf{End\text{-}Vertex}) \sqcap \\
& (= 1\ \mathsf{has\text{-}parts}\ \mathsf{Middle\text{-}Vertex})
\end{aligned}
$$

To represent configurations like the star in Figure 7 corresponding concepts and relations are defined. Furthermore, initial assertions must be submitted to the ABox.

$$
\begin{aligned}
&\mathsf{v1} : \mathsf{Vertex}, \mathsf{v2} : \mathsf{Vertex}, \mathsf{v3} : \mathsf{Vertex}, \mathsf{v4} : \mathsf{Vertex} \\
&\mathsf{e1} : \mathsf{Edge}, \mathsf{e2} : \mathsf{Edge}, \mathsf{e3} : \mathsf{Edge}, \\
&(\mathsf{e1}, \mathsf{v1}) : \mathsf{has\text{-}vertex}, (\mathsf{e1}, \mathsf{v4}) : \mathsf{has\text{-}vertex} \\
&(\mathsf{e2}, \mathsf{v2}) : \mathsf{has\text{-}vertex}, (\mathsf{e2}, \mathsf{v4}) : \mathsf{has\text{-}vertex} \\
&(\mathsf{e3}, \mathsf{v3}) : \mathsf{has\text{-}vertex}, (\mathsf{e3}, \mathsf{v4}) : \mathsf{has\text{-}vertex}
\end{aligned}
$$

It can easily be seen that the ABox classifies the vertices v1, v2 an d v3 as End-Vertices. Therefore, all edges are End-Edges and v4 is a Middle-Vertex. The graph the objects are part of is classified as a Star. The main idea of the "recognition process" has been published in [Haarslev et al. 1994].

In a model construction prover the object representing the graph is automatically generated. If a structural subsumption prover is used, an aggregate to actually represent the star can be created using a *rule*. The vertex v4 can be seen as a representative for the star and Middle-Vertex can be used in the antecedent part of the rule. A semantics for rules with epistemic operators and the use of rules to create aggregates is defined in [Hanschke 1993]. Hanschke [1993] also introduces *transitive closure* as an extension to role specifications. Transitive closures are required e.g. to represent polylines. Transitive closures are also required to augment the task specification in PLAKON and KONWERK. We have seen that one main goal object and a set of additional goal objects can be given as a specification of a construction task. The additional objects must be part-of* the main goal object.

We have seen that graph structures can be interpreted as a special case of part-whole relations. The quintessence is that the construction space for graph structures in general cannot be represented with description logics. However, for "recognizing" specific graph structures, adequate concepts and relations can be defined. That is what description logic is all about: It provides a basis that allows domain-specific concepts and relations to be defined and, thus, allows inference steps to be formally modeled. Completeness of a description logic ABox reasoner ensures that a model developer must not deal with control aspects such as the correct sequence of elementary inference steps or the administration of trigger events etc. If concepts and relations cannot be defined using the constructs of description logics, a more expressive logic could be used. However, if full First-Order Predicate Logic were used, the satisfiability problem would be undecidable, i.e. the reasoner must be incomplete.

3.6 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 [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.

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 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 beforehand 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.

5 Conclusion

The paper demonstrates that PLAKON and KONWERK can be interpreted as a special purpose description logic reasoner, i.e. a model-constructing prover for a very specific description logic with a limited sort of ABox. The construction or configuration process as defined by PLAKON and KONWERK can be "simulated" by a model-constructing satisfiability prover for description logics. The constructed logical model represents the artifact to be designed.

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). Further investigations must show whether the resulting language is decidable. Although, in general, including role value maps leads to an undecidable language (see Hanschke-92a), we must be careful here because there are some restrictions on term forming operators (e.g. negation and disjunction only with names for primitive concepts).

In our opinion, Günter's [1995a] and Richter's [1995] argument that terminological systems are inadequate for reasons of efficiency is misleading as long as the complexity of configuration tasks is unknown, for a careful analysis of the terminological language used in our transformation might show that the satisfiability problem – which is central for configuration tasks – is intractable or even undecidable for this language. 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.

With the implementation of KONWERK, several prototype applications have been built. In comparison to PLAKON, in KONWERK many additional modules have been added (Fuzzy values, optimization

strategies, etc.). This research clearly demonstrates the necessity of adequate representation and reasoning systems. In this paper, we cannot discuss all aspects of this large system (see also, for instance, [Heinsohn 1992] for an approach for modeling uncertainty in description logics). 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 the submodules of KONWERK. The contribution shows that both approaches – practical and theoretical approaches – are valuable contributions to AI research and both can complement each other.

Acknowledgments

We thank Lothar Hotz for explaining numerous details of the systems PLAKON and KONWERK.

References

[Bartuschka 1995] Ulrike Bartuschka. Repräsentation von Graphstrukturen. In Günter [1995b], chapter 19.

[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 1991] Roman Cunis. Modellierung technischer Systeme in der Begriffshierarchie. In Cunis et al. [1991], chapter 5.

[Günter 1991] Andreas Günter. Begriffshierarchieorientierte Kontrolle. In Cunis et al. [1991], chapter 7.

[Günter 1995a] Andreas Günter. Ein pragmatischer Ansatz zur Auswertung von taxonomischen Relationen bei der Konfigurierung. In [Günter 1995b], chapter 7.

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

[Haarslev et al. 1994] Volker Haarslev, Ralf Möller, and Carsten Schröder. Combining Spatial and Terminological Reasoning. In Bernhard Nebel and Leonie Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence – Proc. 18th German Annual Conference on Artificial Intelligence*, Saarbrücken, September 18–23, 1994, volume 861 of *Lecture Notes in Artificial Intelligence*, pages 142–153. Springer-Verlag, Berlin – Heidelberg – New York, 1994.

[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. DFKI Research Report RR-93-46, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, October 1993.

[Heinsohn 1992] Jochen Heinsohn. A Hybrid Approach for Modeling Uncertainty in Terminological Logics. DFKI Research Report RR-92-24, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, August 1992.

[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.

[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.

[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.

[Richter 1995] Michael M. Richter. Kommentierung und Wertung der PROKON-Ergebnisse. In Günter [1995b], chapter 7.