Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

**Document**
D-96-04

# Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96

**Franz Baader, Hans-Jürgen Bürckert,**
**Andreas Günter, Werner Nutt (Eds.)**

**August 1996**

**Deutsches Forschungszentrum für Künstliche Intelligenz GmbH**

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

# Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

# Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96

**Franz Baader, Hans-Jürgen Bürckert, Andreas Günter, Werner Nutt (Eds.)**

DFKI-D-96-04

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-95 004).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1996
This work may not be copied or reproduced in whole of part for any commercial purpose. Permission to copy in whole or part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Deutsche Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.
ISSN 0946-0098

# Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96

Franz Baader, Hans-Jürgen Bürckert,
Andreas Günter, Werner Nutt (Eds.)

August 30, 1996

## Preface

Configuration of technical systems from given components is a domain in which AI systems have been used in industrial applications for quite a while. The first configuration systems, such as the legendary XCON/R1, could only represent the way an expert solves the problem, but it did not provide for means of representing the structure of the configuration problem itself. These systems were rather hard to maintain and handle, which was partially due to this lack of a formal and structured way of representing the problem to be solved. Thus, it is quite surprising that formal methods from knowledge representation, which could be used to overcome these deficits, have not been employed more often in configuration domains, and when so only with partial success.

While the above mentioned AI systems were only concerned with the problem of HOW to configure a technical systems, formal KR approaches could also be used to represent in a structured and formally well-understood way WHAT is to be configured. Examples of such approaches are techniques for handling

- taxonomies of concepts,
- part-whole hierarchies,
- constraints,
- rules,
- vague and uncertain knowledge (e.g. by using fuzzy logic),
- the difference between object and meta knowledge.

The goal of this workshop was to find out whether existing KR techniques are appropriate for treating configuration problems, or whether they are not necessary at all in this domain. To this purpose, different groups of people met at the workshop:

- researchers that are concerned with configuration problems in general, or are working on configuration problems in a specific application domain;
- researcher that are concerned with developing formal KR methods, or are developing a concrete KR system;
- researchers that already employ a KR system in a configuration application.

According to our proposal for the topics and questions to be discussed we got rather different position papers, which are collected in this report. They discuss open problems of practical application as well as logical foundation of configuration systems. Advantages and disadvantages of several description logics as a basis of knowledge representation for configuration, modeling and representing part-whole-relationships, and usage of fuzzy constraints are addressed. Some of the papers describe concrete configuration systems or consider such for their discussion of problems.

Abstracts of all contributions are published in the internet

http://www.dfki.uni-sb.de/~hjb/WRKP-96

both in German and in English together with postscript-files of the English extended versions in this report.

Based on the presented contributions we discussed the following questions in more detail:

- Which methods from formal KR can be employed for modeling and solving configuration problems?
- What are the advantages and disadvantages of using such methods (e.g., ease of maintenance vs. performance deficits)?
- Are some of these techniques appropriate for only a specific subclass of problems?
- How can different techniques be integrated in one system?
- What is the "logic" of configuration problems? Can they be seen as deductive or rather as abductive problems?

We plan to connect a summary of the resulting answers to the internet publications.

August, 1996

Franz Baader, RWTH Aachen
Hans-Jürgen Bürckert, DFKI Saarbrücken
Andreas Günter, TZI, Universität Bremen
Werner Nutt, DFKI Saarbrücken

## Content

# Extensions of Terminological Knowledge Representation Languages for Technical Applications

**Franz Baader**
Theoretical Computer Science, RWTH Aachen
Ahornstr. 55, 52074 Aachen
baader@informatik.rwth-aachen.de

## Abstract

We consider two extensions of traditional terminological knowledge representation languages, which are motivated by technical applications such as configuration of technical systems. The first extension integrates "concrete" domains (such as numbers) and concrete predicates on these domains into the abstract terminological language. The second extension introduces transitive closure of roles, which can, for example, be used to model transitivity of the "part-of" relation.

## 1 Introduction

Terminological knowledge representation (KR) systems are used to introduce the relevant concepts of an application domain (i.e., its terminology). In addition, concrete objects can be characterized with respect to their membership in concepts and their interrelations with each other. An important feature of terminological KR systems is that they are equipped with a formally well-founded semantics, and that they can deduce implicit knowledge from the explicitly represented knowledge. For example, the system can automatically calculate subconcept/superconcept relationships (so-called subsumption relationships) from the definitions of the concepts; these relationships need not be stated explicitly as IS-A relationships by the knowledge engineer.

Whereas early terminological systems (such as KL-ONE [Brachman and Schmolze,1985]) have been developed with natural language processing applications in mind, modern systems (such as CLASSIC [Brachman *et al.*,1991a; 1991b], KREP [Mays *et al.*,1991], BACK [Peltason,1991], LOOM [MacGregor,1991], and KRIS [Baader and Hollunder,1991; Baader *et al.*,1994]) are more and more employed in technical domains (such as configuration of technical systems). It has turned out, however, that the concept description formalisms of traditional terminological systems are not expressive enough for such applications. We consider two extensions of terminological KR systems, which were motivated by applications in mechanical engineering and in process engineering.

*Extension by "concrete domains":* In traditional terminological description languages, all the knowledge about the relevant concepts must be expressed on an abstract logical level. It is not possible to refer to concrete domains (such as natural numbers, real numbers) and predefined (built-in) relations and operations on these domains (such as comparisons like $\leq$ on numbers, arithmetical operations on numbers). In technical applications, one often needs to state geometric or other types of numerical constraints, which explains the need for concrete domains in this context [Baader and Hanschke,1993].

*Extension by transitive closure:* The adequate representation of the complex structure of technical systems necessitates the use of the part-whole relationship in the concept descriptions [Sattler,1995; 1996]. This relationship cannot simply be described by a new binary relation (an atomic "role" in the terminological language) since important properties of part-whole relations (like transitivity) would not be modeled this way. An extension of the terminological formalism by transitive closure of roles allows for a correct representation of transitivity properties, which can thus also be used in reasoning about the concept descriptions.

For both of the above mentioned language extensions, sound and complete inference algorithms (for subsumption and other important inference problems) have been developed [Baader and Hanschke,1991; Baader,1991]. Unfortunately, the combination of both extensions leads to undecidability of these inference problems [Baader and Hanschke,1993].

In the next section, we introduce the prototypical terminological KR language $\mathcal{ALC}$, and the important inference problems for terminological languages. In this and in the subsequent sections, we restrict our attention to the concept description part of the language, i.e., to the formalism for building complex concept descriptions. All results can, however, also be transferred to TBoxes (where names are introduced as abbreviations for descriptions) and ABoxes (where objects and their relationship to concepts and

roles are introduced). The third section considers the extension of $\mathcal{ALC}$ by constructs that refer to concrete domains, and the fourth section considers the extension of $\mathcal{ALC}$ by transitive closure of roles (and by union and composition of roles). The last section considers the combination of both extensions.

## 2 The prototypical language $\mathcal{ALC}$

The description formalism of $\mathcal{ALC}$ allows one to built complex concept descriptions (representing classes of objects) out of atomic concepts and roles (binary relations between objects).

**Definition 2.1 (Syntax of $\mathcal{ALC}$)**
Concept descriptions *are built from concept and role names using the concept-forming operators* negation *($\neg C$),* disjunction *($C \sqcup D$),* conjunction *($C \sqcap D$),* existential restriction *($\exists R.C$), and* value restriction *($\forall R.C$). Here $C$ and $D$ are syntactic variables for concept descriptions, and $R$ stands for a role name.*

Using the concept names `Human` and `Female`, and the role name `has-child`, we can define the concept "woman" as `Human`$\sqcap$`Female`, "man" as `Human`$\sqcap$ $\neg$`Female`, and "father that has only daughters" as

$$\texttt{Human} \sqcap \neg\texttt{Female} \sqcap \exists\texttt{has-child.Human} \sqcap \forall\texttt{has-child.(Female} \sqcap \texttt{Human)}.$$

The next definition gives a model-theoretic semantics for the language introduced in Definition 2.1.

**Definition 2.2 (Semantics of $\mathcal{ALC}$)**
*An* interpretation $\mathcal{I}$ *for $\mathcal{ALC}$ consists of a set $dom(\mathcal{I})$ and an interpretation function that associates with each concept name $A$ a subset $A^{\mathcal{I}}$ of $dom(\mathcal{I})$, and with each role name $R$ a binary relation $R^{\mathcal{I}}$ on $dom(\mathcal{I})$, i.e., a subset of $dom(\mathcal{I}) \times dom(\mathcal{I})$.*

*The interpretation function can be extended to arbitrary concept descriptions as follows:*

- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, *and* $(\neg C)^{\mathcal{I}} = dom(\mathcal{I}) \setminus C^{\mathcal{I}}$,
- $(\forall R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid \forall y.\ (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$,
- $(\exists R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid \exists y.\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$.

An important service terminological representation systems provide is computing the subsumption hierarchy, i.e., computing the subconcept/superconcept relationships between all the concept descriptions introduced in a terminological knowledge base. This inferential service is usually called classification.

**Subsumption:** Let $C$, $D$ be concept descriptions. Then $D$ *subsumes* $C$ (symbolically $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$.

The subsumption problem for $\mathcal{ALC}$-concepts is known to be decidable; more precisely, it is PSPACE-complete, as shown in [Schmidt-Schauß and Smolka,1991]. The algorithm described there is based on a tableau calculus. The underlying ideas can also be used to derive algorithms for various other concept languages (see, e.g., [Hollunder *et al.*,1990; Hollunder,1990; Donini *et al.*,1991a; 1991b; 1992; Hollunder and Baader,1991]).

## 3 Integrating concrete domains

In this section we introduce a formalism that is able to refer to concrete objects (like numbers), and can use predefined relations on these objects in concept descriptions. For example, one might think that being human and female is not enough to make a woman. As an additional property one could require that she should be old enough; for example, at least 21. Thus one would like to introduce a new role age, and define "woman" by an expression of the form `Human`$\sqcap$`Female`$\sqcap \geq_{21}$(age). Here $\geq_{21}$ stands for the unary predicate $\{n \mid n \geq 21\}$ of all nonnegative integers greater or equal 21. In the mechanical engineering domain described in [Baader and Hanschke,1993], reference to concrete notions such as real numbers is mandatory to represent, for example, the geometric aspects of certain classes of lathe workpieces. Stating such properties directly with reference to a given concrete domain seems to be easier and more natural than encoding them somehow into abstract concept expressions.

Before we can define the extended description language, we must first formalize the notion "concrete domain," which has until now only been used in an intuitive sense.

**Definition 3.1** *A* concrete domain $\mathcal{D}$ *consists of a set $dom(\mathcal{D})$, the domain of $\mathcal{D}$, and a set $pred(\mathcal{D})$, the predicate names of $\mathcal{D}$. Each predicate name $P$ is associated with an arity $n$, and an $n$-ary predicate $P^{\mathcal{D}} \subseteq dom(\mathcal{D})^n$.*

The following are examples of concrete domains:

- In the above example we have considered the concrete domain $\mathcal{N}$, which has the set of nonnegative integers as its domain. We have also used one of the unary predicate names $\geq_n$. In addition, we assume that we have the binary predicate $>$.
- The concrete domain $\mathcal{R}$ is defined as follows. The domain of $\mathcal{R}$ is the set of all real numbers, and the predicates of $\mathcal{R}$ are given by formulae which are built by first order means (i.e., by using logical connectives and quantifiers) from equalities and inequalities between integer polynomials in several indeterminates.[1] For example, $x + z^2 = y$ is an equality between the polynomials $p(x,z) = x + z^2$, and $q(y) = y$; and $x > y$ is an inequality between

[1] For the sake of simplicity we assume here that the formula itself is the predicate name. In applications, users will probably introduce their own intuitive names for these predicates.

very simple polynomials. From these equalities and inequalities one can, e.g., build the formulae $\exists z(x+z^2 = y)$ and $\exists z(x+z^2 = y) \vee (x > y)$. The first formula yields a predicate name of arity 2 (since it has two free variables), and it is easy to see that the associated predicate is $\{(r,s) \mid r$ and $s$ are real numbers and $r \leq s\}$. Consequently, the predicate associated to the second formula is $\{(r,s) \mid r$ and $s$ are real numbers$\} = dom(\mathcal{R}) \times dom(\mathcal{R})$.

- The concrete domain $\mathcal{Z}$ is defined as $\mathcal{R}$, with the only difference that $dom(\mathcal{Z})$ is the set of all integers instead of all real numbers.

If we want to use the predicates of the concrete domain in concept descriptions, we need an appropriate interface between the concrete domain and our abstract descriptions. If we reconsider the description Human $\sqcap$ Female $\sqcap \geq_{21}$(age), we see that applying the concrete predicate $\geq_{21}$ to age makes immediate sense only if the role age yields just one nonnegative integer.[2] Thus, we introduce a new type of roles, called attributes, which are required to be functional. We are now ready to define the extension $\mathcal{ALC}(\mathcal{D})$ of $\mathcal{ALC}$, which is parameterized by a concrete domain $\mathcal{D}$.

**Definition 3.2 (Syntax/Semantics of $\mathcal{ALC}(\mathcal{D})$)** *In addition to concept and role names, we have* attribute names. *Attribute names can be used like roles in existential and in value restrictions of $\mathcal{ALC}$. In addition, the concept description formalism of $\mathcal{ALC}$ is extended by one new construct, called predicate restriction. Assume that $f_1, \cdots, f_m$ are $m > 0$ attributes. Then $f_1 \cdots f_m$ is called an attribute chain. If $u_1, \cdots, u_n$ are attribute chains and $P$ is an $n$-ary concrete predicate then $P(u_1, \cdots, u_n)$ (*predicate restriction*) is a concept description.*

*The only differences between interpretations $\mathcal{I}$ of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALC}$ are:*

- *The* abstract domain *$dom(\mathcal{I})$ is required to be disjoint from $dom(\mathcal{D})$. As before, concepts are subsets of the abstract domain, and roles are binary relations on $dom(\mathcal{I})$.*
- *Attributes $f$ are interpreted as partial functions $f^{\mathcal{I}} : dom(\mathcal{I}) \longrightarrow dom(\mathcal{I}) \cup dom(\mathcal{D})$. They establish the only link between the abstract and the concrete domain.*

*The semantics of the predicate restriction is defined as*

$$P(u_1, ..., u_n)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid \exists r_1, ..., r_n \in dom(\mathcal{D}). \; u_1^{\mathcal{I}}(x) = r_1 \wedge ... \wedge u_n^{\mathcal{I}}(x) = r_n \wedge (r_1, ..., r_n) \in P^{\mathcal{D}}\}.$$

[2] If age yielded a set of numbers as possible role-fillers, it would not be clear what it means that this set is at least 21 (its minimum, its maximum, its sum, average, ...?).

*Here the application of the interpretation function to the attribute chains $u_i$ is the composition of the respective partial functions.*

Using the concrete domain $\mathcal{N}$, we can now, for example, define the concept of a "woman whose husband is older than her father" as Woman $\sqcap$ (husband age $>$ father age). The concrete domain $\mathcal{R}$ is very useful for describing geometric properties of objects. For example, assume that we need to talk about rectangles in the plane whose sides are parallel to the axles of the plane. We can represent such rectangles by their (lower left) corner, and the (length of) the vertical and horizontal sides. Thus, the concept "rectangle" can be described as

Axle-parallel-object $\sqcap$
$\exists$corner.($R$(x-coord) $\sqcap$ $R$(y-coord)) $\sqcap$
$\exists$vertical-side.$R_+$(length) $\sqcap$
$\exists$horizontal-side.$R_+$(length),

where corner, x-coord, y-coord, vertical-side, horizontal-side, and length are attributes, and $R$ is (a name for) the concrete predicate in $\mathcal{R}$ that consists of all real numbers, and $R_+$ is (a name for) the concrete predicate in $\mathcal{R}$ that consists of all positive real numbers. One can now also define a concept "pairs of rectangles," which have as first component and as second component a rectangle (where first and second are taken as attributes). As specializations of this concept one can define pairs of overlapping rectangles, rectangles containing each other etc. by applying appropriate concrete predicates from $\mathcal{R}$ to attribute chains.

In order to obtain inference algorithms for the extended language, one must combine the known reasoning methods for $\mathcal{ALC}$ with reasoning algorithms for the concrete domain. This is only possible if the concrete domain satisfies some additional properties.

For technical reasons (to be able to push negation into concept descriptions) we require that the set of predicate names of the concrete domain is *closed under negation*, i.e., if $P$ is an n-ary predicate name in $pred(\mathcal{D})$, then there must exist a predicate name $Q$ in $pred(\mathcal{D})$ such that $Q^{\mathcal{D}} = dom(\mathcal{D})^n \setminus P^{\mathcal{D}}$. In addition, we need a unary predicate name that denotes the predicate $dom(\mathcal{D})$. The domain $\mathcal{N}$ from above does not satisfy these properties. We must add the predicate names $<_n$ and $\geq$. The domains $\mathcal{R}$ and $\mathcal{Z}$ satisfy the properties.

The property that will be formulated now clarifies what kind of reasoning mechanisms are required in the concrete domain. Let $P_1$, ..., $P_k$ be $k$ (not necessarily different) predicate names in $pred(\mathcal{D})$ of arities $n_1$, ..., $n_k$. We consider the conjunction

$$\bigwedge_{i=1}^{k} P_i(\underline{x}^{(i)}).$$

Here $\underline{x}^{(i)}$ stands for an $n_i$-tuple $(x_1^{(i)}, ..., x_{n_i}^{(i)})$ of variables. It is important to note that neither all vari-

ables in one tuple nor those in different tuples are assumed to be distinct. Such a conjunction is said to be *satisfiable* iff there exists an assignment of elements of $dom(\mathcal{D})$ to the variables such that the conjunction becomes true in $\mathcal{D}$.

For example, let $P_1(x,y)$ be the predicate $\exists z(x + z^2 = y)$ in $pred(\mathcal{R})$, and let $P_2(x,y)$ be the predicate $x > y$ in $pred(\mathcal{R})$. Obviously, neither the conjunction $P_1(x,y) \wedge P_2(x,y)$ nor $P_2(x,x)$ is satisfiable.

**Definition 3.3** *A concrete domain $\mathcal{D}$ is called* admissible *iff* (i) *the set of its predicate names is closed under negation and contains a name for $dom(\mathcal{D})$, and* (ii) *the satisfiability problem for finite conjunctions of the above mentioned form is decidable.*

The concrete domain $\mathcal{R}$ is admissible. This is a consequence of Tarski's decidability result for real arithmetic [Tarski,1951; Collins,1975]. However, for the linear case (where the polynomials in the equalities and inequalities must be linear), there exist more efficient methods (see e.g. [Weispfenning,1988]), and in the quantifier-free linear case one even has incremental methods for deciding satisfiability [Jaakola,1990; Jaffar *et al.*,1992]. The concrete domain $\mathcal{Z}$ is not admissible since Hilbert's Tenth Problem—one of the most prominent undecidable problems [Matijacevič,1970; Davis,1973]—is a special case of its satisfiability problem.

In [Baader and Hanschke,1991] it is shown how the tableau-based reasoning algorithm for $\mathcal{ALC}$ can be extended to $\mathcal{ALC}(\mathcal{D})$, provided that $\mathcal{D}$ is admissible.

**Theorem 3.4** *If $\mathcal{D}$ is an admissible concrete domain, then the subsumption problem is decidable for $\mathcal{ALC}(\mathcal{D})$.*

## 4 Integrating transitive closure

Transitivity of the "part-of" relation can be represented by introducing a role `direct-part-of`, and by defining the role `part-of` as the transitive closure of `direct-part-of`. If one wants to allow for a more fine-grained representation of the different types of part-of relations (such as "component-of," "member-of," ...) and their transitivity-like connections, one also needs composition and union of roles [Sattler,1995].

**Definition 4.1 (Syntax of $\mathcal{ALC}^+$)**
*The* role descriptions *of $\mathcal{ALC}^+$ are built from role names with* union *$(R \sqcup S)$*, composition *$(R \circ S)$, and* transitive closure *$(R^+)$ of roles.* Concept descriptions *in $\mathcal{ALC}^+$ are defined as in $\mathcal{ALC}$, with the only difference that role descriptions can be used in value and in existential restrictions.*

For example, the relation "part-of" can be expressed by the role description $\texttt{direct-part-of}^+$, where `direct-part-of` is a role name. Now assume that we want to consider the specific part-of relations "component-of" and "member-of." In order to express not just that "component-of" is transitive, but also that a member of a component is also a component, we can use the following description for the "component-of" relation:

$$(\texttt{direct-component-of} \sqcup (\texttt{member-of} \circ \texttt{direct-component-of}))^+.$$

**Definition 4.2 (Semantics of $\mathcal{ALC}^+$)**
*An interpretation of $\mathcal{ALC}$ is extended to role descriptions in the obvious way: $(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$, $(R \circ S)^{\mathcal{I}} = \{(x,y) \mid \exists z.\ (x,z) \in R^{\mathcal{I}} \wedge (z,y) \in S^{\mathcal{I}}\}$, and $(R^+)^{\mathcal{I}} := \bigcup_{n \geq 1} (R^{\mathcal{I}})^n$.*

**Theorem 4.3** *The subsumption problem is satisfiable for $\mathcal{ALC}^+$.*

This was shown in [Baader,1991]. The algorithm is again tableau-based, but it is a lot more involved than the one for $\mathcal{ALC}$. In fact, a naive adaptation of the tableau algorithm for $\mathcal{ALC}$ would yield a non-terminating procedure. This decidability result can also be obtained by realizing that $\mathcal{ALC}^+$ is just a syntactic variant of propositional dynamic logic (PDL) [Schild,1991; Giacomo and Lenzerini,1994], which is well-known to be decidable. From the known complexity results for PDL we can deduce that the subsumption problem for $\mathcal{ALC}^+$ is EXPTIME-complete, in contrast to only PSPACE-completeness for $\mathcal{ALC}$.

## 5 Combining both extensions

The language $\mathcal{ALC}(\mathcal{D})^+$ is obtained by combining the extensions introduced in the previous two sections. To be more precise, the concept description language of $\mathcal{ALC}(\mathcal{D})^+$ is defined as follows:

**Definition 5.1 (Syntax of $\mathcal{ALC}(\mathcal{D})^+$)**
Role descriptions *are built from role and attribute names using the role-forming operators* union *$(R \sqcup S)$*, composition *$(R \circ S)$, and* transitive closure *$(R^+)$.*

Concept descriptions *are built from concept names and role descriptions using the concept-forming operators* negation *$(\neg C)$*, disjunction *$(C \sqcup D)$*, conjunction *$(C \sqcap D)$*, existential restriction *$(\exists R.C)$*, value restriction *$(\forall R.C)$, and* predicate restriction *$(P(u_1, \cdots, u_n))$. Here $C$ and $D$ are syntactic variables for concept descriptions, $R$ stands for a role description, and $u_1, \ldots, u_n$ stand for attribute chains.*

The semantics is the obvious combination of the semantics for $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALC}^+$.

If we take $\mathcal{R}$ as the concrete domain, this language is expressive enough to define concepts that are of great interest in technical applications. In fact, in this language one can express both geometric properties of objects (using numerical constraints from the concrete domain $\mathcal{R}$) and the structural decomposition of objects (using a transitive "part-of" relation which is expressed as the transitive closure of the role $\texttt{direct-part-of}^+$).

Unfortunately, the price one must pay for this expressiveness is that the subsumption problem is no

longer decidable. This can be shown by reducing the Post Correspondence Problem to the subsumption problem for $\mathcal{ALC}(\mathcal{D})^+$.

First, we recall the definition of the Post Correspondence Problem. Let $\Sigma$ be a finite alphabet. A *Post Correspondence System* (PCS) over $\Sigma$ is a nonempty finite set $S = \{(l_i, r_i) \mid i = 1, ..., m\}$ where the $l_i, r_i$ are words over $\Sigma$. A nonempty sequence $1 \leq i_1, ..., i_n \leq m$ is called a *solution* of the system $S$ iff $l_{i_1} \cdots l_{i_m} = r_{i_1} \cdots r_{i_m}$. It is well-known that the *Post Correspondence Problem*, i.e., the question whether there exists a solution for a given system, is in general undecidable if the alphabet contains at least two symbols [Post,1946].

A solution of a PCS describes a sequence of pairs of words with a previously unknown size. The varying size is represented with the help of the transitive closure on the abstract level, whereas the words are encoded as real numbers, and their concatenation is modeled by predicates of the concrete domain $\mathcal{R}$.

More precisely, words are encoded as follows. For $B := |\Sigma| + 1$, we can consider the elements of $\Sigma$ as digits $1, 2, ..., B-1$ of numbers represented at base $B$. For a given nonempty word $w$ over $\Sigma$, we denote by $\overline{w}$ the nonnegative integer (in ordinary representation at base 10) it represents at base $B$. We assume that the empty word $\varepsilon$ represents the integer 0. Obviously, the mapping $w \mapsto \overline{w}$ is a 1−1-mapping from $\Sigma^*$ into the set of nonnegative integers. Concatenation of words is reflected on the corresponding numbers as follows. Let $v, w$ be two words over $\Sigma$. Then we have $\overline{vw} = \overline{v} \cdot B^{|w|} + \overline{w}$, where $|w|$ denotes the length of the word $w$.

We are now ready to define names for the predicates of the concrete domain $\mathcal{R}$ we shall use in our reduction. For $i = 1, ..., m$,

$$\begin{aligned} C_l^i(x,z) &\iff z = \overline{l_i} + x \cdot B^{|l_i|}, \\ C_r^i(x,z) &\iff z = \overline{r_i} + x \cdot B^{|r_i|}, \\ E(x,y) &\iff x = y, \quad \text{and} \quad L(x) \iff x = 0. \end{aligned}$$

Thus, if $x$ is the encoding of the word $l_{i_1} \cdots l_{i_{k-1}}$, and if $C_l^{i_k}(x,z)$ holds, then $z$ is the encoding of the word $l_{i_1} \cdots l_{i_{k-1}} l_{i_k}$. In addition, for words $u, v$ we have $E(\overline{u}, \overline{v})$ iff $u = v$, and $L(\overline{u})$ iff $u$ is the empty word.

Let $w_l, w_r$, and $f$ be attribute names. The concept description $C(S)$ corresponding to the Post Correspondence System $S$ is now defined as follows:

$$\begin{aligned} C(S) = & \\ & \bigsqcup_{i=1}^{m} \left(C_l^i(w_l, f\, w_l) \sqcap C_r^i(w_r, f\, w_r)\right) \sqcap \\ & L(w_l) \sqcap L(w_r) \sqcap \\ & \forall f^+. \left( \bigsqcup_{i=1}^{m} \left(C_l^i(w_l, f\, w_l) \sqcap C_r^i(w_r, f\, w_r)\right) \right) \sqcap \\ & \exists f^+. E(w_l, w_r). \end{aligned}$$

In addition, we consider the concept "bottom" that is always interpreted as the empty set. Obviously this concept can be expressed by the description $A \sqcap \neg A$ (where $A$ is an arbitrary concept name).

**Theorem 5.2** *The concept description $C(S)$ is subsumed by $A \sqcap \neg A$ if, and only if, the Post Correspondence System $S$ does not have a solution. Consequently, the subsumption problem for $\mathcal{ALC}(\mathcal{D})^+$ is undecidable.*

A proof of this theorem can be found in [Baader and Hanschke,1993]. It should be noted that in the concept $C(S)$ we have used transitive closure of the attribute $f$, and this attribute also occurs in concrete predicates. Thus, it is not clear whether undecidability still holds if transitive closure is restricted to roles, which (by definition of predicate restrictions) may not occur in concrete predicates. For example, the `direct-part-of` role is usually not functional, i.e., it cannot be introduced as an attribute, and thus it must not occur in predicate restrictions.

## References


[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 452–457, Sydney, Australia, 1991.

[Baader and Hanschke, 1993] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proceedings of the 16th German AI-Conference, GWAI-92*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143, Bonn (Germany), 1993. Springer–Verlag.

[Baader and Hollunder, 1991] F. Baader and B. Hollunder. *KRIS*: Knowledge Representation and Inference System. *SIGART Bulletin*, 2(3):8–14, 1991.

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological systems, or making KRIS get a move on. *Journal of Applied Intelligence*, 4:109–132, 1994.

[Baader, 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 446–451, Sydney, Australia, 1991.

[Brachman and Schmolze, 1985] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al.*, 1991a] R. Brachman, D. McGuinness, P. Patel-Schneider, L. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rationale. *SIGART Bulletin*, 2(3):108–113, 1991.

[Brachman *et al.*, 1991b] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, San Mateo, Calif., 1991.

[Collins, 1975] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *2nd Conference on Automata Theory & Formal Languages*, volume 33 of *LNCS*, 1975.

[Davis, 1973] M. Davis. Hilbert's tenth problem is unsolvable. *Am. Math. Monthly*, 80:239–269, 1973.

[Donini *et al.*, 1991a] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge, Mass., 1991.

[Donini *et al.*, 1991b] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 458–463, Sydney, Australia, 1991.

[Donini *et al.*, 1992] F. Donini, B. Hollunder, M. Lenzerini, A. Spaccamela, D. Nardi, and W. Nutt. The complexity of existential quantification in concept languages. *Journal of Artificial Intelligence*, 53:309–327, 1992.

[Giacomo and Lenzerini, 1994] G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, pages 205–212. AAAI-Press/The MIT-Press, 1994.

[Hollunder and Baader, 1991] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 335–346, Cambridge, Mass., 1991.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 348–353, Stockholm, Sweden, 1990.

[Hollunder, 1990] B. Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *14th German Workshop on Artificial Intelligence*, volume 251 of *Informatik-Fachberichte*, pages 38–47, Ebingerfeld, Germany, 1990. Springer–Verlag.

[Jaakola, 1990] J. Jaakola. Modifying the simplex algorithm to a constraint solver. In P. Deransart and J. Małuszyński, editors, *Proceedings of the International Workshop on Programming Language Implementation and Logic Programming, PLILP'90*, number 456 in Lecture Notes in Computer Science, pages 89–105. Springer Verlag, 1990.

[Jaffar *et al.*, 1992] J. Jaffar, S. Michayov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[MacGregor, 1991] R. MacGregor. Inside the LOOM classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[Matijacevič, 1970] Y. Matijacevič. Enumerable sets are diophantine. *Soviet Math. Doklady*, 11:354–357, 1970. English translation.

[Mays *et al.*, 1991] E. Mays, R. Dionne, and R. Weida. K-Rep system overview. *SIGART Bulletin*, 2(3):93–97, 1991.

[Peltason, 1991] C. Peltason. The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119, 1991.

[Post, 1946] E. M. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52:264–268, 1946.

[Sattler, 1995] U. Sattler. A concept language for an engeneering application with part-whole relations. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *Proceedings of the International Workshop on Description Logics*, pages 119–123, Rome, 1995.

[Sattler, 1996] U. Sattler. The complexity of concept languages with different kinds of transitive roles. In *20. Deutsche Jahrestagung für Künstliche Intelligenz, KI'96*, Lecture Notes in Artificial Intelligence. Springer Verlag, 1996. To appear.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sydney, Australia, 1991.

[Schmidt-Schauß and Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Journal of Artificial Intelligence*, 47:1–26, 1991.

[Tarski, 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. U. of California Press. Berkley, 1951.

[Weispfenning, 1988] V. Weispfenning. The complexity of linear problems in fields. *J. Symbolic Computation*, 5:3–27, 1988.

# The Role of Formal Knowledge Representation in Configuration

**Hans-Jürgen Bürckert, Werner Nutt, Christian Seel**
German Research Center for Artificial Intelligence - DFKI GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
e-mail: {hjb, nutt}@dfki.uni-sb.de

## 1 Introduction

Configuration is a traditional application of Artificial Intelligence techniques. However, in contrast to related areas like Planning, there are only few attempts to provide a formal definition of configuration problems and to solve such problems with formally well-founded methods. One reason to change this situation is related to the methodology of doing research.

- A formalization of configuration problems and solutions facilitates the communication of results within the field and with other areas. The degree to which such a formalization has been achieved can be seen as an indicator for the degree of maturity of the field.

In Knowledge Representation (KR) the typical approach is to formulate real world problems as inference problems in a suitable logic, e.g., for a given expression, decide whether it is satisfiable, whether it is a consequence of other expressions, find a model for it, or find a particular kind of consequence.

In the area of configuration, such contributions are rare up to now and they only describe selected aspects of implemented systems (see e.g. the abstract formulation of structure oriented and resource oriented configuration in [5] or the constructive problem solving (CPS) model of [2]) describing configuration as model construction.

One of the advantages to be gained from this approach is that issues like correctness and adequateness of a representation can be discussed independently of the implemented solution procedure. For example, it is reasonable to expect that generic configuration problems are NP-hard, but one should be skeptical with a formalization where configuration problems are no more in the class NP: it might be difficult to find a solution, but the size of a solution should be polynomial in the size of the specification and it should be "easy" to verify that a structure satisfies the specification.

A model of configuration problems should be the basis for designing an adequate domain representation language and for developing a solution procedure for the addressed configuration tasks. One can evaluate the procedure with respect to the representation and determine properties like correctness and completeness.

As an additional advantage, it will be easier to compare questions in configuration with questions in other areas and to transfer results.

In the sequel we will give some more evidence for our argumentation here. We briefly sketch the model, some results and problems given by Najmann and Stein [5]. We then recall and discuss the CPS model of Buchheit, Klein, and Nutt [2]. Finally we summarize some results by one of the authors [6], who used the latter approach and compared the usage of a general KR system CLASSIC [1] and of the CPS system [2] for customizing*retail information systems.*

## 2 The Model of Najmann and Stein

In Germany two different views of configuration systems have been established. The *resource oriented* approach is the basis of systems like COSMOS [4] or MOKON [7] while the PLAKON system [3] is based on a *skeleton* or *structure oriented* approach.

Najmann and Stein [5] developed a formalizations of both resource oriented and skeleton oriented configuration problems as mathematical structures which allows a comparison of the two approaches. As a main result they show that—from their theoretical point of view—there is only a minor difference, since the skeleton ori-

ented systems turn out as a special case of resource oriented ones.

In the following we will briefly sketch their model and discuss some of the problems with their abstraction. In their model Najmann and Stein describe configuration systems as mathematical structures with seven components:

- a set of objects,
- for each object:
  - a set of properties (functionality-value pairs),
- a set functionalities,
- for each functionality:
  - a set of values,
  - an addition operator,
  - a test,
- a set of demands (functionality-value pairs).

The configuration process is modeled as a finite sequence of compositions of objects controlled by the addition operators of the functionalities. Solutions are determined as configurations that fulfill the demands which is approved by the tests. With slight adaptation and less formally we will summarize their approach through a more convenient "object-oriented" definition of configuration problems.

A configuration problem consists of three main parts: the components, the description of how the components can be composed to configurations, and the specification of when a composed configurations is a solution.

The components are given as *objects* with *attributes* and *values*. The attributes are functional (i.e. single valued) with associated sets of admissible values. They reflect the functionalities of the objects. Najmann and Stein do not specify the values in detail. Thus there may be concrete values like strings or integers and there may be abstract values, which may be objects again—although the description suggests that they address concrete values only. For a configuration several "copies" of the objects (with potentially different values for their attributes) may be composed (see below).

The *operators* describe how objects can be composed, in order to obtain configurations. An operator is a partial function on the set of admissible values of an attribute. A composition of two objects—in slight adaptation of Najmann and Stein's definition—can be considered as a new object which inherits all attributes of the composed objects. If the two objects have an attribute in common (with different or the same values), the operator for that attribute computes a new admissible value from the given ones, that will hold for the composition. A configuration is a finite sequence of compositions, and thus may again be considered as an object.

The *tests* specify when such a configuration is a solution with respect to given *demands*. Tests are partial boolean functions on the admissible values of an attribute. The set of demands consists of pairs of attributes and values. A test for an attribute approves whether in the composed configuration the value of that attribute is admissible with respect to the demanded value for that attribute. Thus the tests decide when a configuration satisfies the demands, in order to be a solution to the given configuration problem.

This model is an abstraction of resource oriented configuration problems, which models the resource dependencies of configurations. In extension to that Najmann and Stein's model for skeleton oriented configuration problems has in addition *rules* that specify the structural dependencies between components in the sense that they express the requirement for the existence of (alternatives of) components, if certain other components are already included. However, the above model is abstract enough to express structural dependencies by new functionalities, operators and tests, which by the given level of abstraction is of course not as surprising as the authors mentioned. The skeleton rules are simple constraints on the admissibility of configurations and hence it is not really surprising that that information can be coded into suitable test functions, since those are not restricted in any way.

Because of the high abstraction describing configuration problems as mathematical structures, the model has some deficiencies. For instance, it is rather difficult to see for a concrete system whether and how it fits into the model. Furthermore, it is hardly possible to use that model as a basis for comparing configuration problems with other problem solving areas and hence to transfer results between these areas (which, however, was not the intention of the authors). As tests and operators are mod-

eled as abstract functions, questions concerning the solvability (of tests) or the composability (by operators) cannot be addressed adequately. The model does not allow to distinguish the process of configuration and its result, the configured object. For that reason properties like correctness etc. of the configuration process cannot be studied. Aspects like strategic and control information aren't captured either. It also does not provide a description of the requirement specification of specific configuration tasks to be solved. Since configuration problems are modeled as finite structures, configurations with arbitrary values are not captured.

The approach of Najmann and Stein has turned out useful as it allowed some discussion and comparison of resource and skeleton oriented approaches by polynomial reductions between the two models. Furthermore Najmann and Stein could show that checking for the existence of a solution is NP-complete for the two models and therefore computing a solution and computing cost bounded solutions is at least NP-hard.

The formalization of Najmann and Stein is a simple and abstract model of configuration problems, which to some extent already models the "logic" of configuration systems: The iterative configuration process can be seen as an abstraction of a *logical inference* that composes a configuration with a set of functionalities. A configuration is a solution, if it *satisfies* the demands, i.e. if it is a *model* of the set of demands. In the next section we will discuss a more concrete formalization of configuration as logical inference process.

## 3 The Model of Buchheit, Klein, and Nutt

In contrast to the above model Buchheit, Klein, and Nutt [2] explicitly describe a configuration problem as an inference problem. They distinguish between a (declarative) representation of the domain and task knowledge of a configuration problem on the one hand and a (rule-based) operationalization of a configuration problem as an (abductive) inference process selecting and composing the components in order to solve the specified configuration task on the other hand. In their model they provide the general structure and properties of components and general constraints for structural composition of admissible configurations (summarized as the *domain knowledge*) as well as the specification of requirements for a specific configuration task (the *task knowledge*). The inference process has to take into consideration and to approve the constraints for selection and composition, namely

- the general conditions for composing components in order to obtain an admissible configurations at all, and
- the specific requirements for selecting and composing components to a solution satisfying the required functionalities of the configuration at hand.

Such an approach allows the comparison of configuration systems according to their modeling of the configuration domain and their realization of the configuration process itself as well as general investigations about representation formalisms and inference processes and their adequateness for configuration problems.

Formally the knowledge of the configuration domain is given as a set of logical formulas (in an adequate logical language) describing the components and their functionalities and the constraints about their composability. The requirement specification of a specific configuration task is again a set of logical formulae. A solution of the configuration problem is then a model of the configuration domain satisfying the requirement specification. Of course, it is convenient to describe a solution not as an arbitrary logical model of these sets of formulae, but in terms of the names of the components and their properties. In that sense, such a solution can be seen as a (partial) Henkin or Herbrand model of the configuration domain and the requirement specification.[1]

The inference process is an *abductive* inference process that generates a (description of a) partial Henkin model: Given both the domain knowledge $D$ and the requirement specification

[1] That means, the universe of the model are the ground terms of the logical language. Since the formulae must not be given in clause form it is not required that all implicit object (coded through existential quantifiers) are named by constants or by ground terms (so-called witnesses) as it is usual for Herbrand or Henkin models. Therefore such a model will not necessarily be described by a set of ground atoms. More complex ground formulae may be allowed, for example, the implicit objects have still to be coded through existential quantifiers. Furthermore not all properties may be of interest and hence be reported in the description of a solution. For those reasons we will speak of a *partial* Henkin model.

$R$ over the same logical language $\Sigma$ a configuration solution $C$ is a partial Henkin model (characterized by a set $S$ of ground formulas) over a sublanguage $\Sigma_0$ of $\Sigma$, such that $C$ satisfies the domain knowledge $D$ and the requirements $R$, i.e., $C \models D$ and $C \models R$. Slightly more detailed Buchheit, Klein, and Nutt also differentiate definitional knowledge describing the structure of the configuration domain (as types of components and their functionalities and relationships)[2] and integrity constraints expressing necessary conditions which components of a configuration have to satisfy or ruling out certain combinations of components as impossible. They also define a solution as *an arbitrary* $\Sigma_0$-model satisfying the above conditions. This, however, seems to be to abstract and consequently it turns out that their inference procedure in fact constructs partial Henkin models in terms of sets of ground formulas over $\Sigma_0$.

Buchheit, Klein and Nutt exemplify their approach in detail by providing a sample domain representation language and for that language a rule-based, tableaux-like calculus for constructing configurations. They show properties of that calculus like model preservation of the rules and correctness of the calculus. They mention that this calculus is a conceptual result, which is not to be used as an implementation specification, since strategies and control has to be added as well as user interaction. Thus their model of a configuration problem as "the task to construct for a given specification, which is understood as a finite set of logical formulas, a model that satisfies the specification" has still some limits. Certainly one can use the CPS system directly as a configuration system. However, its language is too expressive, in the sense that the underlying calculus is undecidable. Nevertheless the approach demonstrates that such an abstract view is useful in providing the ability of more detailed general investigation of properties of configuration problems. It also demonstrates that a formal view of the knowledge representation of a configuration system together with a formal description of the configuration as an inference process allows much more insight in those properties.

[2]Notice, that this knowledge is definitional in the sense that it can be used in order to uniquely extend a $\Sigma_0$-model to a $\Sigma$-model.

Based on such an abstraction a configuration system can be developed on a much more serious basis than the usual more experimental approaches: Starting with an analysis of the configuration domain and the requirements of the representation model, we can design a suitable representation language for the components, their functionality and their relationships as well as for the constraints to admissible configurations. Here several alternatives can be taken into account and they can be compared on a formal and solid basis. An abstract calculus can be derived for realizing the configuration process. Again properties of that calculus can be investigated on a formal and solid basis. Finally based on that results common software development techniques can be applied to specify and realize the configuration system with strategic and control information, interaction needs for the user etc.

The CPS approach has been implemented prototypically at DFKI. It has been instantiated and tested for some applications, e.g. with sample knowledge bases for PCs and for programmable logic controls. In the next section we sketch another application, where the approach has been applied to a problem which is not a classical configuration task.

## 4 A Summarized Comparative Study

In his diploma thesis [6] one of the authors investigates the usage of configuration approaches for customizing retail information systems. Based on the CPS model of Buchheit, Klein, and Nutt a configuration approach to that problem is considered. In collaboration with a software house the approach has been implemented as part of the thesis. We briefly recapitulate one of the results.

One aim of the thesis was a comparison of the usage of a classical knowledge representation system on the one hand and a special configuration system on the other hand. Based on an analysis of the domain requirements of customizing those retail information systems, the CLASSIC knowledge representation system [1] and—because a standard configuration system has not been available—the CPS approach [2] as potential configurators are compared according to the following three crucial criteria, language, functionality, and interactivity.

**Language.** The main question was, whether the language provides the constructs needed to configure such retail information systems. The development of the sales domain showed that we need constructs, which can describe a decomposition tree as well as external effects on the the tree's form. Taxonomic relations and part-of relations have to be modeled, and constraints for describing the real part-of-relations are needed. In addition rules for describing the external effects are necessary.

Since the two languages of both CLASSIC and CPS provide all these facilities there is no conceptual difference between the two. Differences, however, come up, when the detailed syntax is considered: the rules of CLASSIC are to restricted and it does not allow for modeling disjunctive information, e.g. for alternative configurations. Therefore CPS came up to be more suitable for modeling customization of retail information systems, and of configuration in general.

**Functionality.** It turned out that CLASSIC originally is developed as a knowledge classification system. That means the classifier is the heart of the system, which, however, does not play the important role in configuration. Much more important is here the functionality for proving constraints and for applying rules. A classical, purely rule-based expert system would on the other hand also not be the optimal choice, since much information had to be described as taxonomic and partonomic information, which would not be that comfortable with rules.

The two systems have turned out to be nevertheless similar suitable for configuring and parameterizing the retail information systems form the given specifications. The main difference again comes from the difference in languages. While CLASSIC does not solve nondeterminism adequately, since it does not support disjunctive modeling on the class level, CPS tries to evaluate all possible solutions for the configuration task at hand. CLASSIC would need a suitable extension through its implementation language, in order to support this.

**Interactivity.** Here clearly CLASSIC had some advantages. The user can interactively influence the configuration process by proposing modification of the solution. Currently this is unfortunately only supported by textual input. A graphical extension for those interactions should, however, be easily possible. In addition CLASSIC's inference engine provides explanations, if the user is interested in. Similar extensions for CPS are currently under development. Other useful extensions are facilities for incremental inspection both of the domain model and of the configuration solutions.

## 5 Conclusion

We have discussed and compared two abstract models of configuration systems and sketched an application of the idea behind one of these models for a comparison of suitability of two different approaches to develop a configuration system.

The reason was to demonstrate the advantage of such a formal basis for the field of configuration systems and the necessity of such a direction from a research-methodological point of view.

A completely different motivation for providing a formal basis to the field is related to new potential applications of configuration systems.

- In the future, more and more products will allow for a sufficient number of variants such that for every customer an individual solution can be provided. Thus, configuration will become ubiquitous in sales and production. As a consequence, software systems for configuration will no longer be customized for one particular application, but will solve generic problems.

A system that is intended as a platform for many different applications has to implement an abstract model of configuration. This does not necessarily imply that such a model has to cover all possible kinds of configuration applications.

On the contrary, it is likely that a situation will emerge similar to the one for databases, which are a widely used kind of generic software. Data models and query languages can be described logically. But different models (e.g., the relational, the multidimensional,the object-oriented) are suited for different classes of applications.

## References

[1] R. Brachman, D. McGuiness, P. Patel-Schneider, L. Resnick, and A. Brogida. Liv-

ing with CLASSIC, when and how to use a KL-ONE like language. In *Principles of Semantic Networks.* Morgan Kaufmann, 1990.

[2] M. Buchheit, R. Klein, and W. Nutt. Configuration as model construction: The constructive problem solving approach. In *Proc. of the 3rd International Conference on Artificial Intelligence in Design, AID'94*, 1994.

[3] R. Cunis, A. Günter, and H. Strecker. *Das PLAKON Buch.* Informatik-Fachberichte 266, Springer, 1990.

[4] M. Heinrich and E.W. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems from Modular Components In *Proceedings of CAIA'91*, pp. 257–264, 1991.

[5] O. Najmann and B. Stein. A theoretical framework for configuration. In *Proceedings of the 5th IEAAIE*, 1992.

[6] C. Seel. Wissensbasierte Konfiguration von Warenwirtschaftssystemen. (German) Diploma Thesis. University Saarbr"ucken, 1996.

[7] B. Stein and J. Weiner. MOKON. Internal report SM-DU-178, Uni Duisburg, 1990.

# Multiple Part-Hierarchies

**Anne Engehausen, Simone Pribbenow, Ulf Töter**
Universität Hamburg
Fachbereich Informatik
Vogt-Kölln-Straße 30, 22527 Hamburg
e-mail: {lengehau, pribbeno, ltoeter}@informatik.uni-hamburg.de

**Abstract:** The representation of knowledge about objects is necessary for most configuration tasks. Especially important for most applications is extensive knowledge about the compositional or part hierarchy of the enties to be configured. The goal of this contribution is to discuss the problems arising by modeling the part hierarchy of objects. If formalisms like terminological logics should be used, it is necessary to consider the specific properties of the part-whole relation. In this article, we first mention the formal regularities of the relation and the differentiation into different kinds of part-whole relations. In the remainder, we will concentrate on the question what kind of ontological entites could be parts of objects. The point is discussed for configuration of all kinds of buildings , e.g., the design or modification of the lay-out of flats, houses, and so on. The discussion leads to an ontology of design objects and to multiple views on one and the same entity.

## 1 Knowledge about parts

The representation of knowledge about objects is necessary for most configuration tasks. Compositional and taxonomical relations are used to model the application domain and to form abstraction principles claimed as one way to guide the search through huge configuration spaces. Especially important for most applications is extensive knowledge about compositional relations (cp. [Biundo et al. 93], [Cunis et al. 91]) which are often called "part-whole" or "part-of" relations in knowledge representation contexts. These compositional or "part-whole" relations build up hierarchies, sometimes called "partonomies" in parallel to "taxonomies" representing "is-a" hierarchies. Part hierarchies or partonomies normally form trees because the whole object is supposed to be divided into disjunct, non-overlapping parts, which themselves are divided in the same way and so on.[1] If a formal representation of partonomies or part hierarchies should be given, e.g., by using terminological logics, some problems must be considered.

First, at least some of the *formal properties* of the part-whole relations must be guaranteed. For example, the transitive closure of the part-whole relation must be available through inference. Otherwise it is not possible to compute indirect relations which are nevertheless important in most of the configuration tasks. As the contributions of Baader and Sattler in this volume show the property of transitivity is not easy to implement in terminological logics. Other properties, which are interesting, are the irreflexivity and asymmetry on the individual level, while part-whole relations on the conceptual / terminological level are reflexive and antisymmetric. On each level it is normally assumed that objects are only the same if they do not only have the same parts but also share the same spatial and functional relations between these parts. Most of the mentioned properties cannot be provided in an explicit way by terminological logics but must be "build-in" in specialized formalisms. On the long run, it might be more flexible and adequate to use a formalism that can express and handle properties of relations in a more general way.

Second, a detailed analysis shows that there is more than one kind of part-whole relation. This assumption was first made to cope with various problems of intransitivity. As a result, several *classifications of part-whole relations* were developed. Perhaps the most important one is that of Winston, Chaffin and Hermann [Winston et al. 87] which was designed from a linguistic point of view. Different part-whole relations require different ways of processing them. Therefore some knowledge based systems were build up in the last years that use more than one kind of part relation, e.g., the approaches of [Uschold 95] or [Markovitz et al. 92]. A domain independent and processing oriented approach for a classification of part-whole relations is given in [Gerstl & Pribbenow 95]. The approach assumes two different classes of parts which are represented and/or processed in different way. The first class are the so-called "structure dependent" parts that are given by the internal structure of an entity, e.g., the components of a car. As those decompositions are permanent the resulting part-whole relations belong to the conceptual knowledge of the decomposed entity. The second class of parts are constructed parts which could be computed by using internal features like color or external schemes like spatial frames for partitioning an object. Examples of such parts are "the metal parts of her car" or "the left

[1] Please note that the same do not hold for the mereological "part-of" or "proper part-of" relation, which was invented as an alternative to set theory (compare for example [Simons 87] for an overview of Classical Extensional Mereology). The transitive and asymmetric mereological "proper part-of" relation forms a lattice without zero element because there always exists a (unique) mereological sum for any non-empty class of existing individuals.

half of his car". The referred parts are temporary constructions and do not belong to domain knowledge; they are computed by construction processes. A first approach to model some of this different relations in a rich terminological logic is described in [Gerstl 93].

Third, it is not really clear what kind of entities could be parts of a configuration. Normally parts are three-dimensional, solid objects like the back, the seat and the legs of a chair. In many domains it is sufficient to consider only solid objects as entities. For modeling the parts of a car this is not enough because it includes liquid entities like radiator water which must be considered as mass and even non-material parts like the interior. In the domain of rooms, flats, and buildings this problem is even more convincing. We will discuss this problem and possible solutions in the following section.

## 2 Considerations about buildings

For the design of flats, houses, factories, and so on, the world knowledge needed for the configuration task includes part hierarchies of buildings. There are at least two kinds of entities that should be considered. Three-dimensional, solid parts as walls, floors, and roofs are important if you try to build, rebuild, or modify a building. "Induced", that means only dependent from material objects existing entities as interiours and surfaces are important for designing flats or functional buildings. We will show that the representation of such different ontological entities can be achieved by using *multiple partonomies*. As preparation, we will have a look at different possibilities to describe buildings or flats by single partonomies.

A (minimal) example is given in figure 1 that shows a flat consisting of two adjacent rooms, a door between them and the five walls, that build up the rooms. Is it possible to give one partonomy that describes the situation in a correct and adaquate way? The first step to answer this question is to decide what kinds of entities should be taken into account.

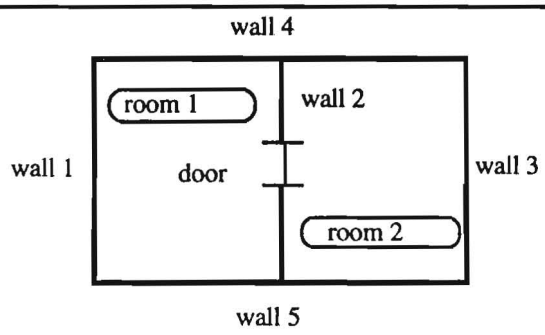

Figure 1: Floor plan of a flat

Alternative A:

The first possibility is to look at material objects only. In our example, this ontological decision reduces the flat to five walls and a door. The two rooms cannot be addressed as objects having certain properties like size or shape. In that situation it even does not make sense to speak of a two-room-appartment. Therefore, alternative A is only of limited use and will therefore not be further analyzed.

Alternative B:

The second possibility is to make no distinction in representation and processing between rooms and material objects like walls and doors. Figure 2 shows the resulting partonomy which simply divides the flat into the two rooms, the five walls and the door.[2] No information about the relations between rooms and walls is available, e.g., which walls surround each of the rooms. The missing information leads to a crucial problem: You can remove wall 2 without melting the two rooms into one! The reason is that the representation deny the dependencies of rooms from walls and treated rooms as independent entities. If alternative B is used correctness with respect to rooms cannot be garanteed automatically but must be checked after each addition or deletion of walls. Local modification are inefficient and easily faulty if such kinds of partonomies are used.

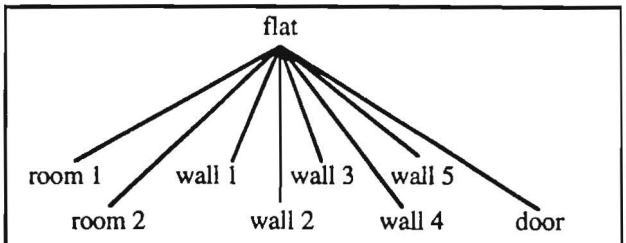

Figure 2: Partonomy of the flat for alternative B

Alternative C:

The next possibility focusses on rooms as primary objects. The flat is divided into the two rooms which themselves are built up by the walls that surround them. For example, room 1 is built up by wall 1, wall 2, and the left halves of wall 4 and wall 5. To achieve that, wall 4 and wall 5 must each be divided into parts a and b which belong to different rooms (see figure 3). This view causes two problems. First, the information, that each wallpart a together with each wallpart b form a complete wall is lost. Second, the whole wall 2 is used to build up the two different rooms. Wall 2 cannot be divided into wallparts in a natural way like wall 4 or wall 5; a longitudinal separation would be possible but highly artificial. The resulting partonomy has the form of a graph and not of a tree because wall 2 belongs to both rooms. The effect is that the two rooms overlap which in fact is not true. A formal representation modeling the two rooms as disjunct, non-overlapping

[2] Another possible partonomy for alternative B represents the door as part of wall 2 as shown in figure 3. This slight modification does not solve the basic problems of alternative B.

entities is not possible. The problem of material objects like walls and false ceilings belonging to more than one "induced" object like rooms or floors multiplies if buildings are considered.

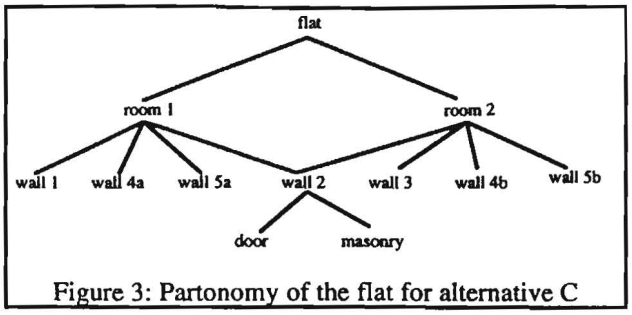

Figure 3: Partonomy of the flat for alternative C

## 3 Ontological considerations and multiple partonomies

The problem to build up one single partonomy is caused by the different ontological status of the entities involved in the example. The door and the walls are solid, three-dimensional objects while the rooms are only cavities, that means "bounded hollow spaces". As mentioned above a room can only exist if there is at least one wall to bound it (like the internal space of a ball). So, in opposition to *solid objects*, that can exist by themselves, cavities can be classified as *derived entities*, that are induced by other objects and cannot exist on their own. In addition to *cavities* the class of derived objects contains *surfaces* (two-dimensional), *lines* (one-dimensional) and *points* (null dimensions) that can be summed up as boundaries. Figure 4 shows all the entities hierarchically structured in an ontology.

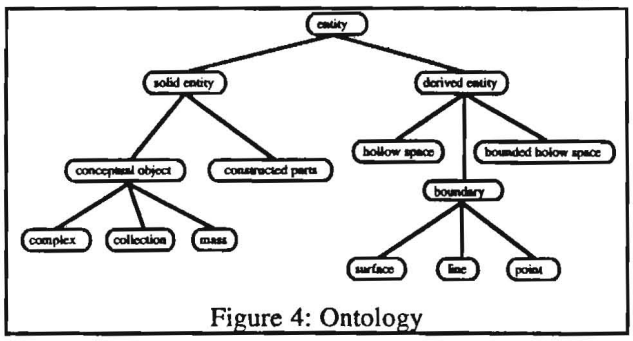

Figure 4: Ontology

There has already been made an approach to distinguish between different kinds of objects by [Landau & Jackendoff 93], whose research about "names for things" in natural language leads to the distinction between *solids* that are three-dimensional, solid objects, *containers*, e.g. a cup or a box, *negative objects* like a scratch and *surfaces*. Note, that Landau and Jackendoff use the notion of a *surface* for three-dimensional objects that are conceptualized as two-dimensional like a sheet of paper or a record.

The result of considering the ontology given in figure 4 in constructing the partonomy of the flat is shown in figure 5. A room is now regarded as a bounded hollow space with the surfaces of the surrounding walls serving as its boundaries. To achieve this effect, the surfaces of the solid object, in our example of the walls and the door, must be introduced as further relevant (derived) parts. Then, it is possible to describe the surfaces of the walls, that are visible from the rooms. The wall-surface 2.1 including door-surface 1 and mansonery-surface 2.1 belongs to room 1 and wall-surface 2.2 including door-surface 2 and mansonery-surface 2.2 is part of room 2. The solid, material wall 2 itself belongs to neither of the rooms but only to the solid parts of the flat itself.

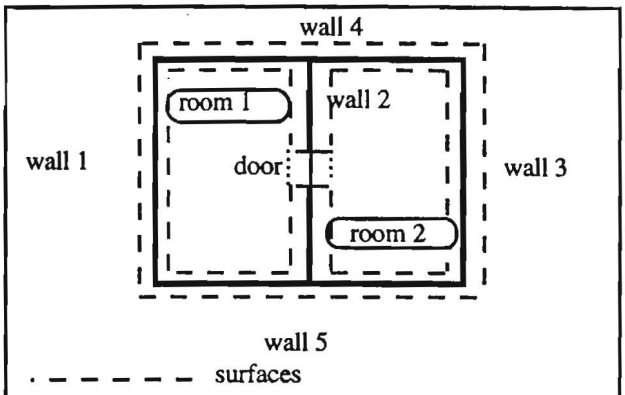

Figure 5: Flat with solid and derived parts

Following these considerations, a single partonomy could be given like the one shown in figure 6. Unfortunately, for many objects derived and solid parts exist. Therefore most levels are a mixture of ontologically different entities. Additionally, the new hierarchy resembles more a lattice structure than a tree because the surface parts are shared between rooms and walls. This does not cause the same problems than the two rooms sharing wall 2 in alternative B described above. Wall 2 as a material object causes an overlap of the sharing objects while the surfaces as two-dimensional, derived objects behave neutral. Nevertheless, an adequate partonomy is normally supposed to be a tree as we stated in section 1.

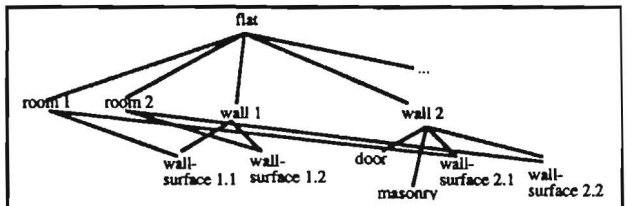

Figure 6: Mixed partonomy of the flat

One possibility to avoid this mixture is to build up two partonomies for the flat, one for a *solid view* and one for a *derived view*, according to the distinction between solid and derived objects. Two possible partonomies are shown in figure 7a and b; the derived partonomy is presented in a reduced version, where only a few of the wall-surfaces are listed. Please note, that only the two partonomies together give a complete representation of the whole flat.

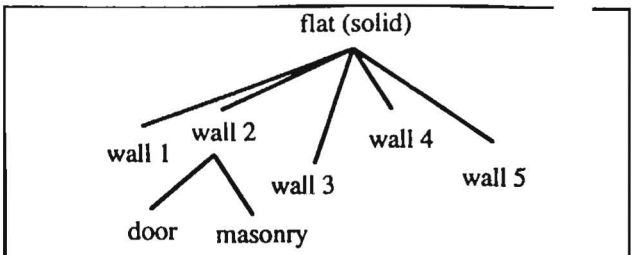

Figure 7a: Solid partonomy of the flat

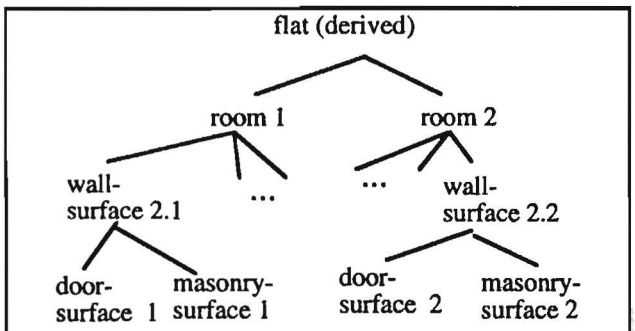

Figure 7b: Derived partonomy of the flat

To model multiple partonomies it is necessary to use a representation language able to handle views, e.g. the BHIPS concept-hierarchy representation language described by Cunis in [Cunis et al. 91]. In BHIPS the two views on flats or buildings are similar to view-overlapping concepts with a fixed linking of views, in our case the derived and the solid one. Contrary to what we will present in the next section, there seem to be no possibility to model the existing connection between the two views.

## 4 Interaction of multiple partonomies

If an object is described by multiple partonomies in different views, it is normally necessary to have some kind of correlation between the different hierarchies. One need for correlation is caused by constraints that interrelate between components or features described in different partonomies. If, for example, the breadth of the door should be exceeded, the effect on other solid objects must be computed. A larger door would also lead to larger door-surfaces and this information has to be processed by some kinds of links connecting the different partonomies. In general, there are two possibilities to represent the correlation: a declarative and a geometric one. A declarative representation explicitly names every single relation between a material object and its derived parts, e.g., the relation between all derived surfaces of a wall. A geometric representation as a spatial model of the domain implicitly contains all relations between solid and derived objects.[3] Special procedures are needed to make them explicitly available. The geometric solution is recommended if a spatial task is carried out which requires some kind of arrangement model anyway.

The remainder of this section presents a short description of our system Teigar, that models the part structure of solid objects and processes concerning conceptual and constructed parts (for a more detailed description compare [Pribbenow 95]). Figure 8 show the system which is composed of a propositional and a diagrammatic module. The propositional module realizes the concept hierarchy, the partonomies belonging to the concepts and (upward and downward) inheritance reasoning between parts and wholes about attribute values. The diagrammatic module is based on a CAD-model that handles two- or three-dimensional spatial models of individual and certain generic objects.

This hybrid system provides a basis for both correlation possibilities, the declarative and the geometric one. At the moment, only the declarative one is implemented. So-called "association slots" are used to connect derived entities with the solid objects that induce them (cf. [Engehausen & Töter 96]). The planned geometrical solution will use the CAD-models of objects as analog spatial models. Using these models, it is possible to detect boundaries (easy), e.g., the surfaces of walls, and hollow spaces (not always easy), e.g., the rooms of a flat. Once the derived objects are computed, it is possible to combine derived and solid partonomies of an object .

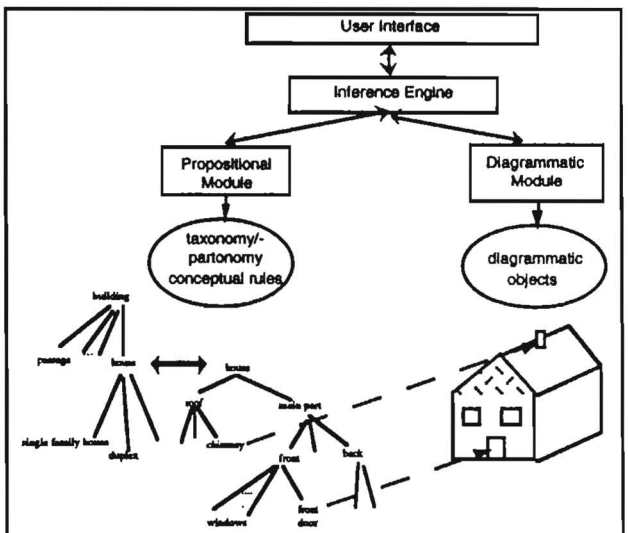

Figure 8: Architecture of the hybrid system

## 5 Configuration of flats

The solid view of the flat is more important for all tasks that deal with construction issues. For example, it can be used as a basis for computing the thermal insulation of different walls out of their material and thickness. The derived view showing rooms and surfaces can be used for designing flats or the furnishings of rooms. If the multiple views introduced in section 4 are

[3] An analysis of the role of diagrams and other analog spatial formalisms in the representation of partonomies is given in [Habel et al. 95].

used the representation of the flat can serve for both kind of tasks at the same time. It is not necessary to construct one representation for constructing issues and another one for design.

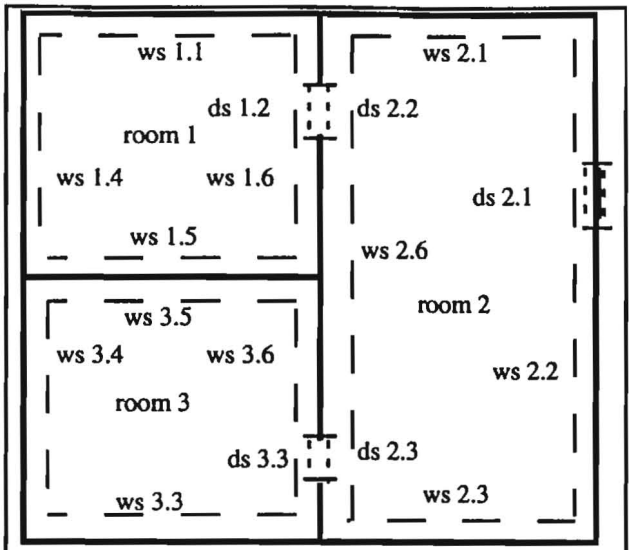


Figure 9a: Geometrical model of a three-room-flat

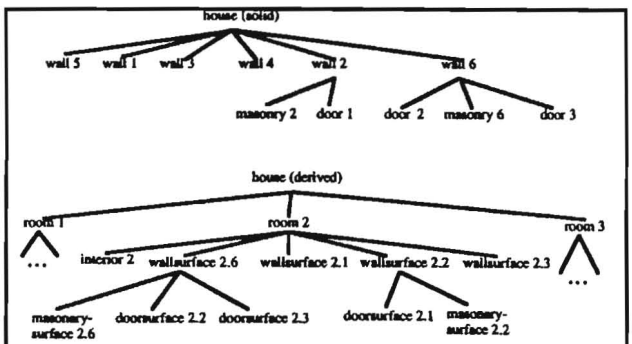


Figure 9b: Solid and derived view for the three-room-flat

As the following example shows, the proposed representation is suitable also for (more or less) realistic examples. Figure 9a and b shows a three-room-flat and the two partonomies, representing the solid and the derived view. The partonomy does not use wallparts in order to represent the rooms, in contrary to alternative C described section 2. Therefore it is easy to represent the splitting of room 2 into two new rooms (figure 10a). In all approaches dealing with wallparts this modification would lead to an extensive changes in the partonomy. However, the use of multiple views allows for local modifications that do not affect parts taken over from the former partonomies (figure 10b). That shows that multiple partonomies combined with a geometric model can be used as a flexibel and versatile formalism in the domain of design and construction of buildings.

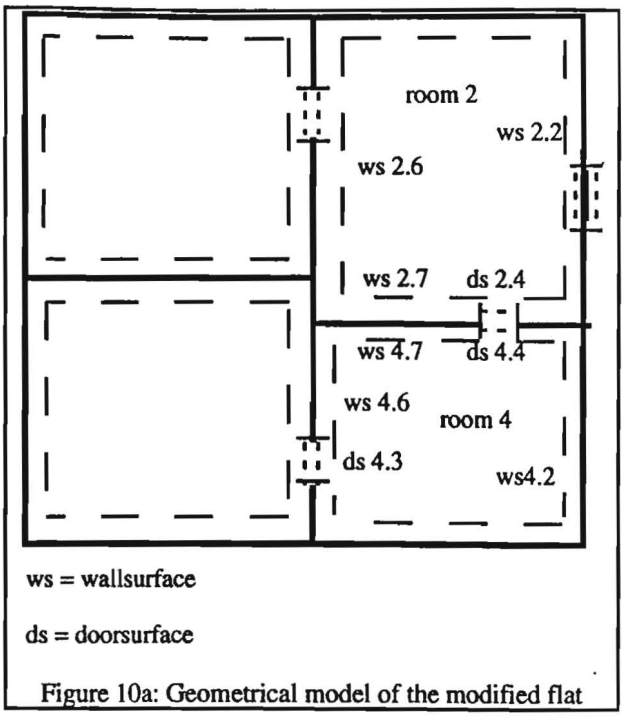


Figure 10a: Geometrical model of the modified flat

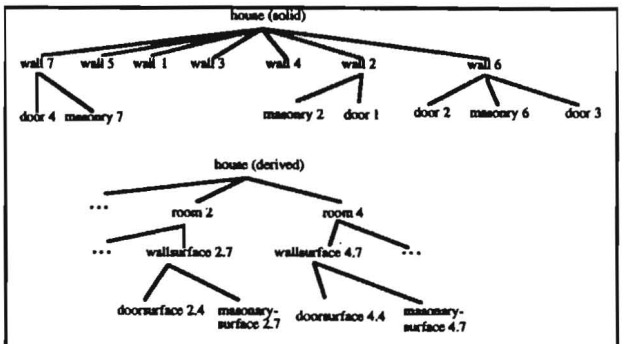


Figure 10b: Solid and derived view for the modified flat

## Literature

[Biundo et al. 93] Biundo, S. / Günter, A. / Hertzberg, J. / Schneeberger, J. / Tank, W. (1993): Planen und Konfigurieren. in: Görz (Hrsg.): Einführung in die Künstliche Intelligenz. Addison Wesley

[Cunis et al. 91] Cunis, R. / Günter, G. / Strecker, H. (1991): Das PLAKON-Buch. Informatik Fachberichte Nr. 266, Springer Verlag

[Engehausen & Töter 96] Engehausen, A. / Töter, U. (1996): Implementation eines Systems zur Verarbeitung multipler Repräsentation. Domäne: Gebäude. Studienarbeit, Universität Hamburg

[Gerstl 93] Gerstl, P. (1993): Die Berechnung von Wortbedeutung in Sprachverarbeitungsprozessen: Possessivkonstruktionen als Vermittler konzeptueller Information. Dissertation, Universität Hamburg

[Gerstl & Pribbenow 95] Gerstl, P. / Pribbenow, S. (1995): Midwinters, End games, and Bodyparts: A classification of part-whole relations. *International*

*Journal of Human and Computer Studies, Vol. 43*, 1995

[Habel et al. 95] Habel, Ch. / Pribbenow, S. / Simmons, G. (1995): Partonomies and depictions: A hybrid approach. In: J. Glasgow, H. Narayanan & B. Chandrasekaran (Hrsg.): *Diagrammatic Reasoning: Cognitive and Computational Perspectives.* Cambridge, MA: MIT-Press. 627-653.

[Landau & Jackendoff 93] Landau, Barbara / Jackendoff, Ray (1993): "What" and "where" in spatial language and spatial cognition. *Behavioral and Brain Sciences, 16.* 217-238

[Markovitz et al. 92] Markowitz, J. / Nutter, T. / Evens, M. (1992): Beyond IS-A and Part-Whole: More Semantic Network Links. in: F. Lehmann (ed.): Semantic Networks. Pergamon Press, 377-390

[Pribbenow 95] Pribbenow, S. (1995): Modeling Physical Objects: Reasoning about (different kinds of) Parts. Proc. TSM'95 (Time, Space and Movement), Bonas, France

[Simons 87] Simons, P. (1987): Parts. A Study in Ontology. Clarendon Press: Oxford

[Uschold 95] Uschold, M. (to appear): The Use of Ontology to Guide Naive Users in Representing Substructure. *Data and Knowledge Engineering*, special issue on part-whole relations

[Winston et al. 87] Winston, M. / Chaffin, R. / Herrmann, D. (1987): A Taxonomy of Part-Whole Relations. *Cognitive Science 11*, 417-444

# Knowledge Representation for Configuration Systems

**Andreas Günter**

Technologie-Zentrum Informatik, Universität Bremen
guenter@informatik.uni-bremen.de

## Introduction

Configuration is one of the fields in expert system technology in which the application of AI-methods has advanced a great deal over the past few years.

To solve a configuration task means to compose a system (configuration) from single components which meets all requirements. Configuration tasks have the following characteristics [Günter 1991; Günter et al. 1992]:

- large solution space
- objects are composed of components
- dependencies between the objects
- heuristic decisions
- consequences of the decisions are not totally predictable

Expert system technology must provide suitable formalisms and mechanisms to handle typical problems of a configuration task. For a long time, the development of configuration expert systems was influenced by the rule-based paradigm. The best-known representative is the system XCON [McDermott 1982]. Although XCON is considered a success and it is one of the most-cited expert systems, the concepts of XCON are not applicable in many domains. The rule-based paradigm has been criticized [Günter et al. 1990; Harmon et al. 1989]. The criticism refers to the following aspects: serious problems with knowledge acquisition, consistency checking and in particular maintenance; missing modularity and adaptability. The necessity to reverse decisions leads to problems; and the integration of user instructions and case based approaches is inadequate.

Some promising suggestions already exist in the field of configuration problems and there are also some expert system applications. Apart from rule-based systems, the following concepts of AI-research have been employed for configuration systems:

- object oriented representation of the configuration objects
- administration of the dependencies with a constraint-system
- top-down-design
- "intelligent" backtracking
- case base configuration

## Discussion

We will discuss same new aspects of knowledge representation in configuration systems:

- description logics and configuration problems
- integration of CAD-systems, database-systems and STEP/Express
- structure-oriented or ressource-oriented representation and problemsolving
- representation of uncertainty

## KONWERK

KONWERK is a tool-box for configuration tasks. The conception and realisation of this modular tool aims to support a developer of expert systems for configuration and design tasks. KONWERK (s. [Günter 1995]) consists of several integrated methods and representations mechanisms which are well suited for configuration and design tasks. Some concepts are based on the system PLAKON [Cunis et al. 1991].

A central requirement for tools like KONWERK is its applicability in different domains. Thus its concepts have to be universal and must not be too special. On the other hand every domain has its special needs. Therefore we propose a solution of modules that can be chosen by the developer in order to solve his problem.

The problem-solving modules are divided in basic modules and extension modules which either enhance the abilities of basic modules or provide new abilities. The basic modules cover the following general tasks in configuration and design systems

- representation of domain objects
- representation and processing of constraints and heuristics
- formulation of configuration tasks
- control of the configuration process

Extension modules enlarge the functionality of KONWERK in several ways. Due to the concept of a tool-box with a variety of modules the user of KONWERK can build his tool in a very flexible way by selecting the necessary modules.

## Literature


[Cunis et al. 1991] R. Cunis, A. Günter und H. Strecker (Hrsg.) *Das PLAKON Buch - Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen,* Springer Verlag, 1991

[Günter 1991] A. Günter. *Flexible Kontrolle in Expertensystemen für Planungs- und Konfigurierungsaufgaben in technischen Domänen,* Dissertation, Universität Hamburg; erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz Band Nr. 3, infix-Verlag (1992), 1991

[Günter 1995] A. Günter (Hrsg.) *Wissensbasiertes Konfigurieren – Ergebnisse aus dem Projekt PROKON,* infix St. Augustin, 1995

[Günter et al. 1990] A. Günter und R. Cunis. *Separating Control from Structural Knowledge in Construction Expert Systems*, in: *3. IEA/AIE,* Charleston, USA, (Seite 601-610), ACM Press, 1990

[Günter et al. 1992] A. Günter und R. Cunis. *Flexible Control in Expert Systems for Construction Tasks*, in: *Journal Applied Intelligence, Kluwer*, Vol. 2, (4), (Seite 369-385), 1992

[Harmon et al. 1989] P. Harmon und D. King. *Expertensysteme in der Praxis*, Oldenbourg, 1989

[McDermott 1982] J. McDermott. *R1: A Rule-Based Configurer of Computer Systems*, in: *Artificial Intelligence*, Vol. 19, (1), (Seite 39 - 88), 1982

# What Is Still To Do In Order To Solve Configuration Problems In Practice?

**Harald Meyer auf'm Hofe**
DFKI GmbH
P/O Box 2080
D-67608 Kaiserslautern

## 1 Introduction

So far several methods have been proposed to represent a configuration problem as well as to solve it. The ability to represent and solve either resource-oriented or structure-oriented configuration problems seem to become a least standard to speak of a configuration system especially in Germany. However, the ability to be beneficially applicable to real world has been proved only by some prototypes. This is a serious situation because in the meanwhile certain commercial tools are available that claim to support knowledge based configuration like the *product configuration within R/3 (SAP)*. Research on configuration systems enters competition with these tools and will be, consequently, forced to prove its relevance for practice. Thus, it is necessary to analyse application scenarios of commercial tools and real world applications of knowledge based configuration systems in order to work out the benefits, research on knowledge representation can promise. Unfortunately, research work is typically still based upon the following preliminaries that do usually not hold in the real world:

1. *Configuration tasks typically concern certain and well-founded knowledge.* Contradicting conventional wisdom all human experts and especially engineers are neither accustomed nor even able to express their results in a well-founded language. There knowledge typically consists of many "rules of thumb" of unknown relevance and exceptions. Even certain knowledge is mostly given by informal representations like e.g. construction plans.
2. *"Declarative" languages reduce the effort of acquiring and maintaining the knowledge base.* This claim has not been proved so far. It depends at least on what is understood by declarative languages. Unfortunately, most human experts will only consider languages as "declarative" in the sense of "relatively easy to understand" that they are accustomed to use.
3. *The task of configuration is, to compute the description of a machine that complies with a given specification.* Unfortunately, most real world configuration processes do not fit into this scheme. Yet, human beings have been the only available *problem solvers* in industrial practice. Thus, in the past there has been no need to avoid conflicting and fuzzy specifications of configuration tasks, because human beings do not work too accurately. As a consequence, configuration can only be done dealing with conflicts and under-specified problems. In many cases, configuration systems can only hope to assist an expert, and there are a lot of questions left open concerning intelligent design or configuration assistant systems.
4. *Configuration denotes a problem class of industrial practice.* In contrast to research, configuration in the real world never ends in itself. It is always only a part of e.g. the bidding process, product development or pricing. Each of these processes has its own needs, that have to be concerned by a system that aims to improve this process. E.g. an engineer will only deploy assisting systems, where he is allowed to rule everything. In contrast, a salesman will generally be overtaxed with such a system. *Prizing* implies i.e. the ability to represent the specification of "robustness" of bids. It has to be ensured that delivering a machine due to a given bid does not lead to a loss for the company.

Research on configuration will be forced to drop these preliminaries. This paper analyses own experiences with a commercial tool and two real world problems. Referring to these experiences improvements of the described approaches are discussed that require and justify further research work. A concluding section points out consequences that seem to follow from the given experiences.

## 2 Experiences

In order to work out some characteristics of real world configuration problems this section will introduce into

- a commercially available configuration tool to point out industrial practice in configuration,

- a DFKI research project on *soft constraints* that led to a commercial scheduling system being available soon, and
- the analysis of a real world configuration problem that demands further research work in order to be solved.

For each example the problem is described followed by a brief introduction into the approach of solving the problem. This is not intended to be a technical paper. Thus, the approaches are sketched superficially. Finally, potentials for further research are mentioned from the author's perspective.

## 2.1 Product configuration within R/3

This section points out aims and abilities of the *product configuration within the R/3-system* [SAP AG, 1996]. The system is roughly spoken a programming environment specialized on relatively simple configuration tasks. The product data base of a company comprising all objects to be bought or sold is possibly very large. To avoid maintenance intensive redundancy in the product data base of R/3, compound products are represented in an intensional manner stating the name of a product class, e.g. car in fig. 1, followed by some attributes representing requirements, like cheap and economical. For certain tasks e.g. commission this representation has to be replaced by a *reduced bill of materials (BOM)* holding all the stuff that the compound product is made of. This problem seeming to be a typical configuration problem exhibits certain characteristics that are not commonly assumed:

- Requirements are always specified as attributes of a certain product class. This property eases representation compared to the more general problem of structure-oriented configuration, where the relation between configuration goals and object classes is hardly a one-to-one relation.
- For efficiency reasons design decisions shall not be retracted. Thus, a classical search problem is turned into a problem of programming.
- A typical requirement on designing systems for use in industrial practice is compliance with traditional processes. In this case the configuration *programs* remove components from a maximal bill of materials if they are not needed to satisfy the given requirements. Such maximal BOMs are generally used to represent compositional relations. Referring to more complex products they can comprise a hundred and more materials. Consequently, experts are needed to maintain them.

Thus, the system is designed especially to handle large numbers of commissions effectively, each given by a product class with some attribute values specifying additional requirements. However, it is in the responsibility of the merchant to distinguish consistent requirements from unsatisfiable ones. Such order forms are supposed to be collected over the day and then transferred to a server, where reduced BOMs are computed by the configuration system and concluding processes have to be initiated e.g. in the forwarding department or the stock-keeping.

### The approach

Basically, the configuration procedure within R/3 is the traditional procedure as sketched in fig. 1. The knowledge base mainly consists of preconditions for the selection of materials in the maximal BOM and of selection rules. The product configuration expert ensures by introducing such rules, that only the actually needed materials become parts of the reduced BOM to the current product requirements. If a compound product is selected by such rules, then it will be configured by the system, as well. This mechanism implements configuration involving more than one level in the object hierarchy. In the example of fig. 1 rule 1 selects the engine 1.6i because the actually configured car is required to be cheap and to have a 75 PS powered engine. Rule 3 demonstrates a "special feature" of the system. In fact, the rules are allowed to cause arbitrary actions ranging from assigning a certain cardinality, as in the example, to the execution of arbitrary programs in the R/3 system.

The "knowledge base" of the configuration system is embedded in R/3's data base scheme. This special property, necessary for implementation reasons, structures the rule base in a manner being rather intuitive for domain experts. Thus, the domain expert is able to find out conveniently the conditions that have to hold true to select a certain item of the BOM.

Lately, the configuration within R/3 has been extended in order to comprise all elements of structure-oriented configuration. Elements of R/3's class system can be constructed to specify certain classes of materials. Such material classes then may appear in BOMs to express that this component of the compound product has to be an instance of the material class. Another extension concerns the introduction of *constraints* to improve the specification of integrity and compatibility conditions[1]. Their construction in the dynamic knowledge base is triggered in a way very similar to the *conceptual constraints* in PLAKON [Cunis *et al.*, 1991].

The problem of conflicting requirements has lead to an additional "interactive" configuration mode. Stock data can be used to test the availability of the current configuration. The system tries to display conflicting selections to the user. However, this information reflects at best the dependencies within the knowledge base that have been discovered

[1] However, R/3's constraints ignore all results of constraint processing research completely.

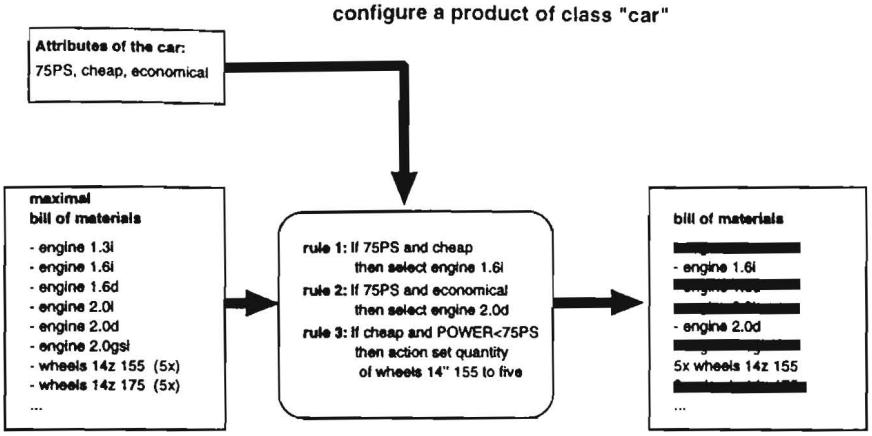


Figure 1: Configuration in the R/3–system: Removing components.

through the latest inference steps. In non-trivial cases this information is only a small hint. It is, then, in the responsibility of the user to retract the selection being responsible for the conflict.

### Research areas

Nevertheless, configuration within R/3 has many problems that can basically be described as follows: Most of the work requiring some intelligence is left over to the programmer or the user. As mentioned above this is especially true for interactive configuration. Conflict management is certainly an actual research area. This is the topic of section 4.

But even the stand alone mode exhibits some problems mainly concerning the task of constructing and maintaining the knowledge base. The example in fig. 1 illustrates this. The given attribute values cause the selection of *two* engines while a car typically has only one. This error is caused by a lack of representing invariant properties of the knowledge base like "a car has only one engine". The example seems to be too simple to justify research work, but one has to remind that there might be a quite large palette of distinguished engines available for the car. In this case it is very difficult to ensure the selection of only one engine because one has to consider the tails of a large number of selection rules distributed all over the objects of the knowledge base (the maximal BOM).

Alternatively, the intended invariant property of the knowledge base could be represented by a material class engine appearing in the maximal BOM. In a rather similar way engines could be represented as a configurable material engine. Unfortunately, the experts having to construct a knowledge base are probably rather domain experts than configuration experts. Thus, they are not familiar with maintenance problems of configuration systems and will, as implementing people usually do, consider any problem being specifyable in their programming language as solvable. Consequently, they need hints to achieve robust representations like automatic discovering of product classes etc.

However, such a representation can cause new problems in R/3. BOM items being material classes are only reduced to an instance, if enough attribute values of the class are known to determine an instance of the class. Otherwise, the class name either appears in the reduced BOM as well, or the user is asked for some technical details he possibly does not know to distinguish available alternatives. The result of the first effect are very unsatisfying configurations stating for instance only, that a car comprises a material of class engine and four equal instances of class wheel without further details. The latter case is a typical problem of interactive configuration.

This example addresses problems that has been widely neglected by research. Knowledge bases must have a structure that makes maintenance easier. Such problem structures often depend on the maintenance task to be done actually. Therefore, it is often necessary to provide more than one view on the knowledge, e.g. several levels of abstraction, and conventional terminologies. It is certainly a potential of knowledge based systems to provide techniques of integration, transformation, and translation of several languages enabling the management of several views on the same thing. Avoidance of expensive employee training justifies the effort to be spend on the development of such systems.

For example in R/3 selection rules are appropriate to assess some adequacy aspects of the knowledge base, while the class system has to be used in order to state some invariant properties of the knowledge base. Translation from the first into the latter view is assumed to be rather valuable improving the construction of the knowledge base. Finally, a transformation according to the implemented inferences can solve the above mentioned performance problems. However, these approaches have not been applied beneficially, yet. This will remain true unless the development process of knowledge bases is better understood.

Additionally, the necessity of integrating AI in-

ferences effectively with database management is proved by the application scenario of configuration within R/3. Certain properties of the products in the products database such as availability and especially the price have to be taken into account when doing configuration. In fact optimization of the results is one of the most important potentials of methods coming from AI research. However, the example is also a problem, where inferences are necessarily limited to very effective methods.

## 3 ConPlan: Dealing with large search spaces

Configuration is often told to be a search problem. Whenever this holds not true in any particular case, many configuration problems are too complex to hope for a configuration procedure avoiding any retraction of design decisions. Thus, techniques to solve similar search problems from *scheduling* or more exactly *time tabling* are of interest here. The ConPlan project has been about such a problem: nurse scheduling. Generally, work on non-trivial *real world problems* is not very praiseworthy, because initial requirements are usually hardly satisfied in complete and the initial time schedule is never kept[2]. The final result of such work is an increase in the number of open questions. Nevertheless, exactly these questions — concerning the representation of optional requirements and default assumptions as well as techniques of local and heuristic search — may be of interest in this context.

The problem of nurse scheduling is illustrated by fig. 2. At the moment, most hospitals still use a three-shift model with only one early-morning-shift F1, one day-turn S1, and one night-shift N1. Personnel scheduling is typically done by hand. Due to cost pressure and the deficiency of qualified and experienced personnel it has been recently recognized that working times must be much more flexible and efficient. A reasonable and promising solution seems to be the introduction of additional overlapping shifts (e.g. six- or nine-shift model) with less working hours. The new shifts can be scheduled in a way, that the overlapping hours are during very work-intensive periods. Part-time employees can be flexibly scheduled. Overtime work can be avoided more easily. There is more room for employees' requests. Of course, the problem of personnel scheduling becomes much harder and the simultaneous satisfaction of all constraints can hardly be managed by hand.

The task of nurse scheduling is to assign a *shift* to each nurse on a day in a certain period of time, typically four or five weeks. A variety of requirements with differing importance must be considered. These requirements comprise

- compliance with legal regulations,
- minimizing personnel costs,
- optimizing personnel assignment with respect to expenditure of work,
- consideration of special qualities of employees,
- management of vacation and absence,
- management of working time and shifts,
- consideration of employees' requests,
- established working time models (sequences of shifts, that comply with legal regulations) should be followed.

Obviously, compliance with legal regulations is required, whereas the consideration of personnel qualities is recommended. Low personnel costs are more important than the consideration of employees' requests.

In the representation of nurse scheduling as a problem with soft constraints, there is a constraint variable for each nurse on each day — about 900 to 1000 variables. The domains of the variables consist of possible shifts (also comprising holidays and idle shifts). Constraints between the variables shall ensure compliance with the requirements of the domain. Weight and priority parameters of the constraints are given to enable the system to resolve conflicts. The task is to find a labeling of each variable, that satisfies all hard constraints and as important soft constraints as possible.

### The approach

Optional requirements are explicitly specified by *soft constraints* in order to achieve a flexible and compact problem representation. State of the art in combinatorial search is still encoding such information about preferred solutions into the problem solving procedure[3]. The same kind of soft constraints has been beneficially applied to represent properties of presumably good solutions of the problem. Prospective processing of such constraints has been exploited to inform *branch&bound* search about the most promising assignments being probably part of a sufficient solution [Meyer auf'm Hofe and Tschaitschian, 1995]. But even this improved tree-search algorithm failed to compute acceptable solutions for

[2]This statement considers only work on real applications. Contrary, the term "real world problem" is often used to denote problems of realistic size and well known specification that are usually used to prove the relevance of a given solution approach. Both notions differ in an important point. When work on a real world problem in the sense of this paper starts, there is hardly known anything about it in detail. The decision to apply certain tools and techniques is triggered by the increasing knowledge on the problem. In contrast, real world problems in the academic sense are inversely chosen according to a solution approach whose benefits have to be proved. The latter task is much easier.

[3]For example the ILOG-Solver as the most common constraint solver offers the opportunity to set *choice points* explicitly in tree search algorithms like the *branch&bound* [ILOG, 1996].

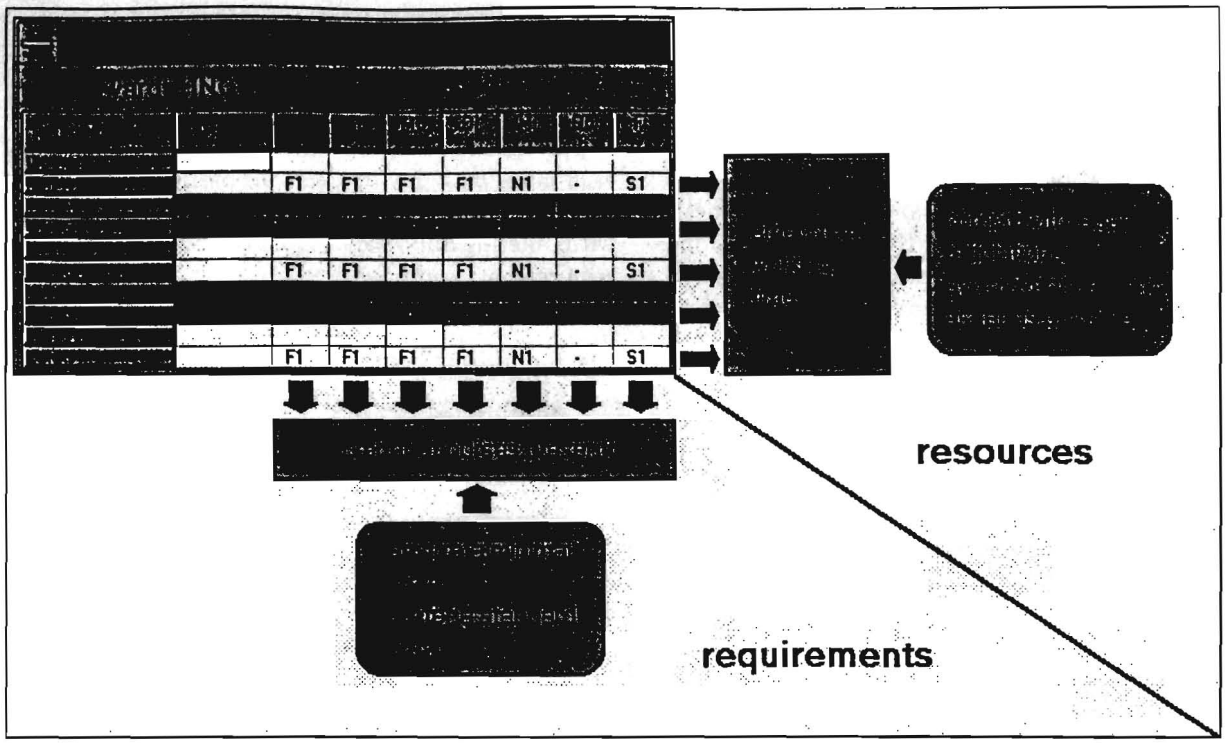


Figure 2: The nurse scheduling problem of ConPlan.

many relevant problems within a reasonable amount of time. Consequently, repair-based mechanisms had to be developed to improve tree-search results. The algorithm searchs for a Pareto optimum that is good enough to be accepted as an answer of the system[4] [Minton *et al.*, 1992]. Pareto optima of the nurse scheduling problem are of rather different absolute quality. First approaches of generic repair heuristics have not been able to do their job sufficiently. Therefore, the current version of the system incorporates problem specific repair strategies. This version will be commercially available soon. However, future research is necessary to discover repair strategies according to the given problem specification automatically.

**Research areas**

This example raises three questions to be answered by research.

The first one is about the semantics of large combinatorial optimization problems. Large search problems like the nurse scheduling example can be treated in one of the three following ways:

1. The problem representation is modified in a way that enables the application of exhaustive search. This may be done adding new constraints to prune the searching space or avoiding certain unkind problem specifications. The applicability of exhaustive searching algorithms to such problems ensures the optimality of the solution referring to the problem specification, but this specification is not guaranteed to have anything in common with the original problem.
2. One tries to inform search as good as possible about consequences of searching steps. This procedure requires to put large effort on acquiring such control knowledge without guarantee, that it will work (as the ConPlan example shows).
3. One applies local or heuristic search procedures [Minton *et al.*, 1992, Wallace and Freuder, 1995]. Applied to optimization problems such procedures do *not* guarantee optimality according to the problem specification. One even does not know how good the returned solution is compared to the optimal one.

Often approach 1 is told to be the right way because it's semantic is assumed to be clearer. However, this argument is only relevant to applications that allow to estimate the effect of problem simplification. On the other hand approach 2 and 3 have some advantages from a technological perspective offering the same opportunities to retrieve good solutions. They can be combined. For example the ConPlan project at first aimed at following approach 2 that guarantees clear semantics and does not force to simplify the problem. As exhaustive searching turned out to be unable to cope with the problem, the acquired control knowledge has been used to inform local search without further changes.

Secondly, experiences with ConPlan suggest to develop more elaborate methods to exploit certain

[4]The applied algorithms have not yet been published. Nevertheless, the overall procedure has been described within another context [Meyer auf'm Hofe, 1996a].

properties of the actual problem. Adopted repair strategies are needed to compute a solution of reasonable quality. As another example consider structure-oriented configuration enriched by global constraints being well known from the scheduling context. All resource-oriented configuration problems [Heinrich, 1993] can be specified in such a language. However, for the latter problems an effective problem solving procedure is known. Thus, transformation into resource-oriented configuration is recommended whenever possible[5].

The third question concerns the handling of preferences and believe. Certain inferences have been proposed to deal with fuzzy sets, fuzzy relations, probability, confidence and so on. Amazingly, management of more or less preferred requirements has hardly been explored in detail, yet. Most of the approaches mentioned above assume a *"commensurability between preference levels pertaining to different constraints, i.e. the user who specifies the constraints must describe them by means of a unique preference scale"* [Dubois *et al.*, 1993]. The importance of an optional requirement is in contrast obviously determined with reference to the other requirements because importance is intended to control conflict resolvation. Unless cash prices there is no global preference scale to distinguish the merits of requirements. Nevertheless, *prioritized* respectively *fuzzy constraints* [Dubois and Prade, 1992] have been praised as declarative representations of optional requirement specifications in configuration systems. This claim contradicts even another intuitive demand. One generally expects that a compliance with more requirements is preferred. In a system of *prioritized constraints* the priority of the most important violated constraint determines the merit of a labeling of the variables. Representing each requirement by a constraint the labelings being consistent with more constraints should be preferred. Using *prioritized constraints* (or *fuzzy constraint*) this is not necessarily the case. All solutions consistent with the most important constraints are considered to be of equal merit.

More promising seems to be an approach based upon a partial ordering $\succ$ of all sets of requirements [Meyer auf'm Hofe, 1996b]. The searching algorithm tries to find a solution that is consistent with one of the $\succ$-largest satisfiable sets of constraints. However, specifying such preferences by pairs of sets of requirements to represent any intended strategy of conflict resolvation explicitly is too complicated. Thus, certain methods have to be estimated to derive $\succ$ from e.g. a partial ordering of single requirements in a way that leads to the intended behaviour of the system.

[5]This has been the reason for Holger Wache to integrate the balance operation of resource-oriented configuration into the TooCon system [Wache and Abecker, 1996].

Consequently, this problem still requires a lot of empirical research. Unfortunately, optional requirements often appear corresponding with believe for example in bidding and pricing processes. A bid has to be computed according to partially optional requirements, but prices and availability of products has to be estimated. Such correspondence has not been explored, yet.

## 4 KoALa: Interaction and conflicting decisions

In contrast to the previous examples this section is about the results of a problem analysis. Further work has been canceled because of financial reasons. Consequently, the section on solution approaches is left over. Nevertheless, nearly all aspects of interactive configuration can be illustrated by this problem.

Client has been a company developing systems for optical control of manifactoring processes. As illustrated in fig. 3 a picture is grabbed from a video camera and than analysed by a computer. Therefore, the picture is subdivided in certain regions of interest. Each of these regions serves as input to some fast filter programs that are able to extract some interesting attributes from pictures. These attributes are then used as input of the classifier who determines manifactoring failures. The classifier is built of software modules that have to be configured. As a result of the project the modules should as well comprise some machine learning algorithms to enhance performance of the system in several situations. The problem has two important characteristics:

- The system is to be used in various different situations. Thus, learning scenarios are varying.
- The developers of the system do hardly know anything about machine learning.

The configuration system is required to improve the use of the machine learning tools and, additionally, to provide all the necessary knowledge on machine learning.

**Research areas**

Such complex construction processes induce rather complex problem specifications. Consequently, the user is *always* superior to the system in at least one aspect: He or she know more details about the current problem. It does not matter whether the system is better informed in certain relevant areas. It will always depend on the problem specification given by the expert that is *a priori* neither complete nor satisfiable. Thus, a close dialog of expert and system is recommended.

Additionally, the probability of conflicting problem specifications is likely to grow with the complexity of the specification language. It is certainly a task of the configuration system to avoid conflicts, because retracting design decisions is very difficult and time consuming. Time and patience of the expert can be expected to be rather rare and expensive.

3. Section 3 discovered the representation of optional requirements as open and urgent questions. With reference to weakly defined or incomplete problem specifications the results of a configuration systems are often much more required to be *good* than correct. In these cases e.g. local or heuristic search is appropriate to compute optimized solutions.
4. Section 2.1 describes a system being specialized on efficient processing of commissions but unsuitable for another application scenario — as an intelligent assistant. To prove its relevance research has to put much more emphasize on the scope of proposed inference techniques and system designs referring to typical application scenarios. Configuration problems are too manifold to be solved by a unique approach.

Apparently, it is hardly justified to distinguish between the knowledge itself and available techniques of knowledge processing. The two (potential) technological advantages of knowledge representation are

- integration of many knowledge resources and
- provision of task specific views on this knowledge.

However, the availability of complex, generic, and efficient inference techniques determines whether these claims are hypothetical or not.

## References


[Abecker *et al.*, 1996] A. Abecker, H. Meyer auf'm Hofe, J. P. Müller, and J. Würtz, editors. *Notes on the DFKI-Workshop: Constraint-Based Problem Solving*, Document D-96-02. Deutsches Forschungszentrum für Künstliche Intelligenz, 1996.
`http://www.dfki.uni-kl.de/~aabecker/WS-CO.html`.

[Bergmann *et al.*, 1994] R. Bergmann, J. Paulokat, A.-M. Schoeller, and H. Wache, editors. *PUK-94: 8. Workshop "Planen und Konfigurieren"*, number SWP-94-01 in SEKI Working Paper, 1994.

[Cunis *et al.*, 1991] R. Cunis, A. Günter, and H. Strecker, editors. *Das PLAKON-Buch*, volume 266 of *Informatik-Fachberichte*. Springer-Verlag, Berlin–Heidelberg, 1991.

[Dubois and Prade, 1992] D. Dubois and H. Prade. Possibility theory as a basis for preference propagation in automated reasoning. In *Proc. of the 1st IEEE International Conference on Fuzzy Systems*, pages 821–832, San Diego, CA, 1992.

[Dubois *et al.*, 1993] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. of the 2nd IEEE International Conference on Fuzzy Systems*, pages 1131–1136, San Francisco, CA, 1993.

[Günter, 1993] A. Günter. Verfahren zur Auflösung von Konfigurationskonflikten in Expertensystemen. *KI – Künstliche Intelligenz*, (1):16–23, March 1993.

[Heinrich, 1993] M. Heinrich. Ressourcenorientiertes konfigurieren. *KI – Künstliche Intelligenz*, (1):11–15, March 1993.

[ILOG, 1996] ILOG. ILOG SOLVER: White Paper, 1996.
`http://www.ilog.com/products/solver/papers/WHITEPAP.ps`.

[Jampel *et al.*, 1995] M. Jampel, E. C. Freuder, and M. Maher, editors. *Workshop Notes* CP95 Workshop on Over-Constrained Systems, Cassis, France, 1995.

[Meyer auf'm Hofe and Tschaitschian, 1995] H. Meyer auf'm Hofe and B. Tschaitschian. PCSPs with hierarchical constraint orderings in real world scheduling applications. In *[Jampel* et al., *1995]*, pages 69–76, 1995.

[Meyer auf'm Hofe, 1994] H. Meyer auf'm Hofe. Verarbeitung von Constraints in der Expertensystementwicklungsumgebung IDAX. In *[Bergmann* et al., *1994]*, 1994.

[Meyer auf'm Hofe, 1996a] H. Meyer auf'm Hofe. Partial satisfaction of constraint hierarchies in reactive and interactive configuration. In *[Ruttkay and Hower, 1996]*, 1996.

[Meyer auf'm Hofe, 1996b] H. Meyer auf'm Hofe. Representation of requirements through preference orderings of soft constraints. In *[Abecker* et al., *1996]*, January 1996.

[Minton *et al.*, 1992] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction problem and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Petrie *et al.*, 1996] Ch. Petrie, H. Jeon, and M. R. Cutkosky. Combining constraint propagation and backtracking for distributed engineering. In *[Ruttkay and Hower, 1996]*, 1996.

[Ruttkay and Hower, 1996] Z. Ruttkay and W. Hower, editors. *Notes on the ECAI'96 Workshop on Non-Standard Constraint Processing*, 1996.
`http://yeats.ucc.ie/~walter/ECAI96w/announce.html`.

[SAP AG, 1996] SAP AG. Product configuration within R/3's system enterprise resource planning, July 1996.
`http://www.sap-ag.de/kiosk/literature/pdf/procon03.pdf`.

[Wache and Abecker, 1996] H. Wache and A. Abecker. Constraint-Processing in der Konfiguration. In *[Abecker* et al., *1996]*, January 1996.

[Wallace and Freuder, 1995] R. Wallace and E. C. Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In *[Jampel* et al., *1995]*, pages 97–101, 1995.

# Hybrid Knowledge Organization within an Object Framework *

**Wolfgang Oertel and Uwe Petersohn**
Technical University of Dresden
Department of Artificial Intelligence
D-01062 Dresden
Fax: 49-351-463-8335
Tel: 49-351-463-8430/8431
Email: oertel,peterson@freia.inf.tu-dresden.de

## Abstract

The presented paper summarizes the experiences of the own application-oriented work concerning the representation and use of knowledge in technical configuration and design domains. The main focus is put not on theoretical studies of certain models but on their practicality for solving real world problems. As a result, we propose to use objects representing real world phenomena or conceptual terms of a domain, organize these objects in a network built by some semantically predefined relations, and define behaviours to guide the problem solving process. Besides this global structure of the knowledge, the fine structure is given by the inside of the objects consisting of sets of knowledge elements like production rules, logic clauses, or constraints. Finally, the interface of the knowledge model to available external design models essentially decides about the acceptance of the knowledge model in an application domain.

## 1 Introduction

Configuration means the composition of given elements described by parameters or structures to more and more complex aggregates according to a set of generic rules to satisfy certain restrictions and to reach a certain functionality.

The solving of configuration tasks can be classified primarily as a synthesis process. However, such a synthesis can not be done correctly without performing analysis processes, as well. The main problem of modeling real configuration domains is their complexity. So, a model must be found, that divides a whole domain into a set of smaller parts that are easier to handle both by the system and by the user. The model must allow an adequate representation but also an efficient implementation.

*This research was supported by the Federal Ministry of Education, Science, Research and Technology (BMBF) within the joint project FABEL under contract no. 01IW104. Project partners in FABEL are GMD – German National Research Center for Information Technology, Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

The paper describes a rather generic knowledge organization model that satisfies these demands to some extent. Section 2 examines the application and system background of the model. The model itself is presented in section 3. Section 4 shows how the knowledge is used in the problem solving process. In section 5, it is explained how the model works in connection with external design platforms. Section 6 describes an example system and its knowledge base. And finally, section 7 imparts some experiences concerning the practical work with the knowledge model and the developed systems based on it.

## 2 Application Domains and Systems

We have made the main experiences of performing configurations during the development of the application system DOM [1] [8]. The special architectural domain of this system handles the technical installation of buildings. Elementary duct pieces (for water, sewage, air conditioning, electric power, gas, heating etc.) are put together to complex duct systems satisfying certain constraints and functionalities. The system supports the analysis as well as the synthesis of respective building layouts.

A second developed application system is GEAR, a simple automatic construction system for multistage gear systems in the field of mechanical engineering. The knowledge and its representation in this system are described in [7].

Also, some other – on the first view not configurational – applications seemed to have similar demands, problems, and solutions. For instance, the composition of several single connections or diversions to a complex connection or diversion in a traffic information and dispatching system can also be regarded as a very simple form of configuration.

Based on the experiences made in these domains, it has been tried to generalize the necessary structures and operations and put them in the form of generic tools into a knowledge-based development system FAENSY [9] [10]. This system provides the main structures, operations, and interpreters for the development of concrete knowledge-based application systems, also in configuration domains.

All the mentioned systems are implemented in Allegro Common Lisp and run on Unix workstations with user interfaces made in Tcl/Tk. The systems DOM and FAENSY and the knowledge models they are based on have been developed within the context

of the FABEL project [11], [3].

## 3 Knowledge Model

The developed knowledge organization model described in [10] can be regarded as a multi-layered hybrid approach that is based on a consequent object orientation. The model represents real world phenomena of the universe of discourse or conceptual terms of the respective technical language of the domain as complex knowledge objects. These objects are embedded in a network made by a set of predefined relations. Behaviour structures, finally, specify problem solving approaches on the object and relation sets. They also define the semantics of the relations. The global organization of the knowledge is shown in figure 1.

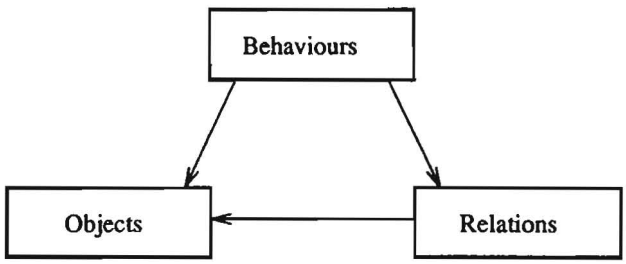


Figure 1: Knowledge Organization

**Relations:** The decisive semantically predefined relations of the model are specialization, partialization, realization, and association as well as their inverse relations. They determine the global structure of the knowledge fundamentally. Some of the relations are useful not only for getting an adequate representation of the configuration problem, but also for technical purposes like storing, changing, reasoning, efficiency, and management.

For configuration tasks, the relations specialization and partialization are of special importance. The specialization distinguishes between specific objects affected by the configuration process and generic objects describing the configuration process. The generic objects contain analysis rules and synthesis rules. The partialization divides complex specific and generic objects into smaller parts that can be handled easier by the system. So, there are, for instance, whole complex layouts on the one hand and single unstructured design elements on the other hand.

With the help of the realization relation, semantic descriptions of a layout can be connected to respective pure spatial geometric or graphic representations. Finally, the association relation allows to connect objects of the same specialization, partialization, and realization level, for instance, to express similarities between objects that can not be specified more precisely.

**Objects:** The internal structure of the generic objects is described by a set of knowledge elements belonging to several pure knowledge models. The number of usable knowledge models depends on the set of available interpreters. These knowledge elements determine the fine structure of the knowledge. For analytic tasks, the logic clause model yields acceptable results. For synthetic tasks constraints or production rules seem to be the favorite models. Though, in principle, these knowledge elements are defined locally within the objects, they have to refer via interfaces to other objects, too. Besides generic knowledge objects, specific objects contain knowledge elements in the form of relations, hierarchies, or networks.

At many points of the knowledge elements, the expressiveness and the efficiency of the used knowledge models are not enough. So, it is necessary there to go back to the implementation language of the system and to include expressions of it on the knowledge level.

**Behaviours:** At first, there are necessary basic behaviours that define the semantics of the introduced relations. Especially, such properties of the relations as transitivity, symmetry, and reflexivity must be guaranteed. But also, interdependencies between the relations may occur.

For configuration domains, the most important behaviour structures are analysis and synthesis. Analysis means that a conceptual / semantic representation of a layout is computed from a given spatial or structural layout. With the help of this representation, it can be decided if there are inconsistencies within the layout or not and what are the functionalities of the layout. Synthesis behaviours describe the process in the opposite direction. They take this or another semantic representation to generate or change a spatial layout satisfying the given restrictions and functionalities.

During a configuration process, a permanent switching between analysis and synthesis is necessary. The global behaviours should be defined in a straightforward way to restrict undoings and revisions to local levels.

Analysis and synthesis can not only be done in a deductive manner using generic rules. Also, the stored specific case knowledge in combination with generic similarity and adaptation knowledge supports the problem solving processes. This is done by analogical reasoning.

Finally, behaviours for learning and reorganization are possible.

## 4 Problem Solving

The configuration process is understood as a problem solving process. It starts with a semantic specification and a given initial layout. Within the process, all kinds of stored knowledge can be used to transform the actual layout. The process terminates when the semantic specifications are satisfied and a result layout is reached.

The problem solving process is performed locally by the internal knowledge of the objects in combination with respective knowledge-based interpreters. There are interpreters for production rules, logic clauses, constraints, similarity functions, and adaptation rules.

The process is guided globally by behaviours defined upon the semantic network linking the objects. To activate such behaviours, global interpreters are necessary. For this purpose, the Lisp interpreter as well as the production rule interpreter is used.

By defining and activating behaviour structures, the whole configuration process can be organized in different ways. So, it is possible to define a lot of elementary behaviours. The user can activate them in a sequence and interact, if necessary, after each single step. On the other hand, elementary behaviours can be combined to complex behaviour structures carrying out the whole process automatically and deciding on their own what to do in which situation.

## 5 Design Model Interface

For the implementation of practically usable knowledge models for configuration domains, interfaces to other models are essential. Real composite objects are not developed within a knowledge-based system today. In most cases, external graphical or geometrical design platforms like AutoCAD or DANCER [5] are used. If a knowledge-based configuration system shall be successful, it must take into account the models offered by these design-based systems.

The first advantage of such an interface is the cooperative or alternative use of problem solving facilities of the different systems. The second advantage is that there is almost always a large set of layouts available in design systems that can be used as specific case knowledge in the knowledge-based system.

How is the configuration knowledge usually organized in a design system? It must be distinguished between specific and generic knowledge.

The specific knowledge is represented in the form of three-dimensional geometrical layouts consisting of sets of design elements. In order to reduce the complexity, subsets of design elements can be clustered to complex design elements like blocks, layers, or groups. Additionally, non-geometrical, mostly textual, data can be glued to the single design elements to refer to certain semantics.

The generic knowledge on the other hand is made by pieces of program code or textual data. The program code can be interpreted and evaluated by the system. It does for instance complex calculations or checks of the consistency.

Because of the differences, it is often not possible to transform between the whole design model and the whole knowledge model. A possible way is, however, to take the layouts for the purpose of an interface and define transformations for the specific knowledge. So, we get the paradigm of a common design board where layouts can be loaded and transformed in different representations. Different methods - design-based methods and knowledge-based methods - operate on them. The approach of the interface is shown in figure 2.

In [6] a transformation between a design-based and a knowledge-based representation of layouts was implemented. It uses the DXF language as the central interface language. Doing this transformation, it must be observed that there are different semantics of the elements in the single models. So, as for

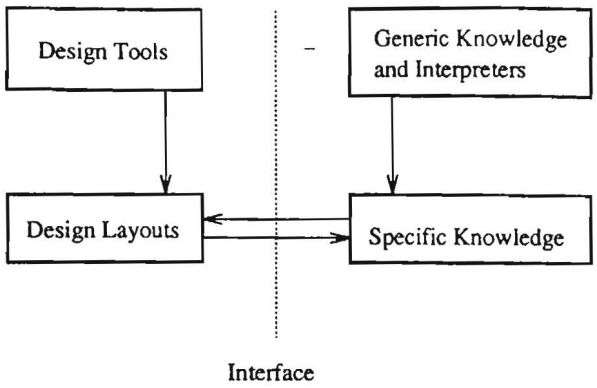


Figure 2: Design Model Interface

the semantics, there are three kinds of elements in each representation:

- elements with semantics only in the knowledge-based system,
- elements with common semantics, and
- elements with semantics only in the design-based system.

When a transformation is done, the elements of the one model that have no semantics in the other model, must be saved and recreated during a later transformation back. Because, in general, the transformation is not a unique mapping, it should be controlled by knowledge, too. So, it is possible to get different transformations on different levels of abstraction. A typical example for this is the use of bounding boxes in the knowledge-based system of design elements described in a detailed geometrical manner in the design-based system.

Finally, there is a list of problems to be aware of when working knowledge-based in a real design environment. The design level is characterized by:

- detailed geometric data,
- several existing geometry models (2D, 3D),
- several co-ordinate systems,
- additional graphic data,
- additional structuring concepts like blocks, layers, and groups,
- additional product and administrative data,
- programs operating on the data,
- a lot of domain-specific implicit semantics (conventions), and
- restrictive possibilities to specify semantics of data explicitly.

## 6 Example System DOM

The components of the introduced knowledge model shall now be illustrated at some examples extracted from the knowledge base of the building application system DOM. Most of the definitions and example structures are taken from [2].

The system DOM has been developed for supporting architects in the process of designing highly installed buildings. Especially, the technical installation of the buildings is modeled in the knowledge base of the system. The system is able to handle concrete layouts of buildings, analyse them, or synthesize them automatically or guided by the user.

The analysis and synthesis of concrete layouts can of course be done using generic domain knowledge. But, not all problems of the domain can be solved in this way. Additionally, some kind of case knowledge is necessary to use former handled layouts for solving the actual problem. If there are gaps of knowledge in the model still again, simple learning methods try to fill them.

The generic knowledge used to build up a domain ontology is the building installation model ARMILLA [4]. The representation for the specific knowledge is based on the A4 model used in the system DANCER [5].

**System Structure:** One major characteristic of the architectural design is that the requisite knowledge is accumulated experimentally. The important implication is that we have to deal with incomplete knowledge and to take precautions for a stepwise extension as well as for a goal-oriented modification of the knowledge without incurring the full cost of re-representation and re-organization of the whole system.

Therefore, the whole system is organized using two technologies: the object-oriented technology and the knowledge-based technology. The main system components are defined as classes. There are the classes behaviour, knowledge, data, and transfer. Each class includes static components (slots) and dynamic components (methods). A class specification together with its set of instances build a so-called base. So we get a transfer base, a data base, a knowledge base, and a behaviour base. The slots are containers for sets, for instance sets of files, design elements, connections, concepts, or assistance functions. The methods are defined to operate on whole instances of a base, on slots of an instance, or on single elements of the sets stored in the slots of the instances. Examples for these methods are create, delete, get, eval, draw, load, save, open, close. The system structure is shown in figure 3.

Because the main parts of the system are realized as object classes, an arbitrary number of instances of these classes can be created during one session with the system.

The knowledge base (AKB) is the heart of the system. It incorporates concepts, schemes, and cases. Under the term concept, a variety of generic domain knowledge chunks are subsumed, such as the definitions of permissible design entities, aggregate objects, topological relations. These concepts can be used as rules or maxims for analysis and synthesis tasks.

On the lower abstraction levels, we have specific cases representing former design states. That means syntactic layouts possibly with glued semantics. The schemes can be regarded as intermediate objects between generic concepts and specific cases. They abstract from detailed case data but are not yet generic enough to be concepts. Here, we find prototypical layouts, layout patterns that can be instantiated, or heuristics.

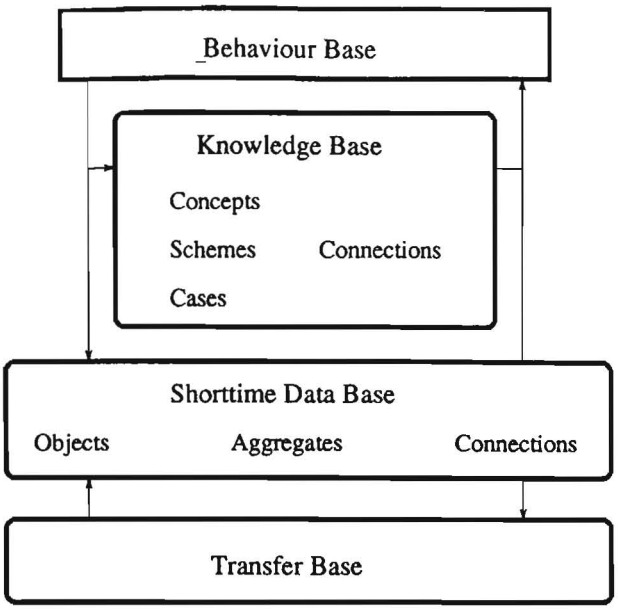


Figure 3: System Structure

Additionally, there exist connections between the single knowledge objects. There are partialization, specialization, realization, and association relations. They are important for getting the whole knowledge base structured. Especially, they represent the partonomy and taxonomy of the ontology.

So, the slots of the knowledge base have the following meaning:

- Concepts: set of concepts which describe certain design terms with analysis and synthesis parts,
- Schemes: set of schemes which describe certain design patterns with analysis and synthesis parts and a set of design elements (objects),
- Cases: set of cases with sets of design elements (objects) belonging to them,
- Connections: set of partialization, specialization, realization, and association relations between knowledge objects.

Each knowledge object, additionally, is determined by an identifier (that makes a direct access possible) and the creation space and time (as an information about where and when the object was defined).

The second main part of the system is the shorttime data base (ADB). It builds the actual working memory – the internal design board – of the system. External building layouts can be loaded in this base, internal operations can query or update this base, and finally the resulting internal structure can be saved in an external format again. The reasons for using a separate internal building structure are manifold. The most important ones are the independence from the changes of outside data formats and activities of other modules, and the possibility to

maintain additionally and temporarily inconsistent data.

The shorttime data base contains a set of objects representing as design elements the current state of the design layout. Furthermore, these objects can be involved in topological relations, here called connections. Special connections refer from single objects or object clusters to concepts of the knowledge base in order to define their semantics. The object clusters are called aggregates. The slots of the shorttime data base have the following meaning:

- Objects: set of defined elementary design elements belonging to one layout,
- Connections: set of pairs of geometrically contacting, including, overlapping, or otherwise related design objects, as well as relations between objects of the shorttime data base and concepts of the knowledge base,
- Aggregates: set of composite design elements.

Figure 4 shows an example layout handled in the shorttime data base of the DOM system. It contains ceilings, floors, walls, as well as four duct systems for cold water, warm water, sewage, and return air. The duct systems of the different aspects and the whole layout can be regarded as a composite object made from single design objects within a configuration process.

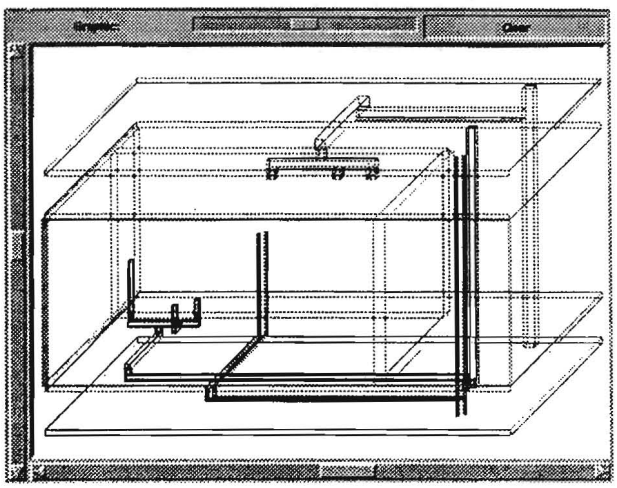

Figure 4: Example Layout

The third main part is the behaviour base (ABB). It contains a set of behaviour structures. Each behaviour structure describes the solution steps for complex system tasks such as collision check or coherence test. These behaviour structures serve as a direct interface to the user of the system. They can be divided into query operations and update operations. A query operation realizes a complex task of getting information about the actual state of the system. An update operation realizes a complex task of changing the internal state of the system. The corresponding slots of the behaviour base are:

- Queries: set of query operations,
- Updates: set of update operations.

Finally, it is important for DOM to be able to interact with other design systems. In order to make the system independent on different and changing external design models, it supports an internal and an external data structure for building layouts. External layout structures are stored in a separate, changeable transfer base (ATB). Because of the differences between external and internal structures, operations are necessary to transform them into each other. Now, if the external structure changes for any reason, only the transformations have to be customized but not the whole data structure functionality in the system.

In the following, the bases of the system DOM are described more precisely in order to get an idea of how the knowledge is organized in the system. We will start with the transfer base and conclude with the behaviour base.

**Transfer Base (ATB):** The transfer base contains a set of external building layout structures. Each of these layout structures is stored in a separate file. It contains a set of design elements. As an example, we show the structure of the elements of the A4 design model. A design element in A4 is called an A4 object. It is a Lisp structure with the following slots:

```
(defstruct (a4object)
  id project x dx y dy z dz time dtime ttag
  dttag aspect morphology precision scale
  user alternative meta composition
  a-v-value ancestor descendant subobjects
  color linewidth graphic-type)
```

An instance of this structure has the following form.

```
(:id 6914 :project murten :x 0.001511 :dx
 1440.002075 :y 960.000061 :dy 2160.000244
 :z 760 :dz 40 :time 0 :dtime 0.01 :ttag
 754935109 :dttag 999999999 :aspect zuluft
 :morphology verbindung :precision huelle
 :scale 6 :user ludger :alternative nil
 :meta case :composition nil :a-v-value ()
 :ancestor 0 :descendant 0 :subobjects
 (3935 2394 1714 1707 1433 3844 3827 3845
  3828 3913 3912 3889 3867) :color
 (:l-red 0 :l-green 0.400006 :l-blue
  0.466674 :l-alpha -1 :fill nofill)
 :linewidth 0 :graphic-type rectangle)
```

As an interface to commercially available design systems (like AutoCAD), the DXF language is used.

**Shorttime Data Base (ADB):** The shorttime data base (ADB) is the actual working memory of the system. Loaded external building structures can be manipulated here. For the representation of layouts, a special internal format is used. The structure of the shorttime data base is given by a class definition.

```
(defclass adb ()
  ((objects) (oconcepts) (aggregates)
   (contacts) (includes) (overlaps)
   (relations)))
```

In the system, an arbitrary number of instances of this class can be created. Each slot of a shorttime data base contains as value a set of further structured elements. The element structure is given by the following slot definitions.

- Objects : set of elementary design objects of a layout; each entry is determined by the identifier (id), the x-value (x), the x-distance (dx), the y-value (y), the y-distance (dy), the z-value (z) the z-distance (dz), the time (time), the time distance (dtime), the aspect (aspect), the morphology (morphology), the precision (precision), the scaling factor (scale), the graphical presentation type (presentation), the presentation color (color), a list of references to other objects (references), and a list of subordinated objects (subobjects)

```
(defstruct (aobject (:type list))
  id x dx y dy z dz time dtime aspect
  morphology precision scale
  presentation color references
  subobjects)
```

- Oconcepts: set of connections between design objects and concepts of the knowledge base represented by an object identifier (id) and a concept name (name)

```
(defstruct (aoconcept (:type list))
  id name)
```

- Aggregates: set of composite objects represented by a design object structure which especially contains a list of included partial design objects (subobjects)

```
(defstruct (aaggregate (:type list)
  (:include aobject)))
```

- Contacts: set of pairs of geometrically contacting design objects represented by two identifiers (id1 and id2)

```
(defstruct (acontact (:type list))
  id1 id2)
```

- Includes: set of pairs of geometrically including design objects represented by two identifiers (id1 and id2)

```
(defstruct (ainclude (:type list))
  id1 id2)
```

- Overlaps: set of pairs of geometrically overlapping design objects represented by two identifiers (id1 and id2)

```
(defstruct (aoverlap (:type list))
  id1 id2)
```

- Relations: set of not predefined relations represented by the relation name (id) and a set of tuples (tuples)

```
(defstruct (arelation (:type list))
  id tuples)
```

The following example shows the contents of a concrete shorttime data base. I contains two design objects, two aggregates, their connection to concepts of the knowledge base, and their topological relations.

```
(OBJECTS ((3843 400.0 40.0 2620.0 180.01
           720 40 0 0.01 zuluft verbindung
           huelle 6 rectangle "#006677" nil
           nil)
          (6914 0.0 1440.0 960.0 2160.0 760
           40 0 0.01 zuluft verbindung
           huelle 6 rectangle "#006677" nil
           (3935 2394))))
(OCONCEPTS ((3868 astleitung)
            (3911 astleitung)
            (3910 astleitung)
            (g254 astansatzleitung)
            (g253 astansatzleitung)
            (g252 astansatzleitung)))
(AGGREGATES ((g267 0.0 1440.0 960.0 2160.0
              760 40 0 0.01 zuluft
              verbindung huelle 6 rectangle
              "#006677" NIL (3868 6914))
             (g253 300.0 140.0 1760.01
              240.0 720 80 0 0.01 zuluft
              verbindung huelle 6 rectangle
              "#006677" nil (3913 3845))))
(CONTACTS ((3913 6914) (3912 6914)
           (3889 6914) (3843 3911)))
(INCLUDES ((6914 3913) (6914 3912)
           (6914 3889) (6914 3868)))
(OVERLAPS nil)
(RELATIONS nil)
```

Some necessary slots are not stored statically, but can be derived dynamically via function calls. They can be regarded as dynamic slot elements:

- aobject-centre, aobject-radius, aobject-direction,
- aocontact, aooverlap, aoinclude,
- aointersection, aounion, aocompound.

**Knowledge Base (AKB):** The knowledge base is intended to collect all domain knowledge. So, we need structures for cases, schemes, concepts, as well as connections between them. The knowledge base has the following internal structure given as class definition.

```
(defclass akb ()
  ((concepts) (schemes) (cases)
   (partials) (reals) (specials)
   (assocs)))
```

In the system, an arbitrary number of instances of this class can be created. Each slot of the knowledge base structure contains as value a set of further structured elements. The element structure is given by the following definitions.

- Concepts: set of concepts which describe a certain design term; each entry is determined by an identifier (id), the creation space (crspace), the creation time (crtime), an analysis part (analyse), and a synthesis part (synthesize)

```
(defstruct (aconcept (:type list))
   id crspace crtime
   analyse synthesize)
```

- Schemes: set of schemes which describe a certain design pattern; each entry is determined by an identifier (id), the creation space (crspace), the creation time (crtime), a set of subordinated objects (objects), an analysis part (analyse), and a synthesis part (synthesize)

```
(defstruct (ascheme (:type list))
   id crspace crtime
   objects
   analyse synthesize)
```

- Cases: set of cases which describe a certain state of a design layout; each entry is determined by the case identifier (id), the creation space (crspace), the creation time (crtime), and a set of objects (objects)

```
(defstruct (acase (:type list))
   id crspace crtime
   objects)
```

- Partials: set of partialization relations between concepts, schemes, and cases represented by an identifier (id) and a list of identifiers of partial objects (ids)

```
(defstruct (apartial (:type list))
  id ids)
```

- Reals: set of realization relations between concepts, schemes, and cases represented by an identifier (id) and a list of identifiers of real objects (ids)

```
(defstruct (areal (:type list))
  id ids)
```

- Specials: set of specialization relations between concepts, schemes, and cases represented by an identifier (id) and a list of identifiers of special objects (ids)

```
(defstruct (aspecial (:type list))
  id ids)
```

- Assocs: set of association relations between concepts, schemes, and cases represented by an identifier (id) and a list of identifiers of associated objects (ids)

```
(defstruct (aassoc (:type list))
  id ids)
```

The interpreter that works on analysis parts is a Prolog-like deduction clause interpreter that tries to prove goals with the help of horn clauses by using the approaches resolution and depth-first-search. Each clause consists of a head (the first list element) and a tail (the rest of the list elements). The character + stands for a disjunction, - for a negation, $ for a function call, and % for a function predicate call. Constants begin with the character '.

The synthesis parts contain sets of production rules that are evaluated by a respective production rule interpreter. A single production rule consists of the number, the condition, the arrow --->, and the action. The elements of the condition and the action can be positive, negative (preceded by -), or function calls (preceded by $). Variables begin with the character %.

The following example shows the contents of a concrete knowledge base. There are three concepts for the terms **astleitung**, **astansatzleitung**, and **stammleitung**. Each of the fist two concepts contains only a clause in the analysis part. The last concept has additionally three production rules in the systhesis part. Further, there are not empty partialization and specialization relations.

```
(CONCEPTS
  ((astleitung (mu-sh) 9409050935
    ((('astleitung o) ('leitung o)
       (+ ('x-leitung o)
          ('y-leitung o))
       ('astleitungsebene o1)
       (%('aoinclude o1 o))))
    ())
   (astansatzleitung (dd-ul)
    9409050948
    ((('astansatzleitung o)
       ('stamm-anschluss s)
       ('leitung o1)
       (+ ('x-leitung o1)
          ('y-leitung o1))
       (%('aocontact s o1))
       ('leitung o2) ('z-leitung o2)
       (- ('stammleitung o2))
       (%('aocontact o1 o2))
       ($('aocompound o1 o2 o))))
     ())
   (stammleitung (dd-wo) 9409050910
    ((('stammleitung o)
       ('leitung o) ('z-leitung o)
       ($('aobject-dz o dz))
       ('deckenhohlraum d)
       (%('aooverlap o d))
       ($('aobject-dz d ddz))
       (%('> dz ddz))))
    ((1((stammleitung %l)
        ($('aobject-aspect %l %la)))
       --->
       ((stammleitungsaspekt %la)))
     (2((blattleitung %l)
        ($('aobject-aspect %l %la))
        (-(stammleitungsaspekt %la))
        (anschlussort %o)
        ($('neue-stammleitung %la %o %l1)))
       ---> ((stammleitung %l1)))
      (3((stammleitung %l1)
         (stammleitung %l2)
         ($('aoinclude %l1 %l2 t)))
        --->((-(stammleitung %l2)))))))))

(SCHEMES nil) (CASES nil)
(PARTIALS ((astansatzleitung (leitung))))
(REALS nil)
(SPECIALS ((leitung
            (astleitung stammleitung
             astansatzleitung))))
(ASSOCS nil)
```

**Behaviour Base (ABB):** With the shorttime data base and the knowledge base, a set of passive structures and corresponding operations are defined. Using the whole Lisp facility and some special func-

tions, behaviour structures can be defined that determine which operation is to be applied on which data structure to solve which problem. These behaviours are stored in DOM in a separate behaviour base. So, it is possible to modify them if necessary analogously to the elements of the other bases of the system. The behaviour base has the following structure given as class definition.

```
(defclass abb ()
  ((queries) (updates)))
```

In the system, an arbitrary number of instances of this class can be created. Each slot of the behaviour base contains as value a set of further structured elements. The element structure is given by the following definitions.

- Queries: set of query functions or methods; each entry is determined by the query name (name), the list of used variables (calllist), and the definition of the query function or method (definition)

  ```
  (defstruct (aquery (:type list))
     name calllist definition)
  ```

- Updates: set of update functions or methods; each entry is determined by the update name (name), the list of used variables (calllist), and the definition of the function or method (definition)

  ```
  (defstruct (aupdate (:type list))
     name calllist definition)
  ```

The following example shows the contents of a behaviour base. The first behaviour describes the computing of the include relation for a given layout, the second one determines not allowed collisions in a layout.

```
(UPDATES
  ((build-aincludes (adb)
    (defmethod build-aincludes ((adb adb))
      (!join (objects adb)
        #'(lambda(o1 o2)
            (and (not (equal o1 o2))
              (aoinclude o1 o2)
              (def-ainclude adb
                (list(aobject-id o1)
                  (aobject-id o2)))
              nil))
        (objects adb)) t))))
(QUERIES
  ((assess-acollision (adb)
    (defmethod assess-acollision ((adb adb))
      (!restr (overlaps adb)
        #'(lambda (ovl)
            (!subset ovl
              (!restr-proj
                (oconcepts adb)
                #'(lambda (s)
                   (equal 'leitung
                     (second s)))
                #'car))))))))
```

## 7 Discussion

Finally, some thesis are formulated that can be regarded as general experiences collected during the work with the knowledge model and the developed application systems.

- Configuration without a solid amount of generic background knowledge of the domain representing the semantics of the universe of discourse can not be successful.
- It is not possible to achieve a complete generic knowledge model of a configuration domain. Other techniques (like case-based, learning, but also traditional manual and algorithmic methods) are necessary to fill the gaps and to make the model dynamic.
- Configuration tasks demand a close interaction between analysis and synthesis methods. Synthesis fails without a powerful analysis.
- One-level knowledge models are not suited for configuration tasks. Models on different specialization, partialization, and realization levels have to co-operate with each other.
- Configuration systems can not work adequately and efficiently using a pure knowledge representation language. Different representational formalisms – including the basic programming language for the worst case – must be used.
- With a knowledge-based approach, one gets a rather declarative working model of the domain after a certain acquisition phase. But, the longer the model exists and is in practical use, the more submodels are transformed into more procedural representations for the reasons of exactness and efficiency.
- In each of the handled configuration domains, a taxonomy of technical terms as well as procedures for their application are available. They can be used directly for structuring the knowledge and guiding the problem solving process. Using these specific characteristics of the domain, a lot of general technical and representational problems can be avoided.
- Knowledge models for configuration tasks must have a close connection to design models actually used in concrete application domains. The interface between the knowledge model and the design model must be organized knowledge-based, too.

## 8 Conclusion

The presented multi-layered hybrid approach is a useful way to handle configuration domains of practically relevant sizes. The developed application systems, especially the building application system DOM, prove this fact. This way involves a compromise between the adequate representation of a universe of discourse on the one hand and the efficient implementation of a system on the other hand. Of course, the main focus within this compromise is shifting during the time of existence of a system. To improve this compromise, further investigations will

be necessary in the theoretical foundation as well as the practical use of the model.

## References


[1] Bakhtari, S.; Bartsch-Spörl, B.; Oertel, W.: DOM-ARCADE: Assistance services for construction, evaluation, and adaptation of design layouts. in: Gero, J. S.; Sudweeks, F. (Eds.): AI in Design'96. Kluwer Academic Publishers, Dordrecht, Standford, 1996

[2] Bakhtari, S.; Bartsch-Spörl, B.; Oertel, W.; Eltz, U.: DOM: Domain Ontology Modelling for Architectural and Engineering Design. FABEL Report Nr. 33, GMD, Sankt Augustin, 1995

[3] Börner, K. (Ed.): Modules for Design Support. FABEL Report Nr. 35, GMD, Sankt Augustin, 1995

[4] Haller, F.: ARMILLA - ein Installationsmodell. Institut für Baugestaltung, Baukonstruktion und Entwerfen, Universität Karlsruhe, 1985.

[5] Hovestadt, L.: A4 Digital Building: Extensive Computersupport for the Design, Construction, and Management of Buildings. FABEL Report Nr. 15, GMD, Sankt Augustin, 1993

[6] H. Lein: Schnittstelle AutoCAD - DOM. Diplomarbeit, Technische Universität Dresden, Fakultät Informatik, 1996

[7] Oertel, W.: Eine funktionale Methode der Wissensrepräsentation. Dresden, Techn. Universität, Fak. Elektrotechnik-Elektronik, 1988, Dissertation A

[8] W. Oertel, S. Bakhtari: Interaction of Generic Knowledge and Cases in DOM. In: Proceedings of the Third Congress on Computing in Civil Engineering, Anaheim, USA. ASCE, New York, 1996

[9] W. Oertel: FAENSY: Fabel Development System. FABEL Report Nr. 27, GMD, Sankt Augustin, 1994

[10] Oertel, W.: Zur Integration von fall- und regelbasierten Verfahren im Entwicklungssystem FAENSY. in: Burkhard, H.; Lenz, M. (Eds.): Fourth German Workshop on Case-Based Reasoning: System Development and Evaluation, Informatik-Berichte, Berlin, Humboldt-Universität, 1996

[11] Schmidt-Belz, B. (Ed.): Scenario of FABEL Prototype 3 Supporting Architectural Design. FABEL Report No. 40, GMD, Sankt Augustin, 1995

# Fuzzy Logic in Configuration

**Ilka Philippow[1], Fred Roß[1], Ulf Döring[2]**

[1]Technische Universität Ilmenau
Fakultät für Informatik und
Automatisierungstechnik
D-98684 Ilmenau
fred.ross@systemtechnik.tu-ilmenau.de

[2]R3M Softwarebüro
Unterpörlitzer Straße 8
D-98693 Ilmenau
Tel (+49 3677) 670295
Fax (+49 3677) 840578

The authors research on development of fuzzy based systems for years, in which systems for control of complex and/or non-linear systems (fuzzy controller, fuzzy classificators) as well as large knowledge based systems (diagnoses and management systems) were investigated. Regarding our experiences especially in the latter field we want to discuss to what extent the use of fuzzy based techniques to solve complex configuration tasks is possible and sensible.

In principle a configuration task may be divided into the two spheres „automated configuration" and „man/machine communication". In both spheres we see (partly different) possibilities to use fuzzy technology.

## 1 Fuzzy Logic

### Semantics of Degrees[3]

The degrees (values between 0 and 1) processed using fuzzy logic may have different semantics, usually uncertainty and fuzziness. But it's very possible that they carry the semantics of weights or priorities. The concrete semantics of a degree depends on its source as well its further employment in algorithms.

If a degree d is calculated as correctness of a premise (eg „x is high", the degree of membership of x to the set of high ones is calculated according to a membership function) it's called fuzziness. In this example a degree d = 0.8 means that x is relatively high.

Degrees of uncertainty may caused by (automatically) detected violations of integrity or by explicit definition done from outside of the system. Degrees of uncertainty can carry the semantics of possibility, probability or certainty, where the possibility is an upper bound and the certainty is a lower bound of the probability of a premise (refer to [Zimmermann91]).

[3] The term „degree" is used as generic term for „degree of probability", „degree of fuzziness"... An alternative may be the term „measure".

But mostly the original semantics is ignored and degrees of different semantics are mixed. Those systems are not so flexible, however they are often sufficient to solve simple tasks. Nevertheless to solve complex problems in a proper way extended modelling methods and extended information flows are necessary, see [Döring94] or [Philippow96].

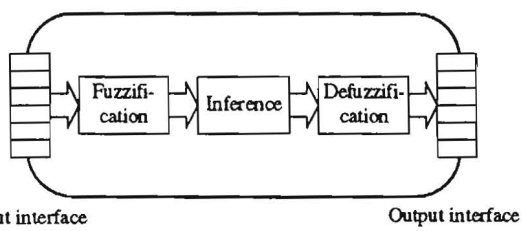


fig.1: Common flow of information in a fuzzy system

### Choice of algorithms and operators

For the transformations made in the flow of information in a fuzzy system (see fig.1) - which can be part of a greater knowledge based system - a lot of standard algorithms and operators exist, see [Driankov93], [Mayer93]:

- fuzzy operators (t-norms, t-conorms, average operators and other), which may have parameters. They are used to combine degrees to a new (overall) degree. Important properties of operators, which have to be taken into account, are for instance null dominance for t-norms (t(0,d1,d2,..dn)=0) and one dominance for t-conorms (s(1,d1,d2,..dn)=1)
- weighted fuzzy operators (see [R³M96] or [Philippow96]), which are used if weights are attached to the degrees. These weights may be constant (set during modelling) or calculated during inference. They are useful to make differences between the premises of a rule.
- defuzzification algorithms, which are used to compute an exact value based on the degrees estimated during inference

- comparison operators, they can check premises like „x is (equal to) middle" and other relations eg „x is smaller than middle". If the value x is not exact but fuzzy itself, special algorithms may be defined to handle different semantics of comparisons like „x is *somehow* grater than middle" or „x is greater than middle *on average*"

For adequate representation in large systems (eg configuration systems) the variety has to be used. If in special cases no algorithm or operator gives proper results a new one (suitable) has to be designed and to be attached to the development tool (if the tool supports such extensions).

**Membership functions**

Linguistic Terms („small", „seldom") can be assigned to linguistic variables for representation of properties (eg „width", „appearance"). The linguistic terms are defined by means of membership functions (msf). A msf is used, if an exact value is to be transformed into a degree (membership to a fuzzy set, which is represented by a linguistic term).

To define the shape of a membership function a lot of different types exist. Often parametrized standard types are used, because they can be described easier and may allow optimized algorithms (fig.2).

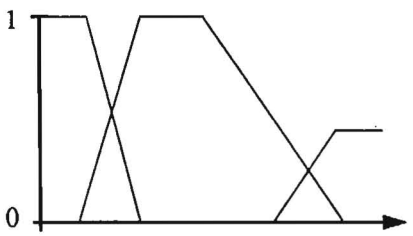


fig.2: Example for a trapezoid shape, the height of the trapezoids is one parameter

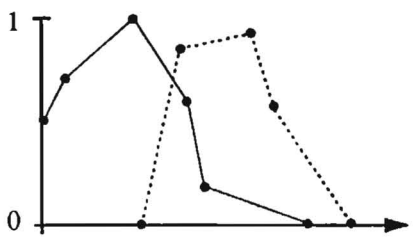


fig.3: Example for a definition by means of free setable control points

More complicate shapes are modelled using control points. Between these points (usually linear) interpolations are made (fig.3). An exclusive use of eg simple trapezoid shapes (as usual for controllers) may soon lead to inadequate results.

**Fuzzy constraints**

The usefulness of constraints in modelling configuration problems seems to be widely accepted, especially if the variety of possible forms is taken into account, eg tables, functions, (in-)equations or rules. Constraints are (always dependant on the abilities of the algorithms) used in different ways. A constraint like

$$a = [0.2...0.5]*b \qquad (1)$$

eg:

- to estimate a value for a if b is known
- to assign an interval to a if b is known
- similar for b if a is known - this requires that the inversion of (1)

  $$b = [2...5]*a \qquad (2)$$

  can be found
- to check the fulfilment of the relation between a and b modelled in (1) if a and b are known

The constraint example (1) models only a simple version of fuzziness - an interval definition, which can be handled using interval arithmetic. Considerable more possibilities gives the use of fuzzy numbers and fuzzy intervals, see [Mayer93] or [Zimmermann91].

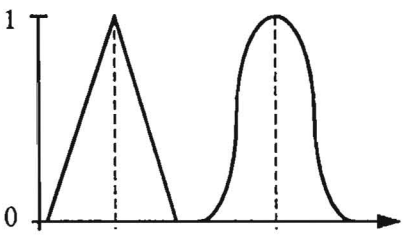


fig.4: Example for fuzzy numbers

Furthermore the check of fulfilment of constraints can be done in different ways. Conventional algorithms will return a binary result. The corresponding conventional constraints shall be seen here as hard constraints, because they have to be fulfilled completely. With soft constraints (fuzzy constraints) can be modelled:

- that the hart fulfilment is striven for in a certain degree, that has the consequence that the hart non-fulfilment of the original hard constraint decreases the quality in a certain degree (so we have weights or sentences for not fulfilled hard constraints)
- or constraints are not hard (and therefore soft) in the sense that a degree of fulfilment can be calculated (possibly in combination with weights) and so a quality for the whole configuration or for a part of the configuration can be computed (using operators). These qualities can be compared and the best configuration may be selected. Soft constraints

like „a is about b" may be used to model rules of thumb, where „about" is defined as msf with variable centre (similar to fig.4).

In addition to the use in checks the softness can be used in assignments. During assignments the instances of variables (eg a from the example above) can store information about the softness of the used constraints too. If in (1) b is equal to 20 and for the constraint was defined a probability = 0.8, then the assigned value (a=[4..10]) has a probability of 0.8 too.

## 2 Automated Configuration

The use of conventional methods in modelling knowledge about a problem and about the way of solution forces developers to assign exact values to variables even if the exact values do not exist or are unknown. The developers have to estimate the parameters or to define them arbitrarily. So the search space for a valid solution is cut. Mostly this has the advantage of a faster inference but possibly the search space is cut in a way that only poor or sometimes no solutions are found. Concepts like „unknown" are a binary step into the „right" direction (extend of possibilities to model). From the fuzzy logical point of view „unknown" can mean „any value with no degree of membership, certainty or probability". The semantics „with no degree of possibility" would not be correct because if „unknown" is assigned to a variable each value keeps possible.

### Choice of Components

In the representation of knowledge about a problem (eg using msf) as well as during solution of configuration tasks (execution of appropriate algorithms) the consideration of known fuzziness or / and uncertainty can lead to better results.

So an adequate choice of components can be reached if not only a binary „suitable" or „unsuitable" is used but also a degree of suitability. Such degrees can be defined as single values (regarding possible choices) or as membership functions (according to a design parameters represented as number).

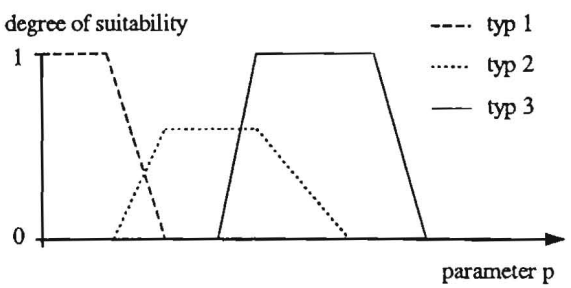


fig.5 Example for different suitability for 3 types of a component according to a parameter p

In fig.5 is shown, how membership functions can be used to model different suitabilities of 3 types of a component according to a parameter p which is assigned based on the state of a certain configuration during configuration process. As you see type 2 is not so suitable at all however in the middle no alternative exists. Thus when suitabilities of different configurations are compared such configurations with a p that is in the middle will be estimated as not so suitable.

How serious this unsuitability influences the overall suitability depends on the choice of weights and operators. So the estimation of a group of components could be calculated as the minimum or a kind of average of the suitabilities of its components (eg arithmetic mean or geometric mean). During selection of an operator some decisions have to be made, eg whether a total unsuitability (represented by 0) leads to the total unusability of the whole group or not. Such restrictive behaviour (null dominance) can be reached using t-norms (eg minimum or algebraic product). In this place we want to point out that the often used min-max-inference (use of minimum and maximum in the according inference steps) leads to a leak of information which is mostly not wanted. Thus the suitability of operators has to be proved in each case.

If the single suitabilities $p_i$ are to be accumulated differently in the overall suitability then weights (numbers between 0 and 1 too, but dependant on the algorithms other domains may be possible) can be assigned to the $p_i$. These weights have influence on the result according to the chosen scheme of computation. In the following differences between possible schemes of computation shall be shown. As operator the algebraic product was chosen,

$$\prod_{i=1}^{n} p_i \qquad (3)$$

because it's null dominant and (in contrast to the minimum) it always leads to an above-average decrees of overall suitability if components are not full suitable (example: 0.5*0.5 = 0.25). No information about decreased suitabilities of components is lost.

For the use of weights generally two approaches exist [R³M96]. To simplify the examples it should be assumed that the weights $w_i$ of the values $p_i$ which have to be accumulated are normalized ($w_{max}$=1). The first approach assumes that a decreased weight (smaller than 1 or rather than $w_{max}$) causes that the according values *fall out* of the computation and therefore the number of original n values is „fuzzy" decreased. For the algebraic product this leads to the fact that if some weights are smaller than 1 the result is never smaller than the result got if all $w_i$ are 1. Because the unweighted algebraic product (3) is a special case ($w_i$ = 1 for all i) of the weighted versions (eg (4) and (5)) these weighted versions never return results which are smaller than the result of (3). The behaviour of *falling out* is reached by producing neutral elements according to the $w_i$ (for t-norms like the algebraic product 1 is the

neutral element). Two simple algorithms of the approach shall be shown:

$$\prod_{i=1}^{n}\left((p_i - 1)\cdot w_i + 1\right) \qquad (4)$$

$$\prod_{i=1}^{n} p_i^{w_i} \qquad (5)$$

In equation (5) the null dominance of the $p_i$ is kept if the $w_i$ is not 0 (see example No.2 in tab.1). This is mostly wanted and therefore the linear version (4) is only good for special cases.

| | | i | | | result according to | | |
|---|---|---|---|---|---|---|---|
| No. | | i = 1 | i = 2 | i = 3 | (3) | (4) | (5) |
| 1 | $p_i$ | 0.5 | 0.5 | 0.5 | 0.125 | 0.404 | 0.379 |
| | $w_i$ | 0.1 | 1.0 | 0.3 | | | |
| 2 | $p_i$ | 0.0 | 0.5 | 1 | 0.0 | 0.45 | 0.0 |
| | $w_i$ | 0.1 | 1.0 | 0.3 | | | |
| 3 | $p_i$ | 1.0 | 0.8 | 0.2 | 0.16 | 0.608 | 0.494 |
| | $w_i$ | 0.1 | 1.0 | 0.3 | | | |

tab.1: Examples for the influence of weights

The other approach assumes that dependant on the $w_i$ the according values are replaced by a mean (eg weighted arithmetic mean). This approach is not so good for accumulating weighted suitabilities because the replacement by a mean of the values assumes that values are dependant from each other. But the suitabilities of the components are independent and so the first approach is better for this accumulation. A replacement by means is useful for instance if the values are estimations of the value of the same variable and the weights are the according degrees of uncertainties.

As example for choices using membership functions refer to [Arnold95] too. There is shown how bearings are chosen according to the degree of their suitability for specific loads of bearings.

#### Choice of Parameters

As shown above fuzzy techniques can be used to model choice of components. In addition to that modelling the choice of parameters may be supported too, especially if the knowledge are only rules of thumb.

Provided that all needed data can be accessed, according to the rules of thumb a fuzzy rule base can be formulated and then optimized in a suitable development tool [Roß94]. Such a rule base would make inferences as shown in fig.1 and could be encapsulated into functions and be attached to the configuration tool.

Furthermore fuzzy (and possibly weighted) constraints could be used to model knowledge about parameters. These constraints describe the aim of configuration. The configuration that fulfils the set of constraints in the highest degree is the best configuration (the best choice of parameters). So we have to solve an optimization task - the parameters are the inputs of the quality function, the maximization of quality (of fulfilment of the constraint set) is the aim of optimization. In contrast to hard constraints the quality function doesn't return „fulfilled" or „not fulfilled", but it returns the degree of fulfilment for a certain parameter set. To solve the optimization task several algorithms can be used. For large parameter sets it could be necessary to develop special optimization (i.e. problem solving) strategies which take problem specific aspects into account to reduce search costs.

Even to model problem solving strategies fuzzy techniques can be applied to - at least if linguistic variables (eg search spaces for parameters and step sizes of search), linguistic terms (eg „large", „right") or distributions of search power are to be specified.

## 3 Man / Machine Communication

The exchange of information between man and computer takes place based on terms, numbers and if applicable graphics. Regarding a configuration task usually large amounts of data describe a problem / the actual state of the solution. These data can hardly be represented as pure numbers. Where structure and semantics of data support a visualization that should be done in an appropriate way because this kind of representation is mostly more effective than the other ones (pictures are more expressive than numbers).

However for abstract assessments the transformation form numbers to terms is better. Corresponding transformations can be defined based on linguistic variables / linguistic terms. The aim is to build a vocabulary for description of properties of configuration objects. This vocabulary makes it possible to use (a certain number of) terms from human language to formulate questions to the system or to process automatic formulation of results in a human like form. This is especially important if the receiver of the data is no expert (a designer) but a layman (a customer) which can hardly estimate the sense of specific numbers. For a layman the statement „capacity is high" would be better than a statement „capacity is 1.7GB".

The choice from components or groups of components from a data base may be supported, by that that in questions linguistic terms are allowed and transformed. So according to a choice:

„LIST FROM motors WITH price=middle"

the list of motors sorted by membership to the motors with a middle price can be returned, possibly only up to a certain minimum membership. Especially the first example illustrates that it's sometimes necessary to maintain membership functions because it's possible

that the meaning of linguistic terms moves (What is a high capacity nowadays?). Furthermore possibly different user types or contexts of use have to be taken into account which demand different sets of membership functions. If there are problems to accept predefined shapes of membership functions these should/could be modified by the user according to his ideas. Displaying membership functions and allowing their adaptation misunderstandings can be avoided or rectified.

The representation of linguistic terms and the realisation of corresponding fuzzy choices in KONWERK, a workbench for solving design or configuration tasks, is exemplarily shown in [Müller95].

While the use of linguistic terms, i.e. their adaptation to different user, can be done in a proper way - the choice of operators is a problem. Regard following query:

„LIST FROM motors WITH price=middle AND revolutionsPerSecond=high"

Which operator should be used to perform the AND? Depending on the chosen operator different motors could be selected. During definition of knowledge bases appropriate operators may be found with trail and error method. Because the knowledge base is usually longer in use and it is worth the cost of finding a good operator - trail and error may be a proper way, but during formulation a query the user can hardly decide always which operator fits his question best. At last a standard operator will be chosen which during validation time of the system was proved to be worthwhile for a group of questions.

## 4 Summary

From our point of view systems which are based on fuzzy logic are far better adaptable to configuration knowledge and different user types than binary logic or the exclusive use of a degree of certainty or fuzziness allows it, because the variation of membership function parameters and shapes, parametrized operators and weights are a stronger mean to describe fuzziness in a human sense. In contrast to modelling membership functions the choice of operators is not so clear and demands higher effort so that here proved standard operators are often used.

**Literature:**


[Arnold95] Olaf Arnold, Konfigurierung, Auswahl und Auslegung von statisch belasteten, hydrodynamisch geschmierten Gleitlagern in KONWERK, in [PuK95] pages 165-169

[Döring94] Ulf Döring, Inferenzmechanismen unter Berücksichtigung von Unschärfe und Unsicherheit, Diplomarbeit 200-94D-08, TU Ilmenau, Fakultät für Informatik und Automatisierung, Institut für Theoretische und Technische Informatik, 1994

[Driankov93] Dimiter Driankov, Hans Hellendoorn, Michael Reinfrank, An Introduction to Fuzzy Control, Berlin: Springer 1993

[Mayer93] Andreas Mayer, Bernhard Mechler, Andreas Schlindwein, Rainer Wolke, Fuzzy Logic, Bonn: Addison-Wesley 1993

[Müller95] Karin Müller, Fuzziness in KONWERK, Proceedings of EUFIT'95, Aachen, Germany, august 95, pages 1245-1250

[Philippow96] Ilka Philippow, Fred Roß, Ulf Döring, Methods to handle Large Fuzzy Rule Bases, Proceedings of EUFIT'96, Aachen, september 96

[PuK95] Susanne Biundo, Wolfgang Tank (eds.), PuK - 95, Beiträge zum 9. Workshop "Planen und Konfigurieren", DFKI-D-95-01

[$R^3$M96] Ulf Döring, Ableitung gewichteter Varianten n-stelliger Operatoren, $R^3$M-Forschungsbericht 4/96

[Roß94] Fred Roß, Ulf Döring, FuzzyEXPERT - Ein Entwicklungssystem für Fuzzy-Systeme zur Entscheidungsfindung, 39. Internationales Wissenschaftliches Kolloqium, Ilmenau, 1994

[Zimmermann91] Hans-Jürgen Zimmermann, Fuzzy set theory and its applications, Boston: Kluwer Academic Publishers 1991

# Knowledge representation in process engineering

**Ulrike Sattler**
RWTH Aachen, uli@cantor.informatik.rwth-aachen.de

## Abstract

Process engineering is surely no pure configuration application, but modeling the structure of chemical processes confronts us, in the field of knowledge representation, with similar problems. First, the tasks we are concerned with in process engineering are described as well as how knowledge representation systems can support these tasks. Roughly speaking, this support consists in helping the user in the handling of an object-oriented database. Then it is argued why terminological knowledge representation systems are suitable tools for giving this support and how this support can be realized by these systems. In the last section, we describe some problems that arise because this task asks for a knowledge representation system with special expressive power.

## Process engineering

Process engineering is concerned with the design and operation of chemical processes that take place in huge chemical plants. This engineering task includes activities like deciding on an appropriate flowsheet structure (e.g. reaction and separation system configurations), mathematical modeling and simulation of the process behavior (e.g. writing down mathematical equations and performing numerical simulations), sizing of components like reactors, heat exchangers etc. as well as costing and engineering economics. All these tasks are based on appropriate models of the process that is to be designed or operated. These models can be different graphical models, verbal or mathematical models. To support these engineering tasks by appropriate software tools, the development of process models has to be supported. The process models are based on standard building blocks [Marquardt,1994; Bogusch and Marquardt,1995] which are objects representing, among others,

- material entities such as reactors, pipes, control and cooling units,
- models of these entities such as device-, environment-, and connection-models,
- interfaces between these models and so-called implementations describing their behaviour,
- symbolic equations specifying these implementations and variables occuring in these equations which are related to each other as specified in the interfaces,
- ...

Our aim is to support the development of these models. The task of modeling chemical processes is surely no pure configuration task, but especially the modeling of the structure of a chemical plant confronts us with similar problems and tasks: Devices and connections are chosen, their respective interfaces are coupled, complex devices are decomposed into their components or segments, etc..

## Problems we want to help with

The highly complex task of modeling chemical plants can be heavily supported by appropriate software tools such as CAD, decision support and numerical tools. In order to give this support, the domain specific knowledge is stored in a frame-based system. This frame-based system is able to store a great variety of the standard building blocks. As the user has to be able to find building blocks (s)he is looking for, standard building blocks are grouped in classes, and these classes are ordered with respect to the *is-a-specialization-of* relation (known also as the *is-a* relation) which yields the class hierarchy. This ordering has to be explicitely stated in each class definition by giving, for each class, the set of its superclasses. As the frame-based system includes powerful features such as methods and triggers, it is far too expressive to compute the implicit subsumption relation on the defined classes. On the other hand, it is flexible in that it can be extended by additional classes of building blocks. This second feature is necessary because in process engineering, the number of standard building blocks increases permanently.

In the sequel, by database we refer to the set of class definitions in the frame-based system. As the complexity of the database increases, navigation in its hierarchy becomes again difficult and modifying or extending the taxonomy becomes dangerous in the sense that they might not yield the desired changes. In fact, the user (the person building models and sometimes extending the database by new classes of standard building blocks) is confronted with the following problems:

1. Navigation in the class taxonomy will become difficult, especially in those parts of the data-

base not often used by the user. Searching for a certain class whose names is not known may take a long time of browsing the hierarchy and comparing different class definitions until the appropriate class is found.

2. Definig a new class $A$, the user has to arrange it into the existing taxonomy according to its intuition or common sense. (S)he knows that $A$ is a subclass of $B$, but might be uncertain whether there is a more specific subclass $B'$ of $B$ such that $A$ is a subclass of $B'$. Because of this uncertainty it is rather probable that the taxonomy gets broader than necessary—which is, on one hand, disadvantageous for the performance of the database system, and, on the other hand, makes navigation more difficult than necessary. Furthermore, it could happen that the user defines an inconsistent or unintendent class. The extension of the database by such a definition can cause needless work.
3. As the database can be modified by more than one user, it is probable that the same class is defined twice—in syntactically different terms and with different names. This does not only blow the size of the database, but is also a source of misunderstanding and trouble.

## How these problems can be solved

To help the user with these problems, the database should be equipped with a system that is able to compute implicit specialization relation between defined classes and that is able to test consistency of class definitions. Unfortunately, the frame-based system has far to much expressive power to allow for this automatic reasoning, e.g., the according inference problems are undecidable. The main reason for this fact is the possibility to define triggers and powerful methods in the frame based system.

Fortunately, there is still something that can be done: The content of the database can be mirrored in a knowledge base whose reasoning services are powerful enough to help the user with the problems mentioned above. As a consequence of the above observation, this translation cannot be exact—if it were exact, the interesting problems would still be undecidable—but, by choosing an appropriate knowledge representation system, they can be sufficiently exact. An important point of this mirroring is that the taxonomy of the knowledge base has to be equivalent to the class hierarchy of the database. Even if some properties described in the database cannot be translated accordingly, this equivalence has to be assured. Then the knowledge representation system should be able to help the user with the navigation and modification of the database. It should include an intelligent browser to help finding classes, propose places in the taxonomy where to place a new class, clarify the meaning of a new class definition before the database is extended by this class, and detect semantically identical classes.

## Why we chose a TKR-system

In this section, we will argue why a *terminological knowledge representation system* (TKR-system) is the appropriate representation system for the task described above. Before doing so, we will briefly describe TKR-systems.

TKR-systems differ mainly in their underlying *description language*, which are characterized by the sets of so-called concept-forming and role-forming operators. Using these operators, one can define *complex concepts* (which are interpreted as sets of elements of the interpretation domain) and *roles* (which are interpreted as binary relation on the interpretation domain) using primitive concepts and roles. Operators available in almost all implemented systems are union, intersection, negation, value restrictions, as well as restrictions on the number of role successors. A terminological knowledge base is a set of concept and role definitions stored in a so-called TBox. A small example for a TBox is given in Figure 1. In this TBox, the concepts `Material-Entity`, `Model`, and `Implementation` are defined (for a matter of space, these definitions are presented only partly). For example, a `Material-Entity` is a `Modeling-Concept` that is associated by the `is-modeled-by` relation to instances of the concept `Model` only, and by the relation `has-function` to instances of the concept `Function` only. A `Model` is, among others, associated by the relation `is-implemented-by` to exactly one `Interfaces`. The concept `Model` are further refined, for example, to concepts like `Device-Model` or `Connection-Model`, which themselves are refined, and so on.

TKR-system are suitable for this task because of the following points:

- TKR-systems can be viewed as a unified framework for class based representational formalisms [Calvanese *et al.*,1994], and are closely related to frame-based systems. The translation from a class definition in a frame-based database to a concept definition in a TBox is natural for many of the properties describable in frame-base systems, hence this translation can be performed automatically.
- For most description languages, there exist sound and complete inference algorithms for the answering of queries. In most cases, these queries are reduced to the basic inference problems such as satisfiability (the question whether a concept can ever be instantiated) or subsumption (the question whether a concept is more general than another one). Soundness and completeness of the inference algorithms implemented in a system imply that queries are always answered correctly after a finite amount of time. The advantage of TKR-systems with sound and complete inference algorithms is that, if the user explicitely describes properties of objects, then these properties are always dealt with by the algorithm—they are not simply disregarded when the algorithm reasons about these objects.
- It is possible to keep the TBox taxonomy equivalent with hierarchy of the database: There are two reasons why a concept could be placed at a different place in the (implicit) taxonomy

```
Material-Entity  :=  Modeling-Concept ⊓ (∀is-modeled-by.Model) ⊓ (∀has-function.Function)
Model            :=  Structural-Modeling-Concept ⊓ (∀possible-alternative.Model) ⊓
                     (∀active-alternative.Model) ⊓
                     (∀is-implemented-by.Implementation) ⊓ (= 1 is-implemented-by) ⊓
                     (∀active-interfaces.Interfaces) ⊓ (≥ 1 active-interfaces)
Implementation   :=  Structural-Modeling-Concept ⊓
                     (∀behaviour.Equation) ⊓ (≥ 1 behaviour) ⊓
                     (∀variables.Symb-vars) ⊓ (≥ 1 variables)
```

Figure 1: Example TBox

of the TKR-system than the according class in the database hierarchy: It can be (1) because of the inexactitude of the translation and (2) because the user placed the class too high in the database hierarchy. If such a mismatch occurs, the user is asked to verify which of the cases did arise. In the first case, the definition of the concept is modified such that afterwards, this concept is placed correctly. In the second case, the superclasses of the new class are modified accordingly.

- The services required for the support of the modeller in the usage of the database can be achieved by TKR-systems. Standard services provided by TKR-systems comprise the calculation of the implicit subsumption relation between two concepts, the calculation of the implicit concept taxonomy, as well as testing whether a concept is satisfiable.

  Based on these services, navigation can be supported in the following way: First, the user is asked to describe—in an incomplete way—the class (s)he is looking for. Then the TKR-system gives him/her the most specific classes subsumed by this description. The user should then be able to give more information concerning the class (s)he is looking for by looking more closely at these classes. Naturally, this information can also include some of the classes proposed by the system which are more general than the one the user is looking for. By iterating this ask-and-tell procedure, the user is guided to the class (s)he is looking for.

  Before adding a new class definition to the database, the user can ask the TKR-system to arrange the according concept into the TBox taxonomy. Investigating this taxomony, we can prevent the user from unintended definitions.

  Testing satisfiability of a concept before adding its according class to the database can prevent from extensions by inconsistent classes.

- Its declarative semantics enables the user to correctly define the concepts (s)he has in mind. Rule based formalisms may seem more natural, but when characterising a class one has in mind, it is difficult to fix all rules necessary to define this class.

## Which TKR-system to choose?

TKR-systems differ in the expressive power of the underlying description language, and we are now confronted with the question *which* TKR-system is the most appropriate one for the task described above. In the last decade, a great variety of TKR-systems has been investigated [Levesque and Brachman,1987; Nebel,1988; Schmidt-Schauss,1989; Patel-Schneider,1989; Hollunder *et al.*,1990; Donini *et al.*,1991; Baader and Hanschke,1993; De Giacomo and Lenzerini,1994; Calvanese *et al.*,1995]. However, there are still many open questions concerning TKR-systems, their expressivity as well as their behavior in realistic applications. It is clear that, for a given application, the description language has to be expressive enough to represent relevant properties of the objects in the application domain. Unfortunately, the more expressive a description language language is, the more time or space is needed to compute query answers[1]. Hence a compromise has to be found between computational complexity and expressive power. Furthermore, as "expressive power" is not 1-dimensional, it is difficult to tell whether the expressive power of one description language is "better" for a given application as the expressive power of another one.

This process engineering application is surely no pure configuration application. Nevertheless, the structural modeling of a plant can be seen as a configuration task: Devices and connections are chosen from a set of generic devices; they are possibly modified according to the actual construction; connections between these devices have to be defined; they are possibly decomposed into their parts in order to get a more precise model; and finally, devices are aggregated from different subdevices modeled by different users. As a consequence, in the field of knowledge representation, we are confronted with problems which occur also in configuration applications:

**Part-whole relations:** As the plants to be modeled are very complex, the user should be able to decompose and aggregate devices and connections of the process to be modeled (this is also important for

[1] However, driven by demands from other applications, it could be shown in [Baader *et al.*,1994] that worst-case intractable languages may behave quite well in practice.

the reuse of models as well as for distributed modeling). Hence the TKR-system has to be able to represent composite objects appropriately.

For this appropriate representation of composite objects, part-whole relations have to be treated correctly by the inference algorithms of the TKR-system. As for other applications [Gerstl and Pribbenow,1993; Franconi,1994; Artale *et al.*,1994; Pribbenow,1995], we are confronted with the question

- which part-whole relations are needed for the appropriate representation of the complex objects in our application. It turned out that objects are decomposed with respect to the component–composite, segment–entity, and member–collection relation, each of them a spezialisation of the general part-whole relation. Roughly speaking, parts with respect to the member–collection are not coupled to each other and are "of the same kind", whereas components can be coupled to each other in any way and may be quite different one from each other, and segments are from a similar kind, but coupled to each other. As the user might want to refer to a part, not knowing on which level of decomposition it can be found, we have to represent the general, transitive part-whole relation as well.
- how these relations interact: If, in the intuition of the user, the segment–entity is transitive, then it has to be represented as a transitive role. But what about a component $a$ of a segment $b$ of a whole $c$ — ist $a$ also a component of $c$? Questions concerning these interactions are not yet completely answered, but they have to be answered in order to handle composite objects appropriately.
- which properties concerning part-whole relations are relevant in the application: For example, the existence of a certain part can be essential for the proper definition of the whole, in contrast to parts being optional; a part can be exclusive in the sense that it might be a part of at most one object $b$ without the possibility to be shared by other objects beside those having $b$ as a part; a part can be functional for an object in that this object does no longer work correctly if this part is broken; and many others more [Simons,1987]. The representation of these properties is quite useful because knowledge concerning these properties is required for powerful consistency-testing procedures: It thus can be verified, for example, if all essential parts are specified and, if this is not the case, either a suitable one can be determined or the user is informed on this missing part.

  As at least the general part-whole relation is transitive, an appropriate TKR-system has to be able to handle some kind of transitive relations. Using transitive relations, the user can refer to parts along a number of decomposition levels not known in advance or along any (finite) number of decomposition levels. Hence an interesting question is, in which ways transitive relations can be included into description languages and handled by their inference algorithms. We investigated this question for an expressive, well-known description language in [Sattler,1996].

**Number restrictions:** As for configuration problems, objects are often characterized by the number of objects they are related to by some relation. For example, we want to describe devices having at least 7 inputs or exactly 5 outputs. In description languages, this can be done using number restrictions as in

$$(\texttt{device} \sqcap (\geq 7\ \texttt{input})),$$
$$(\texttt{device} \sqcap (= 5\ \texttt{output})).$$

This possibility is available in almost all implemented TKR-systems, but not sufficiently expressive for our application: We wanted to describe devices having *the same* number of inputs as of outputs, as in

$$(\texttt{device} \sqcap (= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output})),$$

or devices having less inputs as each of its parts have, as in

$$(\texttt{device} \sqcap (= \alpha\ \texttt{input}) \sqcap (\forall \texttt{has-part}.(> \alpha\ \texttt{input})).$$

In [Baader and Sattler,1996a], these *symbolic* number restrictions are introduced and investigated. Unfortunately, it turned out that the basic inference problems such as satisfiability or subsumption get very complex, even undecidable, if this kind of number restriction is allowed in a unrestricted way. Nevertheless, it could be shown that, if their usage is restricted, then we can reason in a sound and complete way about concepts containing symbolic number restrictions.

Furthermore, we want to restrict the number of objects that are related via a *complex* path of relations to an object. For example, we are interested in describing devices which have at most 7 parts that are components of its components, as in

$$(\texttt{device} \sqcap (\leq 7\ \texttt{has-component} \circ \texttt{has-component})),$$

or we want to describe a device where all the devices it is connected to are controlled by the same controller:

$$(\texttt{device} \sqcap (= 1\ \texttt{connected-to} \circ \texttt{controlled-by})).$$

The complexity of the basic inference algorithms depends on which operators, beside/instead of composition $\circ$ are allowed inside number restrictions. In [Baader and Sattler,1996b] it is shown that some combinations lead to undecidability of the basic inference problems whereas for other combinations, we could give sound and complete algorithms solving these problems.

## References


[Artale *et al.*, 1994] A. Artale, F. Cesarini, E. Grazzini, F. Pippolini, and G. Soda. Modelling composition in a terminological language environment. In *Workshop Notes of the ECAI Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 93–101, Amsterdam, 1994.

[Baader and Hanschke, 1993] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German AI-Conference, GWAI-92*, volume 671 of *LNCS*, pages 132–143, Bonn, Deutschland, 1993. Springer-Verlag.

[Baader and Sattler, 1996a] F. Baader and U. Sattler. Description logics with symbolic number restrictions. In *Proc. of ECAI-96*, 1996. To appear.

[Baader and Sattler, 1996b] F. Baader and U. Sattler. Number restrictions on complex roles in description logics. In *Proc. of KR-96*. M. Kaufmann, Los Altos, 1996. To appear.

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H. Profitlich. An empirical analysis of optimization techniques for terminological representation systems, or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.

[Bogusch and Marquardt, 1995] R. Bogusch and W. Marquardt. A formal representation of process model equations. *Computers and Chemical Engineering*, 19:211–216, 1995.

[Calvanese *et al.*, 1994] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of KR-94*, pages 109–120, Bonn, 1994. M. Kaufmann, Los Altos.

[Calvanese *et al.*, 1995] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of DOOD-95*, volume 1013 of *LNCS*, pages 229–246, 1995.

[De Giacomo and Lenzerini, 1994] G. De Giacomo and M. Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. In *Proc. of ECAI-94*, 1994.

[Donini *et al.*, 1991] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of KR-91*, Boston (USA), 1991.

[Franconi, 1994] E. Franconi. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, November 1994.

[Gerstl and Pribbenow, 1993] P. Gerstl and S. Pribbenow. Midwinters, end games and bodyparts. In N. Guarino and R. Poli, editors, *International Workshop on Formal Ontology-93*, pages 251–260, 1993.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *ECAI-90*, Pitman Publishing, London, 1990.

[Levesque and Brachman, 1987] H. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Marquardt, 1994] W. Marquardt. Trends in computer-aided process modeling. In *Proc. of ICPSE '94*, pages 1–24, Kyongju, Korea, 1994.

[Nebel, 1988] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Patel-Schneider, 1989] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *AIJ*, 39:263–272, 1989.

[Pribbenow, 1995] S. Pribbenow. Modeling physical objects: Reasoning about (different kinds of) parts. In *Time, Space, and Movement Workshop 95*, Bonas, France, 1995.

[Sattler, 1996] U. Sattler. The complexity of concept languages with different kinds of transitive roles. In *20. Deutsche Jahrestagung für Künstliche Intelligenz*, LNAI. Springer-Verlag, 1996. To appear.

[Schmidt-Schauss, 1989] M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proc. of KR-89*, pages 421–431, Boston (USA), 1989.

[Simons, 1987] P. M. Simons. *Parts. A study in Ontology*. Oxford: Clarendon, 1987.

# 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 $\lambda_{\mathsf{Vol}}\, c.\,(\ldots)$ is a unary predicate of a numeric concrete domain for the dimension *Volume* with base unit $m^3$):

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

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 tc

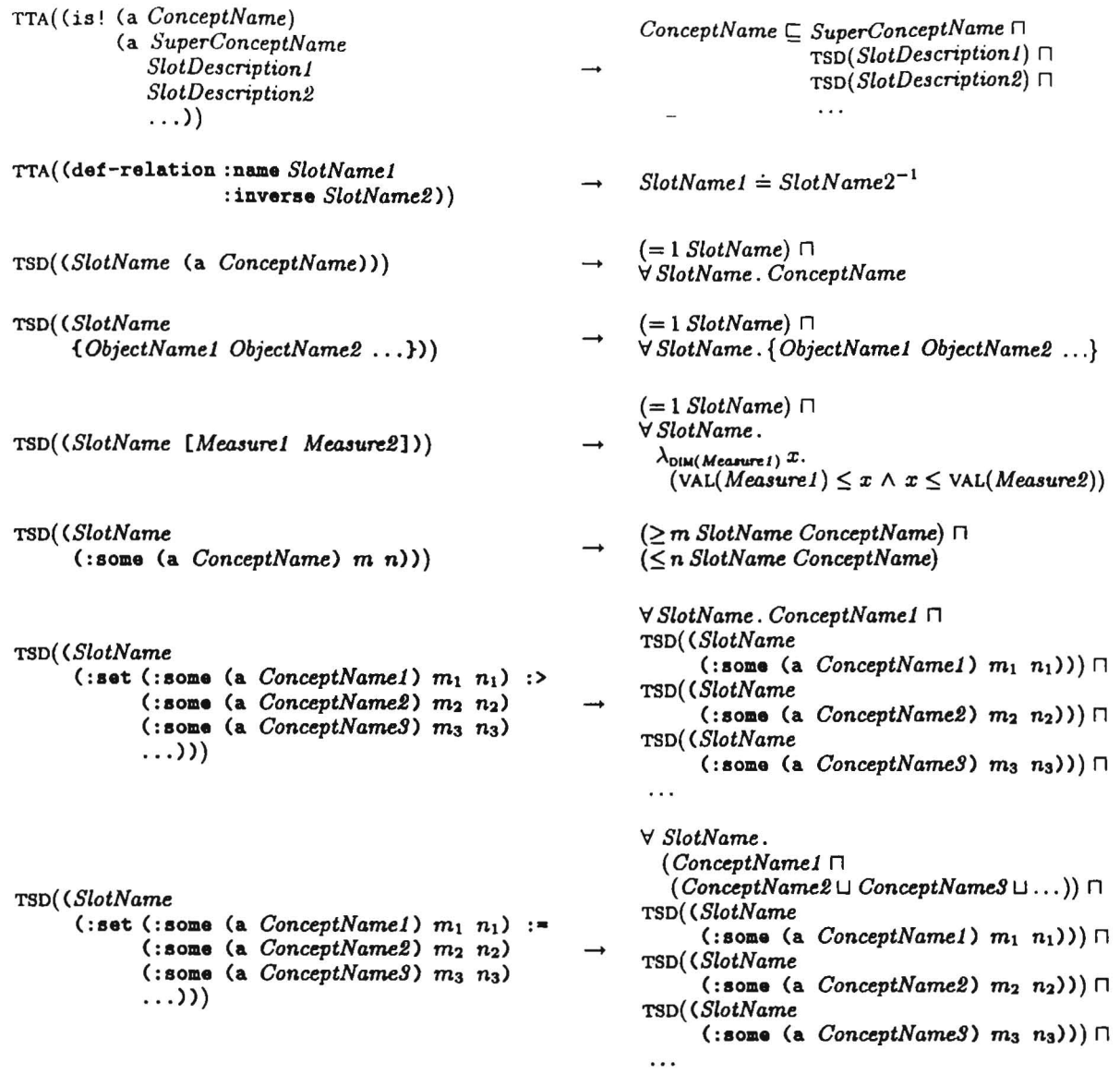


Figure 1: Rules for transforming a BHIBS terminology.

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.

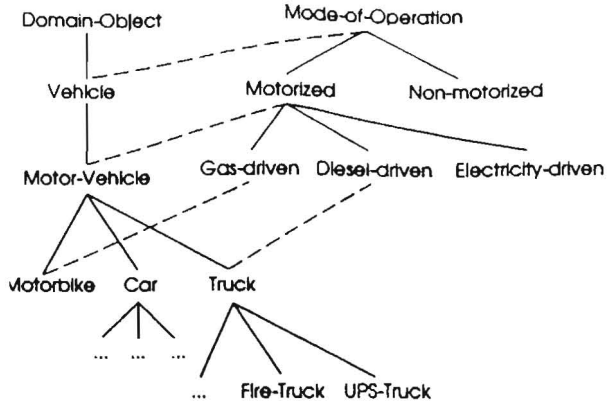


Figure 2: Example for a concept hierarchy with mixins.

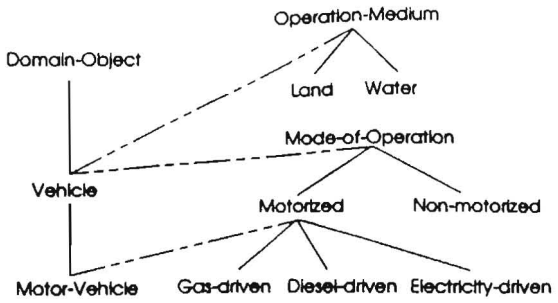


Figure 3: Example for a concept hierarchy with views.

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 × 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-

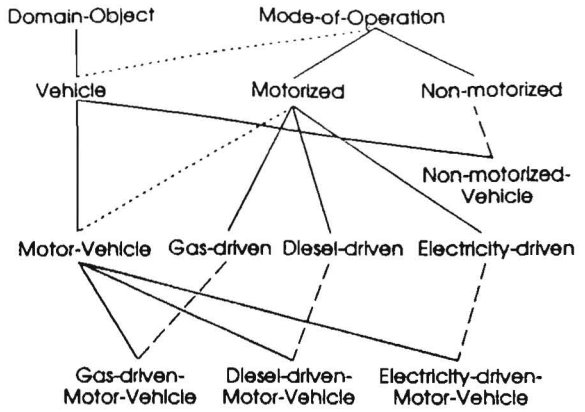


Figure 4: Expanded concept hierarchy.

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:

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

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.

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

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

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.

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

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.

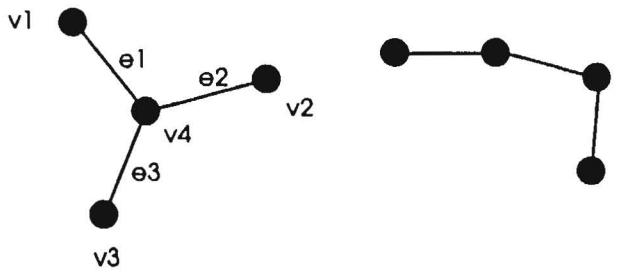


Figure 7: Two examples for configurations of vertices and edges: a star and a polyline.

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:

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

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

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

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.

# Using description logic for configuration problems

**Holger Wache**
Universit"at Bremen
P.O. Box 330 440
28334 Bremen
wache@informatik.uni-bremen.de

**Gerd Kamp**
Universit"at Hamburg
Vogt-K"olln-Str.30
22527 Hamburg
kamp@informatik.uni-hamburg.de

In order to use a model–based approach for describing configuration problems the following types of knowledge have to be represented:

- the *components* of which a configuration can consist. Generally the components are collected in a *catalog.*
- the allowed/necessary *relationships* between the components, that define a valid set of components in a configuration solution.
- a *specification* of the configuration problem to solve.

The configuration problem is solved, when a set of components is selected from the catalog in such a way, that the components satisfy the specifications and do not contradict any relationships.

The main difficulty in representing configuration problems lies not in modeling the components but in describing the relationships. In comparsion to other existing model–based configuration systems like PLAKON [RC91], KONWERK [G"u95a, G"u95b], or COSMOS [Hei93], the term "relationship" is sweeping. Several kinds of relationships can be identified in configuration problems:

- *part–whole taxonomy:* components are described as composition from other components
- *functional dependencies:* components can be considered as function providers. For providing these functions they need other functions provided by other components and so on.
- *physical laws in a technical domain:* such as $U = R * I$ or $m = r * f$
- *procedures* in the mechanical engineering domain there are procedures verifying and/or calculating values like temperature in crankshaft
- ...

Therefore, to describe components and their relationships a KR system needs a rich and expressive language. In our opinion, terminological systems provide such a language.

On the inference level of solving configuration problems, one important task is the selection of appropriate components from the catalog. Normally a component is not identified by its name but by its properties. So selection of components can be seen as a classical classification task.

The difficulty in the classification task lies in the way how the properties of a component are determined. The relationships between the components impose certain requirements (i.e. their properties) and constraints on the components that have to be obeyed. Computing the relevant requirements for a component have to be taken into account during the classification task. Desciption logics provides a powerful mechanism for classification. The main advantage is that procedures for evaluating the relationships can interact with the classification mechanism.

In our oppinion description logics are useful for solving configuration problems because they provide a rich representation language for modelling the components and their relationships and a powerful classi-

fication mechanism for the selection task. As best to our knowledge there is no other knowledge representation formalism that provides this two features on the base of a clear semantics. Many representation formalism allow the object centered representation of the components but do not directly support the classification task.

A terminological system can only act as a part of a configuration system. It is designated for representing the components and relationships and supporting a few inferences (i.e. classification). Representing other knowledge (e.g. control knowledge for guiding the configuration process) or additional inferences (e.g. aggregation of components) have to be done in the other part of the configuration system. The other parts (e.g. a rule based system for aggregation) use description logic as a service provider: description logic holds the (incomplete) problem solution and offers a few inferences.

In a terminological system the components and (a few) relationships are represented as concepts and relations in the T-Box. Our description formalism is based on the non–trivial $\mathcal{ALC}$ language (including OR and NOT) [HN90]. To describe component properties the language must be extended with features. Roles can be used to represent relationships between components. For the part–whole taxonomy a role hierarchy is useful.

While modeling components and their relationships can be expressed in a T-Box, more important for a configuration task are the inferences in the A-Box. Computing the subsumption graph of the components is perhaps an interesting task during knowledge acquisition, but really needed only for a correct behaviour of the A-Box inferences. Important inferences are the consistency test checking the configuration solution wrt. the components and relationships in the T-Box, and the two kinds of individual classification (strong and weak realisation). Strong realisation computes the set of components an individual *has to* classify to; weak realisation determines the components to which an individual *can possibly* classify to, if new information is available.

Unfortunately, a few modifications and extensions of the terminological system are needed: First, not all relationships between components can be expressed in the concept language. In a technical domain it must be possible to represent physical laws like $U = R * I$. Thus algebraic (in)equalities can be handled by an algebraic constraint solver. For correct inferences in the terminological system (especially in the T-Box), a constraint solver has to be integrated into the terminological part. Baader and Hanschke [BH91] describe a generic Concrete Domain interface for integrating external decision procedures. An algebraic constraint solver can be considered as such a decision procedure. But the most available terminological systems do neither provide such a generic interface nor algebraic constraints.

Second, the A–Box inferences needs an adaptation. Normally, the consistency test returns a boolean value. In order to do so, a terminological system based on a tableaux method implicitly generates a model. In the configuration task the calculated model is of interest to guide the next configuration steps. Therefore, the consistency test should also return the computed model, which in the end is the solution to the configuration problem specification.

Returning the computed model also requires an adaption of the decision procedures integrated via a concrete domain interface. The procedures should to be capable of returning their computed model. E.g. an algebraic constraint solver should return the values or ranges of its variables. This requirement restricts the choice of the appropriate decision procedure.

Furthermore, the individual classification can be improved. Description logics are based on an open world semantic. For this reason the individual realisation algorithms are very conservative in classifying. It assumes that a new concept description could be added to the T-Box anytime. During a configuration task it can be assumed that no

new component descriptions will be added to the T-Box. Through a (time-consuming) combination of the strong and weak realisation, individuals can be classified to more specific concepts than the pure strong realisation: if the most specific concept, to which an individual $i$ must be classified (strong realisation), is $C$ and if there exists only one subconcept $D$ of $C$, to which the individual $i$ can be classified (weak realisation), then assume that the individual $i$ is from concept $D$.

In our experience terminological systems extended by Concrete Domains offer a rich and expressive language to represent the domain knowledge (i.e. the components and relationships). An open problem is the formalisation of the specification. Normally, the specification is vague and uncertain. It is not clear, how such a specification can be transformed into a configuration solution.

Another problem are the inferences — especially the time consuming realisation. To speed up the realisation process the model generation should be shifted from a goal–driven to a data–driven inference. This means, that the system does not generate a hypothesis (can a individual $x$ be realised to concept $C$) and then tests this hypothesis but computes the classification direct by using the information in the A-Box (because of that the individual $x$ has the feature $f$ it has to classify to the concept $C$). A data-driven terminological system has to analyse and compile the T-Box before any A-Box inferences will be done.

## References


[BH91] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, DFKI, April 1991.

[G"u95a] A. G"unter. KONWERK – Ein modulares Konfigurierungswerkzeug. In F. Maurer and M.M. Richter, editors, *Expertensysteme 95*, Proceedings in Artificial Intelligence. "infix", St. Augustin, 1995.

[G"u95b] A. G"unter, editor. *Wissensbasiertes Konfigurieren – Ergebnisse aus dem Projekt PROKON.* Dr. Ekkehard Hundt, "infix"-Verlag, St. Augustin, 1995.

[Hei93] M. Heinrich. Ressourcenorientiertes Konfigurieren. *KI*, 7(1):11–15, 1993.

[HN90] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. Technical Report RR-90-04, DFKI, Kaiserslautern, Germany, April 1990.

[RC91] A. Strecker R. Cunis, A. G"unter, editor. *PLAKON – "Ubersicht "uber das System.* Informatik-Fachberichte. Springer, 1991.

# The Necessity of Using Semantic Models for Configuration*

**Olaf Wolter**
Institute of Material Handling and Logistics
Otto-von-Guericke-University Magdeburg
P.O. Box 4120, D-39016 Magdeburg
email: wolter@mb.uni-magdeburg.de

**Uwe Scholz**
Institute of Technical Information Systems
Otto-von-Guericke-University Magdeburg
P.O. Box 4120, D-39016 Magdeburg
email: uscholz@iti.cs.uni-magdeburg.de

## Abstract

The subject of this paper is the integration of configuration of material flow systems and material flow processes in the area of material flow plants. It will be shown the using of knowledge representation for the configuration process of matertial flow plants. Furthermore the necessity of an new quality in representation is illustrated. The described configuration process is very complex. In different to the conventional process the configuration is understand as a process that can be described by steps, loops, and versions, at which the unit of synthesis, analysis, and evaluation is of decisive meaning.

The principal objective is the integration of configuration of material flow plants in an extended model-based configuration concept. This can be represented by semantic models. Such an integrated concept offers the ability of systematically treating the necessary interrelations of the different models of material flow systems and material flow processes and achieves a more effective overall configuration process.

## 1 Introduction

The configuration of material flow plants is a problem of mechanical engineering in the special field of material flow and logistic. The material flow plant consist of the logistic system (technical material flow system and controlling system) and the logistic process (material flow process and controlling process). The following characterized configuration problem relates to the material flow system, the material flow process (physical process), and the controlling process (logical process). In the sequel the summary of the physical and the logical process is designated as material flow process.

* Parts of this work has been sponsored by the german country Sachsen-Anhalt under grant 1957A/025 and 1969A/025.

The configuration problem of this domain is a common development of the technical material flow system and of the material flow process. Configuration steps and configuration decisions influence the modelling of the material flow system during the modelling of the material flow process and vice versa. An integrated view on these two aspects is required in order to handle the interactions, to describe all interdependencies, and to fulfill the requirements of the configuration task. Possibilities of an integration of different aspects and models are described in [CBRR90, GKNO96, Lan94, Had95].

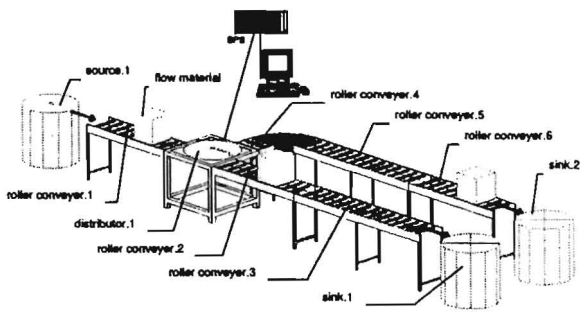


Figure 1: Example for a simple material flow plant

Current configuration concepts are insufficient in the way that they do not support or represent the procedures and the connections of the configuration of material flow plants in a comprehensive form. Up to now there is no possibility to record the connections between the material flow system, the material flow process, and the material flow control in a simple and integrated form. Another problem is the insufficient possibility to assess (sub-)solutions of the configuration. The described problem is to be led back, that the configuration concept is limit up in most kind to synthesis functions, where by part-whole relations find special regard [Gün93].

Further it is introduced an approach of using semantic models as an extension possibility and as a possibility for an integrated model description. predicated Unit-Relationship-Models (**pUR**-Models [DT94]) will be used for the representation of semantic models.

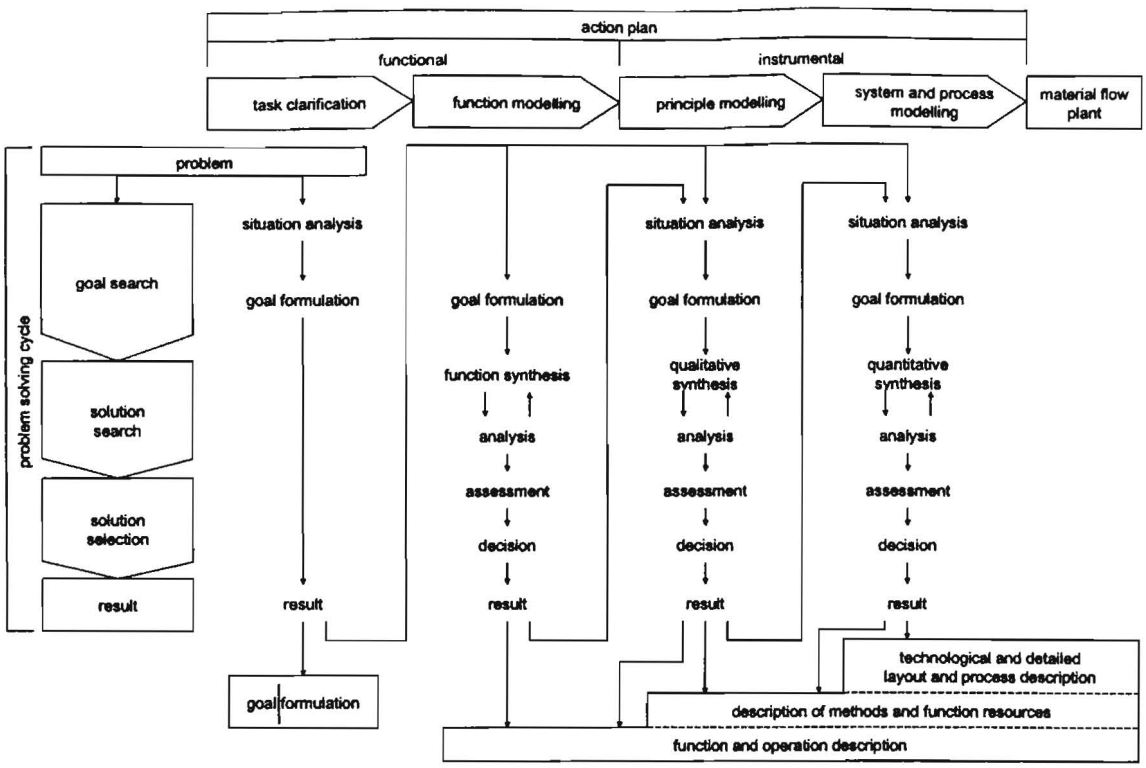


Figure 2: Action plan and problem solving cycle according to [Kra90]

A small parcel distribution plant is described as an example of the mentioned configuration problem. The plant distribute parcels according to a simple criterion (figure 1).

## 2 Configuration of Material Flow Plants

The configuration problem of material flow plants can be described as an evolutionary process of steps, iterations, variants, and versions with different degree of model abstraction of the material flow plant. Moreover the configuration process is generally separated into several phases. The phases of developing models can be divided into functional, principle, and system/process phases (see figure 2). A sequence of phases is defined by an action plan and by a problem solving cycle. A set of rules, control knowledge, and other methods of synthesis and analysis are associated with each phase. The steps of the process are models of different abstraction level which build up on each other by generalization/specialization.

During the solution development there is a permanent change of analyzing and working steps with synthesizing and assessing steps. At the same time the knowledge problem and the views on the models are changing. First of all it is looked for suitable and fitting solution elements. After this the elements must be tuned that they fulfill the demanded total function.

The used configuration techniques are heterogeneous and depended on the kind of knowledge problem, on the reached degree of the solution, and on the kind of the available information. The techniques contain methods for synthesis and analysis models and methods for a selection of model components with different detail. Further the management of model versions and variants must be possible. The methods of analysis describe simple determinate and complex stochastical calculation models in form of analytical formulas.

Existing model states can be analysed by statistical models of process simulation. Analytical methods support the establishing of

- suitable versions and variants,
- solution properties,
- conditions for the work of solutions, and
- consequences, which are shown through the solutions.

Furthermore assessments and views of versions and variants are realizable by state management and by interactive work. Because of that the complexity can be reduced.

The development of models is characterized by permanent changing of view and between abstraction levels and through a step by step increasing of information during the problem solving process. The solution development is influenced through the common configuration of the material flow process and of the material flow system.

## 3 pUR-Models as Formalism for Modelling

A possibility for representing semantic models are predicated Unit-Relationship-Models (**pUR**-Models [DT94]). Brodie [Bro84] defines semantic models as a scheme which describes the following properties of an universe of discourse (UoD):

- The static properties of an UoD are described by fixing objects, properties of objects, and relationships between objects.
- The dynamic properties of an UoD are described by assigning operations to objects, properties of these operations, and relationships between these operations.
- Integrity rules about objects (static conditions) and operations (dynamic conditions) are described by additional requirements and admissible states or state transitions of the considered mini world.

pUR-Models fulfill all these requirements. They represent objects, their attributes, belonging targets, and constraints on three levels: object level, attribute level, and assessment level.

Real objects of an UoD can be represented as object units. There exist units which describe one real world object or object set units which describe a set of equal real world objects. In pUR object units are expressed by boxes with small lines and object set units by boxes with big lines (see figure 3).

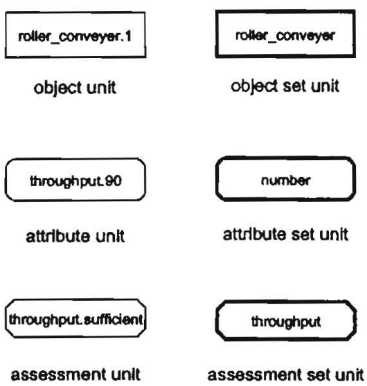


Figure 3: Object, attribute, and assessment units

The properties of objects can be described on the attribute level. For each object (object units or object set units) attribute units could be modelled. They are represented by boxes with rounded corners (figure 3). The assessment level is characterized through assessment units which are expressed by boxes with sloped corners (figure 3).

Furthermore pUR allows to model space and time relations (see figure 4). This relations describe relationships between object units, attribute units, or assessment units themselves, or between the elements on the three levels. For an example see figure 5. This pUR-Model describes a material flow

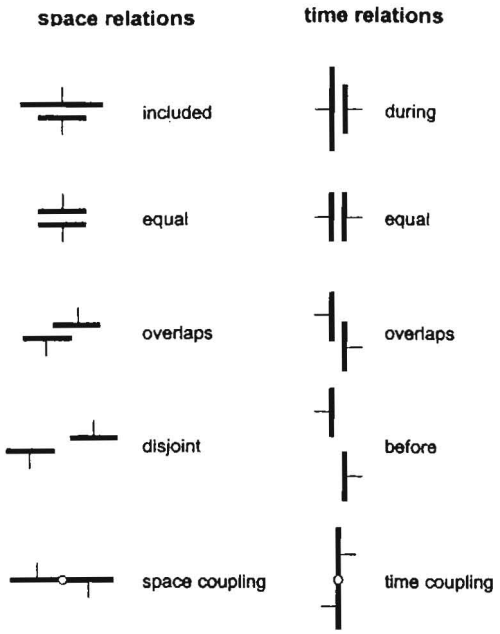


Figure 4: Space and time relations

plant. Every plant has an attribute *throughput*. If the *throughput* of the plant is greater than *900* than this plant fulfills the requirement for an sufficient *throughput*.

With pUR it's also possible to model combinations of time and space relations, e. g. an inclusion (the space relation *included*) which exits only for a determined time.

Through the summary of similar real world objects in object set units the abstraction concept of classification is representable. Furthermore the abstraction concepts of aggregation, association, and generalization/specialization could be modelled (figure 6). In an easy way constraints can be represented by description of an UoD on three levels (object, attribute, and assement level).

## 4 Model-based Configuration

The configuration of material flow plants is a problem of model based configuration, whereby models are of decisive meaning. In different to [Tan91] and [KBN94] the configuration is not only a problem of design rather of technical system planning. The configuration is understand as a modelling process of a material flow system and a material flow process. Basis of a systematic configuration is a corresponding configuration process with a structured and overlapping procedure. Thereby suboptima of solutions will be avoid and an approach at a main solution ensured.

The result of the configuration process is a detailed description of different models of material flow system and material flow process. In the sequel this description defines the requirements on the controlling system and the controlling program. Especially

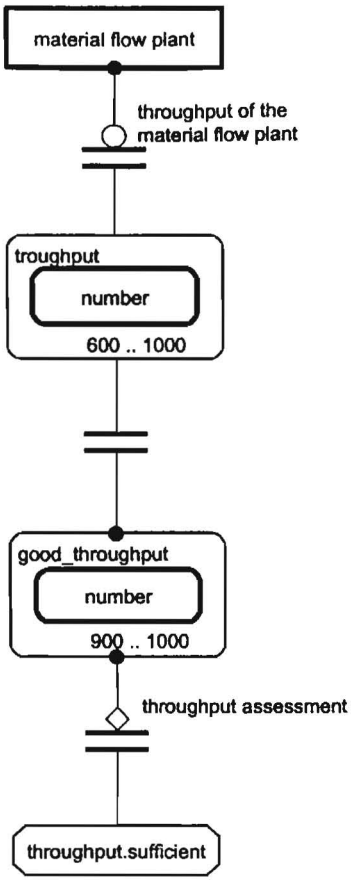


Figure 5: Connection between object, attribute, and assessment level

the complexity of the domain material flow plant requires an integrated model development. In particular the integrated model description will be used for the representation of solutions and for the check of the solution variants and versions in the different phases of abstraction and granularity. A similar approach of integrated modelling is shown in [GKNO96]. The goal of this approach is the integration of action planning and configuration in the area of planning of flexible assembly systems.

The advantages of an integrated model description are

- a better combination of different subsolutions (e.g. in the area of material flow systems and of material flow processes),
- the improvement of data consistency,
- a better transparency of models,
- an expressive common model description,
- the ensurence of flexible and suitable models with an uniform concept, and
- the development of problem adequate model views.

Today all known configuration concepts are restricted to single-stage configuration problems. In this context single-stage means that the solutions

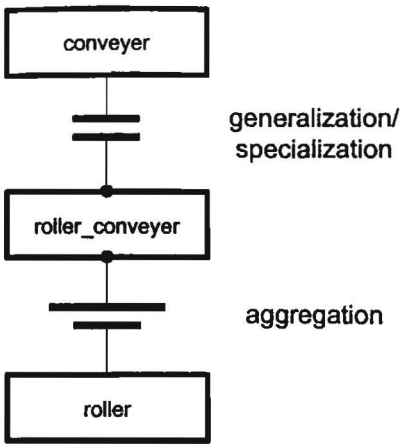


Figure 6: Example for aggregation and generalization/spezialization

of configurations are developed at one step of abstraction with existing configuration objects (see e.g. [CGS+90, SW91, Gün93, Som93, Lan94, GKNO96]. This does not agree with the requirements which appear during the configuration of material flow plants in the area of mechanical engineering. In this process the focus of interest is the development of integrated models of the material flow plant. The resulting models are the solutions to the mentioned configuration problem (see section 2).

Different models on different hierarchical abstraction will be developed in the configuration process. There are correlations between the models which depend on the special configuration task. During the problem solving process there is an information extension step by step which is supported by methods (see section 2). In the configuration process used methods are case based and ambiguous, therefore it is necessary to develop a variety of different solutions (proper configurations) and a flexible and variable assignment of different methods. The methods can not assigned to a special task of configuration (see also [Gün95]). The common development directions are present through the specific configuration task, the function order, and the configuration steps (figure 7). To simplify the further work it is useful to model the process for the primary working function, the functional, principle, geometrical, and process oriented aspects of the material flow plant, first. The complexity of the configuration process can be reduced by determining the primary functions (as described above) followed by extra functions and help functions. An orientation on primary functions can be described as a opportunistic procedure of configuration. Another possibility of reducing the complexity can be achieved by modularisation. This means there is a reducing on interfaces between subsystems and objects. Furthermore are used rough constraints in context with object interfaces. In this situation interfaces are predefined and can not change. This

kind is often used in the configuration process of material flow plants. Another aspect is the assumption that some subsystems and objects (components) and there properties are known in order to carry out a proper configuration.

In the first step of developing the configuration models a simplified view is taken on the formulation and on given relevant constraints and objectives. The formulation describes with the given initial state and the desired final state the material flow objects and their transformations. The functional/operational model is a specified formulation for the selection of technical elements (functional ressources). Functional/operational models will be represented mostly by graphs. The nodes represent functions/operations und the edges represent relations between nodes. The edges also characterize the flow process. On functional/operational models operate process chains and simple mathematical and analytical operations. Then suitable functions and fitting solution components (technical elements) are added to the principle model. This model allows to describe and to compare technical and technological information. In the third and last step this principle model must be tuned to fulfill the required plant function (system and process). Result of this step is the system and process model.

In this configuration process the focus of interest is the development of models, which are of different granularity and application and, of course, are problem adequate. The level of granularity can be characterized through models on different stages with various details. The granularity or abstraction level can change permanently during the process of configuration. The resulting models are the solutions to the mentioned configuration problem. The resulting models can be also of different granularity and are the proper configurations of the plant.

The basic operations of the configuration process which have to be realized by different configuration methods are

- synthesis:
  - *generation/elimination*
    The generation corresponds to the generating of a new component, a solution, and/or a partial solution. The inverse operator for the generation is the elimination, which is used for the deletion of a component.
  - *classification*
    Classification stands for the acquisition of a set of different objects with equal type. The objects are described by homogeneous properties and equal possibilities.
  - *aggregation/decomposition*
    With the bottom up approach aggregation objects/models will be assigned to another complex one. The decomposition corresponds with the top down approach and break an complex object/model into it's simple one's.
  - *generalization/specialization*
    Generalization assigns an object/model to a general class and specialization is the invers operation.
  - *association*
    Association describes relations between objects/models.
  - *combination*
    Combination is used to arrange subsolutions of variable complexity to a solution.
  - *substitution*
    With substitution an object/partial model will be replaced by another one.
  - *selection*
    Is used to select an object which will be used in the configuration process.
  - *transformation*
    Transformation is used for a stepwise refinement of partial solutions that means for representation of increasing the information. Transformation describes the elaboration process.
  - *specification*
    The specification is used for the refinement of a model. This refinement determines additional system attributes.
- analysis:
  - *evaluation (assessment, decision)*
    An assessment is undertaken through a judgment by potential (sub-)solutions thereby particular points of view and features. The choice contains the decision of a (sub-)solution from a final (sub-)solution set.
  - *interpretation*
    When interpreting connections are evaluated, whereby these lead to determined results.
  - *testing*
    The testing serves the consistency and correctness of structure and flow of the plant and the completeness of a configuration.
  - *situation analysis*
    The formulating of goals serves to complete, to structure, and to capture goals of the resulting material flow plant and subaspects as well as goals of the configuration procedure.
  - *simulation*
    The discrete-event simulation is used to determining and analyzing the models and controlling strategies of the material flow process.

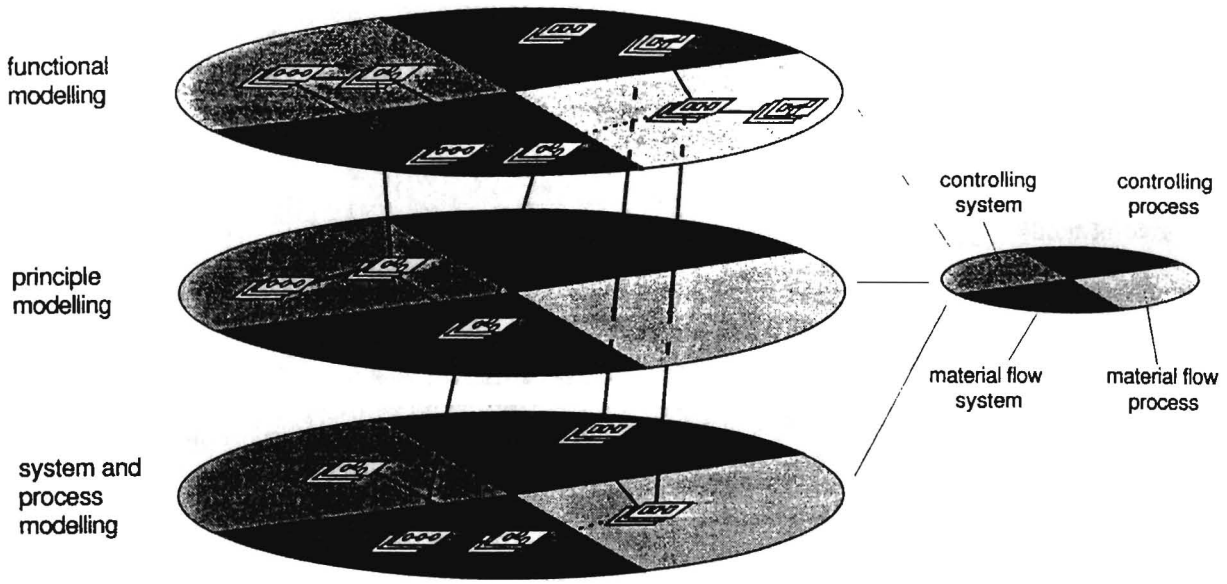


Figure 7: Modelling Space for configuration of material flow plants

The models can be described by versions and variants. Now the configuration of a model on one level can be isolated from the others, but also integrate with models on another level in a larger context. The evaluation of the system model can furthermore result in a replacement of one or more components. But, this replacement is only permitted when all information about possible predescessors, successors, and the function properties are preserved. To guarantee this there exist relations inside the model, between model modules and part models, and also between different models (figure 7).

The possibility to combine and translate models using mapping relations is an important criteria for their efficiency and effectiveness. Therefore the description of the models must be unique, interpretable, detailable, manipulable, and analysable. Using the appropriate abstraction concepts it can be guaranteed that the mapping relations are correct.

A multi-aspect-modelling, which is possible through space and time structures, allows to describe static and dynamic models in an integrated representation. With the integrated representation, connections can be expressed between resource (object), system, process, and flow material. Furthermore it is usable for the creation of jointly realizable structures by an uniform concept.

## 5 Knowledge Representation with pUR

Because of the necessity of permanent changes to the model view and the resulting complexity, emerges the demand on using an extended model based configuration. Such an extended configuration concept can be presented by semantic models. The knowledge representation is based on an object oriented hierarchical model concept called predicated Unit-Relationship-Model (pUR-Model) [DT94].

With pUR-Models it's possible to represent the whole configuration knowledge. pUR describes the knowledge in a hybrid form. Hybrid means in this context that different categories of knowledge and their combinations can also be represented through pUR-Models. In particular the solutions of the configuration process are also pUR-Models. The knowledge representation implies component and system models, the belonging discriptional, and controlling data as well as the configuration knowledge.

In [KB90] the different knowledge forms is structured. Refer to the contents of the knowledge it's distinguished between problem solving dependent and problem solving independent knowledge. Refer to the representation it's distinguished between formal and informal description. pUR-Models allow and support a formal and an informal knowledge representation in form of diagrams or pictures (see [DF96]). During the configuration process of material flow plants three different categories of knowledge must be handled: domain knowledge, problem solving knowledge, and dynamic model knowledge.

The domain knowledge describes the specific properties and the combinations of these properties in the area of material flow plants. The classification of material flow functions in a taxonomic representation, e.g. transport functions and distribution functions, the relations between material flow components, e.g. the connection between a component and a material flow subsystem, and a part-whole representation for components are described through the domain knowledge. Consequently domain knowledge is a part of problem solving independent knowledge.

Further the problem solving knowledge characterizes the procedure of the configuration process. It describes, how the configuration problem can be solved. This knowledge is also a part of the problem

solving independent knowledge.

The following characterized dynamic model knowledge belongs to the problem solving dependent knowledge. Therefore dynamic model knowledge contains the whole solutions and describes the history of the configuration process.

The goal of modelling is to configure and to define a material flow plant. The plant must fulfil the requirements and can be described through a conceptual model description. This description contains the complete form of the plant, which implies structural and functional completeness (see [SG91]). Structural completeness is fulfilled if all objects of the structure are defined and all values of their properties (attribute values and layout relationships) are determined. A description is functional complete if all requirements of the configuration task are fulfilled. Functional completeness expects the consistency of model description but not expects structural completeness.

The pUR paradigma offers a straigth-forward method to represent relationship concepts. Adding temporal relations, fundamental dynamic dependencies can be described. pUR-models are rated - predicated - by adding relations and data with which you can e.g. model constraints. Constraints are used for representing and evaluating of interdependencies between objects. They may refer to objects properties or to the existence of objects. A basis for using of constraints could be the 3-step constraint model of [Gün92]

- the claim of existence and non-existence constraints,
- to fade in and to fade out of constraints, and
- relaxation of constraints.

Furthermore the pUR-concept is applicable for recording the process of solution by model based elaboration and for an integrated description of different modelling aspects.

As mentioned earlier, the configuration solution are defined by models. During the configuration process the models are refined, perfected, and adapted. The transformation of a partial model into a new one by executing a configuration method is called *elaboration* [CGS+90]. The elaboration characterizes the different relationships between models on various development or abstraction levels and on one level (see figure 7). An elaboration tree can be built during the configuration process. The nodes of the tree characterize the models and the edges of the tree describe the model transitions or the working states. The resulting elaboration tree represents the complete history of the configuration process. "It stores the necessary information for 'intelligent' backtracking in case of conflicts, and for an explanation module" [Gün93]. So it's possible to document and to understand the problem solving process.

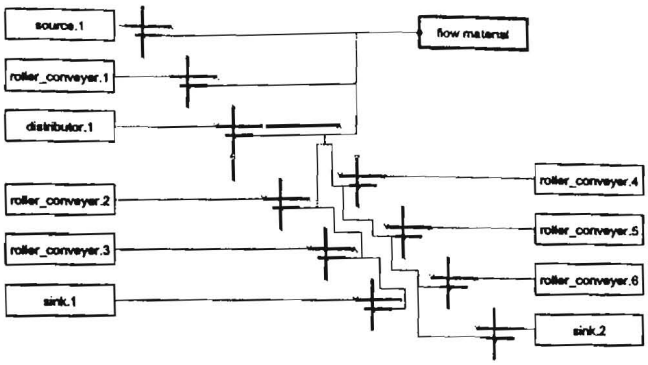


Figure 8: Structure and transport direction of a material flow plant

Another advantage of the model-based configuration with pUR is the practicability of an multi-aspect modelling (see also section 4). Multi-aspect modelling means that different facts can be represented by one pUR-Model. A different consideration of the integrated model description provides different aspects (views). Such views represent e.g. structure oriented and flow oriented properties. A view does emerge through fading out of all relevant properties, whereby an own subaspect model is defined. For example, a consideration of this structure oriented subaspect model can lead to a simulation model [DT96]. The model in figure 8 describes the solution of a configuration task (for the example see also figure 1). The structure of the material flow plant and the transport direction of a flow material can be derived from this one pUR-Model. So it's possible to describe many facts and relationships with few pUR-Models. The objects represent in case of system view components, technical functions, and ressources. In case of process view objects are general operations, work processes, and actions (see also section 4). The relations represent the structure relations between the objects and subsystems. They represent also the flow of material and the flow of information.

An additional example for the multi-aspect modelling is represented in figure 9. The controlling process of the material flow plants is described in this figure. This pUR-Model contains also information or aspects about the structure of the material flow plant which is shown in figure 1.

Finally is to remark that the models can be simplified by hiding of irrelevant properties or aspects.

## 6 Conclusion

In this paper an outline of a model-based configuration concept with pUR-Models was described. The meaning of the introduced concept for model-based configuration in the domain of material flow plants is the creation of an integrated description form. This integrated representation form allows to define problem adequate model descriptions. By using of inte-

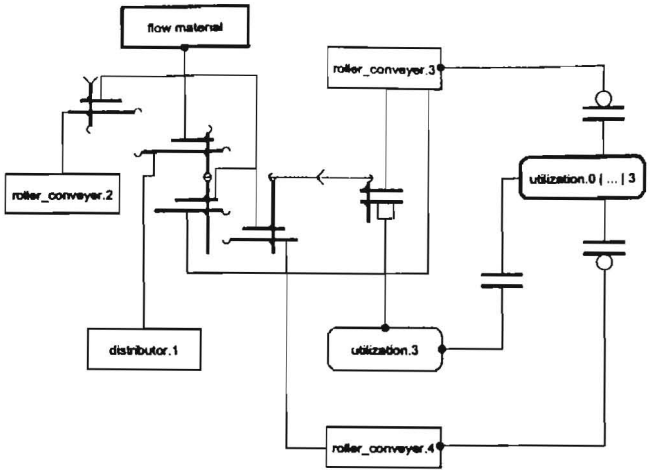


Figure 9: Controlling of a material flow plant

grated representation it is possible to realize a general computer aiding and a reduction of necessary model developments during the configuration process of material flow plants.

CAD-oriented configuration techniques in the domain of material flow plants (see [ZRW96]) were the starting point for the development of a model-based configuration concept with pUR-Models. A first approach for using pUR-Models was described in [RWS96]).

The introduced method was validated on a simple example (see figure 1). In future works the validation of complexer plants must be realized. Simultaneously a concept for mapping of a pUR-Model to an object-oriented scheme or to a relation scheme will be defined. The final goal is the prototypical development of a model-based configuration tool for material flow plants using pUR-Models.

## References

[Bro84] M. L. Brodie. On the Development of Data Models. In *On Conceptual Modeling*, pages 19–47. Topics in Information Systems, Springer-Verlag, New York, 1984.

[CBRR90] R. D. Coyne, M. Balachandran, M. A. Rosenman, and A. D. Radford. *Knowledge-based design systems*. Addison Wesley, Reading, Mass., 1990.

[CGS+90] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON - An Approach to Domain-Independent Construction. LKI-Report LKI-M-4/90, Universität Hamburg, Labor für Künstliche Intelligenz, 1990.

[DF96] V. Dobrowolny and H. Fischer. Nutzerweltorientierte Visualisierung konzeptueller Raum-Zeit-Modelle. In *Proc. Fachtagung Simulation und Animation für Planung, Bildung und Präsentation '96, Magdeburg, 29.2.-1.3.96*, volume 2, pages 213–220. ASIM, Arbeitsgemeinschaft Simulation in der Gesellschaft für Informatik, March 1996.

[DT94] V. Dobrowolny and Ch. Tietz. On Semantic Modeling of Real World Structures. In R. Wieringa and R. Feenstra, editors, *Working papers of the Int. Workshop on Information Systems - Correctness and Reusability, IS-CORE'94*, pages 321–331. Vrije Universiteit Amsterdam, RapportNr. IR-357, 1994.

[DT96] V. Dobrowolny and Ch. Tietz. Über Konzeptionelle Raum-Zeit-Modelle als Integrationsrahmen bei Simulation und Animation. In *Proc. Fachtagung Simulation und Animation für Planung, Bildung und Präsentation '96, Magdeburg*, pages 17–27. ASIM, Arbeitsgemeinschaft Simulation in der Gesellschaft für Informatik, March 1996.

[GKNO96] P. Ganghoff, A. Köhne, G. Näger, and U. Osmers. KNOSPE - Ein unterstützendes Planungssystem für die integrierte Montagesystemplanung. *Informatik-Forschung-Entw.*, 11(1):37–43, 1996.

[Gün92] A. Günter. *Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen*. Infix, Sankt Augustin, 1992.

[Gün93] A. Günter. Modelle beim Konfigurieren. In O. Herzog, T. Christaller, and D. Schütt, editors, *17. Fachtagung für Künstliche Intelligenz "Grundlagen und Anwendungen der Künstlichen Intelligenz"*, pages 169–176. Springer-Verlag, Berlin, 1993.

[Gün95] A. Günter. Architektur des domänenunabhängigen Konfigurierungswerkzeuges KONWERK. In A. Günter, editor, *Wissensbasiertes Konfigurieren: Ergebnisse aus dem Projekt PROKON*, pages 39–60. Infix, Sankt Augustin, 1995.

[Had95] S. Hader. Einsatz von Simulation beim Konfigurieren. In A. Günter, editor, *Wissensbasiertes Konfigurieren: Ergebnisse aus dem Projekt PROKON*, pages 229–235. Infix, Sankt Augustin, 1995.

[KB90] R. Koller and S. Berns. Strukturierung von Konstruktionswissen. *Konstruktion: Zeitschrift für Konstruktion und Entwicklung im Maschinen-, Apparate- und Gerätebau*, 42(3):85–90, 1990.

[KBN94] R. Klein, M. Buchheit, and W. Nutt. Configuration as Model Construction: The Construction Problem Solving Approach. In *Proceedings of the 4th International Conference on Artifical Intelligence in Design, Lausanne, Switzerland, August 1994*, pages 201–217. Kluwer Academic Press, London, 1994.

[Kra90] H. Krampe. *Transport-Umschlag-Lagerung.* Fachbuchverlag, Leipzig, 1990.

[Lan94] V. Lange. *Entwerfen von Fertigungsanlagen mit Modell- und Erfahrungsunterstützung.* Fortschritt-Berichte Reihe 2, Nr. 302. VDI Verlag, Düsseldorf, 1994.

[RWS96] K. Richter, O. Wolter, and U. Scholz. Konfigurierung materialflußtechnischer Systeme mit Hilfe von Raum-Zeit-Beschreibungen. In J. Sauer, A. Günter, and J. Hertzberg, editors, *Planen und Konfigurieren 96: Beiträge zum 10. Workshop "Planen und Konfigurieren" (PuK-96) 15.–17. April 1996, Bonn*, pages 239–242, April 1996.

[SG91] J. Schmidt and P. Ganghoff. Wissensbasierte Planung der Aufbau- und Ablaufstruktur von Montagesystemen. *VDI-Zeitung*, 133(11):85–92, 1991.

[Som93] C. Sommer. *MoKon - ein Ansatz zur wissensbasierten Konfiguration von Variantenerzeugnissen.* Infix, Sankt Augustin, 1993.

[SW91] B. Stein and J. Weiner. MOKON - Eine modellbasierte Entwicklungsplattform zur Konfigurierung technischer Anlagen. In A. Günter and R. Cunis, editors, *PuK-91, Beitäge zum 5. Workshop "Planen und Konfigurieren", 22.-23.4, Hamburg, LKI-M-1/91*, pages 100–106, 1991.

[Tan91] W. Tank. Modell-basiertes vs. assoziatives Konstruieren. In A. Günter and R. Cunis, editors, *PuK-91, Beitäge zum 5. Workshop "Planen und Konfigurieren", 22.-23.4, Hamburg, LKI-M-1/91*, pages 214–215, 1991.

[ZRW96] D. Ziems, K. Richter, and O. Wolter. CAD-Modelle mit Planungs-Know-how zur Konfigurierung von Materialflußsystemen. In D. Ruhland, editor, *CAD '96, Verteilte und intelligente CAD-Systeme: Tagungsband; Kaiserslautern, 7./8. März 1996*, pages 135–149, 1996.

# Software Configuration with Feature Logic

**Andreas Zeller**
Technische Universität Braunschweig
Abteilung Softwaretechnologie
D-38092 Braunschweig
zeller@acm.org

## Abstract

*Software configuration management (SCM) is the discipline for controlling the evolution of software systems. The central problems of SCM are closely related to central artificial intelligence (AI) topics, such as* knowledge representation *(how do we represent the features of versions and components, and how does this knowledge involve in time?),* configuration *(how do we compose a consistent configuration from components, and how do we express constraints?), and* planning *(how do we construct a software product from a source configuration, and what are the features of this product?).*

*Although the research communities of both SCM and AI work on configuration topics, the knowledge about the mutual problems and methods is still small. We show how* feature logic, *a description logic with boolean operations, can be used to represent both knowledge about versions and components, as well as to infer the consistency of possible configurations and thus solve configuration problems in SCM. This interplay of knowledge representation and configuration techniques shows immediate beneficial consequences in SCM, such as the integration and unification of SCM versioning concepts. Moreover, SCM may turn out as a playground for testing and validating new AI methods in practice.*

## 1 Introduction

Software Configuration Management (SCM) is the discipline for controlling the evolution of software systems. While early SCM tools were confined to basic tasks such as revision control (e.g. RCS, SCCS), variant control (e.g. CPP) or system construction (e.g. MAKE), today's SCM systems provide integrated support for tasks such as identification and retrieval of components and configurations, revision and variant control, or consistency checking.

One of the benefits of SCM is that it can be easily automated, since its items are already under computer control; all properties of configuration items can be immediately observed and deduced. It is thus surprising that the artificial intelligence (AI) configuration community has not yet discovered SCM as a fruitful application domain for configuration problems, just as it is surprising that the SCM literature is essentially devoid of formal approaches, let alone formal approaches to configuration problems. A possible reason for this gap is that SCM touches a wide range of divergent subjects, from component identification and configuration problems over software process modeling to general management issues, each with its distinct research community.

Our research group in Braunschweig aims to exploit recent fundamental achievements for the benefit of practice, notably the application of AI techniques in software engineering. We found the core techniques of SCM closely related to well-established AI research topics:

**Knowledge representation.** How do we express knowledge about a software component, such that we can identify and retrieve components and configurations? How does this knowledge evolve in time, and how does it propagate to configurations?

**Configuration.** How can we determine the consistency of a configuration of software components? How do we express configuration constraints, and how are these related to component identification?

**Planning.** How is software constructed and delivered? How do the properties of source components determine the properties of derived components?

Starting with these relationships, we decided to examine the current state of practical SCM, to identify SCM problems, and to devise possible solutions from AI research.

## 2 The Versioning Problem

We begin with a short introduction to SCM. In the SCM domain, we have the problem of maintaining components in several *versions*. Versions are created in several *versioning dimensions*, depending on the intentions of their creator. SCM research distinguishes three versioning dimensions.

**Historical versioning.** Versions that are created to *supersede* a specific version, e.g. for maintenance purposes, are called *revisions*. When a

new revision is created, evolution of the original version is phased out in favor of the new revision. In practice, a revision of a component is usually created by modifying a copy of the most recent revision. The old revisions are permanently stored for maintenance and documenting purposes.

**Logical versioning.** In contrast to revisions, a variant is created as an *alternative* to a specific version. *Permanent variants* are created when the product is adapted to different environments. Variance can again arise in several dimensions, including varying user requirements and varying system platforms, but also variants for testing and debugging.

**Cooperative versioning.** A *temporary variant* is a variant that will later be integrated (or merged) with another variant. Temporary variants are required, for example, to change an old revision while the new revision is already under development, or to realize cooperative work through parallel development threads.

Of these three versioning dimensions, only logical versioning is visible in the final product as different permanent variants. Since maintaining several product variants is more expensive than maintaining one single product, it is a general software engineering issue to keep the number of variants as small as possible. This is achieved through well-known software engineering principles like abstraction, parameterization, generalization, and localization.

One must be aware that only logical versioning can be planned in advance. The creation of revisions and temporary variants may be necessary at any time during the development process, resulting in a huge set of possible configurations, which must be identified and tested.

The problem becomes worse when individual *changes* are considered rather than versions, since each combination of changes results in a different configuration. While software engineering principles help to confine the impact of changes behind a certain abstraction, change combinations nonetheless must be identified and evaluated.

Furthermore, there is a transition from static configuration at compile-time to dynamic configuration at run-time, which results in new configuration problems in the final software product. Already in a small software system with but a few thousand components, SCM can rapidly turn into a nightmare unless intelligent tools help to manage this mess.

## 3 SCM Versioning Issues

The maintenance of several versions can be dramatically simplified by using an automated SCM system. Even the easiest SCM system provides some basic support for the following SCM tasks.

**Identification.** Each component in a software product must be uniquely identifiable and and accessible in some form. Identification schemes, as found in SCM systems, range from simple *revision numbering* (revision 1.0, 1.1, 1.2 ... ) up to *faceted classification* using attribute/value-pairs (*state* = *experimental* $\wedge$ *version* = 1.0). The mechanisms to *select* component versions include creation dates, revision numbers, boolean attribute/value combinations, as well as high-level database query languages.

**Composition.** As specific versions of components are composed to configurations, these configurations must be identifiable and accessible as well. Simple SCM tools allow to tag individual component versions with a *label* identifying the configuration. Selecting a configuration is done through a label selecting the appropriate component versions. For instance, the label *REV_1_0* may denote a configuration including revision 1.4 of component $A$ and revision 1.7 of component $B$. More advanced SCM systems allow versioning of configurations just like versioning of components.

**Consistency.** Advanced SCM systems allow users to specify *configuration rules* constraining possible configurations. Such configuration rules may denote that certain changes imply or exclude each other, that changes may be applied to certain variants only, or that only specific variants and revisions result in a well-tested configuration.

Modern SCM systems provide adequate mechanisms to identify and compose software component versions in a software product. The most important SCM issue in this area is *generalizing*—that is, to find a common subset of versioning techniques to improve the interoperability of SCM systems.

Consistency control, however, is still a challenge—especially because most consistency problems arise through the integration of versioning dimensions, which is still at a very early stage.

As a simple example of the consistency problems as found in SCM, consider figure 1, illustrating the dependencies between changes and revisions. Each revision $R_i$ is the product of some changes $\delta_j$ applied to a baseline revision $R_0$. In the diagram, each set $\Delta_j$ contains the revisions the change $\delta_j$ has been applied to. Hence, revision $R_5$ is the product of the changes $\delta_1$ to $\delta_5$ applied to $R_0$, but the change $\delta_6$ has not been applied.

We see that the changes are not orthogonal to each other. For instance, the change $\delta_2$ implies that $\delta_1$ be applied first, since $\Delta_2$ is a subset of $\Delta_1$. Likewise, the changes $\delta_2$ and $\delta_6$ exclude each other, since the sets $\Delta_2$ and $\Delta_6$ are disjoint. The SCM system must ensure that these constraints are satisfied. The problem becomes worse if not only six, but several thou-

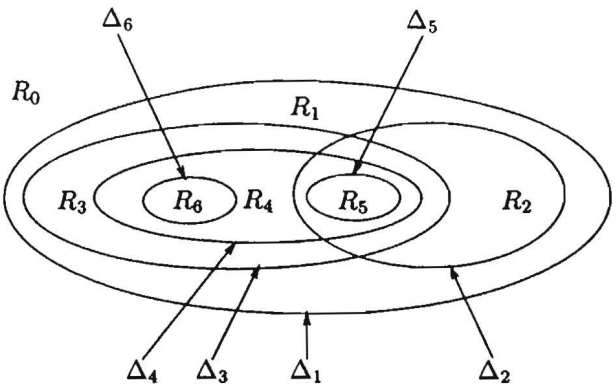


Figure 1: Changes and revisions

sand changes are involved, some of them restricted to a particular variant or configuration, which in turn may impose other constraints.

Finally, constraints are subject to changes as well. For example, a change may be initially visible in a user's temporary variant only. This means that there is a constraint associated with this change that implies a specific user. Later, the same change may be incorporated into the delivered product. This means that the original constraint must be replaced by a new constraint implying a specific configuration.

Just like constraints, the system hardware, process models, or even SCM policies may evolve in ways that cannot be foreseen, and this evolution may be subject to constraints again. This is the true challenge of SCM: Changes and constraints pervade every single item considered, from components to configurations to the processes themselves.

## 4 A SCM Foundation

In our quest for a SCM foundation, we searched for a formalism that allows us to capture and unify the core techniques of SCM, supporting evolution and versioning on every SCM level. As stated in the introduction, we have examined logical formalisms supporting knowledge representation as well as configuration and planning in the SCM domain. We identified two possible foundations:

**Description logics.** In the SCM area, it is common to identify component versions by faceted classification, using a set of *attribute/value* pairs. For us, description logics turned out to be a first-order choice for identifying components and expressing knowledge about their possible use in a configuration.

**First-order logic.** Another technique frequently found in SCM systems is to use first-order logic expressions to select and identify configurations, as well as to express consistency constraints; only configurations satisfying the constraints are consistent. Supporting first-order logic was essential for us in order to capture these selection schemes.

Fortunately, we found a formalism which captures both description and first-order logic: *Feature logic*, as defined by Smolka [1], is a well-founded description logic that, in contrast to most description logics, includes quantification, disjunction, and negation over attribution terms, forming a full boolean algebra. Feature logic gives us one single formalism for both knowledge representation and configuration problems.

## 5 Configuration with Feature Logic

Using feature logic, all components are identified by a *feature term*, describing the component through *features*—that is, a pair *feature-name*: *feature-value*. Typically, components and component versions are identified by a conjunction ("$\sqcap$" or "[...]") of their features. For instance, we can distinguish two versions of a *printer* component with respect to their respective printer language:

$$printer_1 = [print\text{-}language: postscript]$$
$$printer_2 = [print\text{-}language: text] \ .$$

Feature logic now allows us to express feature negations ("$\sim$"), expressing undefined features, or disjunctions ("$\sqcup$" or "{...}"), expressing alternatives. In figure 2, we have summarized the syntax of feature terms.

The ability to express alternatives is essential in SCM, since one frequently must handle all versions of a component as a single item or component family. Using alternatives, we can identify the component family *printer* containing both $printer_1$ and $printer_2$ as

$$\begin{aligned} printer &= printer_1 \sqcup printer_2 \\ &= [print\text{-}language: \{postscript, text\}] \end{aligned}$$

and thus immediately determine the features of the *printer* component family.

Negations can be used to identify component *revisions* by distinguishing whether a change has been applied ("$\delta_i: \top$"), or not ("$\sim\delta_i: \top$"). For instance, consider a *screen* component, where the $\delta_1$ change introduces a new revision which can handle display PostScript:

$$screen_1 = [\sim\delta_1: \top, screen\text{-}language: bitmap]$$
$$screen_2 = [\delta_1: \top, screen\text{-}language: bitmap]$$
$$screen_3 = [\delta_1: \top, screen\text{-}language: postscript]$$

This change determines the *screen* component family as

$$\begin{aligned} screen &= screen_1 \sqcup screen_2 \sqcup screen_3 \\ &= (screen\text{-}language: \{postscript, bitmap\}) \\ &\quad \sqcap (screen\text{-}language: postscript] \rightarrow [\delta_1: \top]) \end{aligned}$$

stating in an implication ("$\rightarrow$" with $A \rightarrow B \equiv \sim A \sqcup B$) that selecting the [*screen-language*: *postscript*] version implies that the $\delta_1$ change has been applied.

| Notation | Name | Interpretation |
|---|---|---|
| $a$ | Literal | |
| $V$ | Variable | |
| $\top$ (also $[]$) | Top | Ignorance |
| $\bot$ (also $\{\}$) | Bottom | Inconsistency |
| $f{:}\,S$ | Selection | The value of $f$ is $S$ |
| $f{:}\,\top$ | Existence | $f$ is defined |
| $f{\uparrow}$ | Divergence | $f$ is undefined |
| $f \downarrow g$ | Agreement | $f$ and $g$ have the same value |
| $f \uparrow g$ | Disagreement | $f$ and $g$ have different values |
| $S \sqcap T$ (also $[S,T]$) | Intersection | Both $S$ and $T$ hold |
| $S \sqcup T$ (also $\{S,T\}$) | Union | $S$ or $T$ holds |
| $\sim S$ | Complement | $S$ does not hold |
| $S \rightarrow T$ | Implication | If $S$ holds, then $T$ holds |
| $S \sqsupseteq T$ | Subsumption | $S$ subsumes $T$; $T$ implies $S$ |

Figure 2: The syntax of feature terms

When composing configurations, they inherit the features from their components; the feature values are unified. This allows us to use feature terms as configuration constraints. As an example, take a component *dumper* which copies data from the screen to a printer. Both formats must be identical, as expressed through the common variable $D$:

$$\mathit{dumper} = [\mathit{screen\text{-}language}{:}\,D, \mathit{print\text{-}language}{:}\,D]$$

Now consider the configuration *subsystem* composed from the three components *dumper*, *screen*, and *printer*. Its features are determined as:

$$\begin{aligned}\mathit{subsystem} &= \mathit{printer} \sqcap \mathit{screen} \sqcap \mathit{dumper} \\ &= [\delta_1{:}\,\top, \mathit{print\text{-}language}{:}\,\mathit{postscript}, \\ &\quad \mathit{screen\text{-}language}{:}\,\mathit{postscript}]\ ,\end{aligned}$$

where all other configurations have been excluded by the features of *dumper* and *screen*. We see how features can represent knowledge about the component as well as constraints about its usage in a specific configuration.

## 6 Constraints as Features

Our primary aim for using feature logic is to use one single identification formalism in all versioning dimensions. The resulting *version set model* uses feature terms to identify arbitrary versions: revisions are identified by changes applied or not applied ($\delta_{47}{:}\,\top$), temporary variants are identified by specific users ($\mathit{user}{:}\,\mathit{lisa}$) or teams ($\mathit{team}{:}\,\mathit{microserfs}$), and permanent variants are identified by general feature terms ($\mathit{os}{:}\,\{\mathit{windows95}, \mathit{windows\text{-}nt}\}$). Feature terms are used for identifying and selecting arbitrary subsets in all versioning dimensions. Many examples illustrating the usage of this model are given in [2, 3, 4], and especially in [5].

Besides this SCM-specific integration of versioning concepts, feature logic has another substantial advantage: the representation of configuration constraints as component features (or version features). Among the most important benefits are:

**Distributed constraints.** Rather than having one central instance defining the possible configurations, each component and each version defines the constraints related to its usage. If the component is excluded from a configuration, its usage constraints are excluded as well. Since the constraints are evaluated incrementally while the configuration is composed, developers can detect inconsistencies already at the subsystem or even at the component level, which avoids propagating inconsistencies across subsystem or workspace boundaries.

**Constraint evolution.** Since configuration constraints are associated with components, they are versioned like the components themselves. Upon creating a new component version, developers can choose whether to inherit the features and constraints of the base version, or to assign new features and constraints. Consequently, versions of the configuration space and component versions determine each other.

**One single representation.** Finally, through the exclusive use of feature logic, there is no hierarchy between objects to be configured and the configuration rules themselves. Configuration rules may imply other features, and vice versa—constraints may be subject to versioning just as specific versions may imply specific constraints. For instance, we may select configurations by stating their constraints ("Show all configurations where the $\delta_{43}$ change implies the UNIX operating system").

The drawback of this flexibility is *computation complexity*. Feature unification, the primary method to determine consistency of feature terms, is $\mathcal{NP}$-complete, which results in exponential worst-case

complexity for all SCM operations. However, "classical" SCM operations—that is, the ones that are used in today's SCM systems—impose no special complexity problems when modeled using feature logic. We found that the majority of SCM problems either imposes very *tight* or very *loose* configuration constraints. Tight constraints, as in change dependencies, are easy to handle since they reduce the configuration space. Loose constraints, as in orthogonal variance dimensions, are also easy to handle since their satisfaction is easily computed.

However, these simplifications apply to today's SCM systems only. Future SCM systems supporting arbitrary versioning dimensions and arbitrary configuration constraints will hurt this complexity barrier. This is a challenge for both SCM and AI researchers. In the SCM domain, we must find out how far new SCM tasks and procedures can go without being endangered by the underlying complexity. And in the AI domain, we must devise and exchange methods to handle huge sets of intertwined configuration constraints and alternatives.

## 7 Conclusion

Using feature logic for both knowledge representation and configuration constraints turned out to be a valuable contribution to the SCM area. Among the preliminary results are:

- The efficient integration of the four main SCM versioning models, resulting in a general SCM foundation [2];
- A unified versioning model for SCM, increasing flexibility in the software process [5];
- The development of FFS, a virtual file system which allows transparent access to arbitrary file and directory versions just by stating attribute constraints [3];
- The implementation of ICE, an inference-based SCM system supporting deductive software construction as well as interactive exploration of the configuration space on top of the FFS [4].

As a conclusion, the application of a theoretical AI foundation to a practical software engineering problem resulted in a success story. The interplay of knowledge representation and configuration techniques raised a number of potential complexity problems, but also showed immediate beneficial consequences in SCM. In general, the SCM domain may turn out as a valuable playground for testing and validating new AI methods in practice—hopefully somewhat closing the gap between configuration practice and configuration theory.

ICE is part of the inference-based software development environment NORA[1]. NORA aims at utilizing inference technology in software tools. ICE software for UNIX systems and related technical reports can be accessed through the ICE WWW page, http://www.cs.tu-bs.de/softech/ice/ as well as directly via anonymous FTP from ftp://ftp.ips.cs.tu-bs.de/pub/local/softech/ice/.

[1]NORA is a figure in Henrik Ibsen's play "A Dollhouse". Hence, NORA is NO Real Acronym.

**Acknowledgments.** This work was funded by the Deutsche Forschungsgemeinschaft, grants Sn11/1-2 and Sn11/2-2.

## References

[1] SMOLKA, G. Feature-constrained logics for unification grammars. *Journal of Logic Programming 12* (1992), 51–87.

[2] ZELLER, A. A unified version model for configuration management. In *Proc. 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Washington, DC, Oct. 1995), G. Kaiser, Ed., vol. 20 (4) of *ACM Software Engineering Notes*, ACM Press, pp. 151–160.

[3] ZELLER, A. Smooth operations with square operators—The version set model in ICE. In *Proc. 6th International Workshop on Software Configuration Management* (Berlin, Germany, Mar. 1996), I. Sommerville, Ed., Lecture Notes in Computer Science, Springer-Verlag. To appear.

[4] ZELLER, A., AND SNELTING, G. Handling version sets through feature logic. In *Proc. 5th European Software Engineering Conference* (Sitges, Spain, Sept. 1995), W. Schäfer and P. Botella, Eds., vol. 989 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 191–204.

[5] ZELLER, A., AND SNELTING, G. Unified versioning through feature logic. Computer Science Report 96-01, Technical University of Braunschweig, Germany, Mar. 1996. Invited for submission to ACM Transactions on Software Engineering and Methodology.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210

e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

## Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

## DFKI Publications

*The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.*
*The reports are distributed free of charge except where otherwise noted.*

---

# DFKI Research Reports

## 1996

**RR-96-05**
*Stephan Busemann*
Best-First Surface Realization
11 pages

**RR-96-03**
*Günter Neumann*
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

**RR-96-02**
*E.André, J. Müller , T.Rist:*
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

## 1995

**RR-95-20**
*Hans-Ulrich Krieger*
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

**RR-95-19**
*Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger*
Distributed Parsing With HPSG Grammar
20 pages

**RR-95-18**
*Hans-Ulrich Krieger, Ulrich Schäfer*
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

**RR-95-17**
*Hans-Ulrich Krieger*
Classification and Representation of Types in TDL
17 pages

**RR-95-16**
*Martin Müller, Tobias Van Roy*
Title not set
0 pages

**Note:** The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

**RR-95-15**
*Joachim Niehren, Tobias Van Roy*
Title not set
0 pages

**Note:** The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

**RR-95-14**
*Joachim Niehren*
Functional Computation as Concurrent Computation
50 pages

**RR-95-13**
*Werner Stephan, Susanne Biundo*
Deduction-based Refinement Planning
14 pages

**RR-95-12**
*Walter Hower, Winfried H. Graf*
Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey
33 pages

**RR-95-11**
*Anne Kilger, Wolgang Finkler*
Incremental Generation for Real-Time Applications
47 pages

**RR-95-10**
*Gert Smolka*
The Oz Programming Model
23 pages

**RR-95-09**
*M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf*
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

**RR-95-08**
*Michael Mehl, Ralf Scheidhauer, Christian Schulte*
An Abstract Machine for Oz
23 pages

**RR-95-07**
*Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt*
The Complexity of Concept Languages
57 pages

**RR-95-06**
*Bernd Kiefer, Thomas Fettig*
FEGRAMED
An interactive Graphics Editor for Feature Structures
37 pages

**RR-95-05**
*Rolf Backofen, James Rogers, K. Vijay-Shanker*
A First-Order Axiomatization of the Theory of Finite Trees
35 pages

**RR-95-04**
*M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,*
*M. Wójcik*
Task Acquisition with a Description Logic Reasoner
17 pages

**RR-95-03**
*Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,*
*Thomas Kieninger, Norbert Kuhn*
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

**RR-95-02**
*Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,*
*Achim Weigel*
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

**RR-95-01**
*Klaus Fischer, Jörg P. Müller, Markus Pischel*
Cooperative Transportation Scheduling
an application Domain for DAI
31 pages

## 1994

**RR-94-39**
*Hans-Ulrich Krieger*
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

**RR-94-38**
*Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,*
*Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer,*
*Hans-Ulrich Krieger,*
*Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.*
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

**RR-94-37**
*Hans-Ulrich Krieger, Ulrich Schäfer*
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

**RR-94-36**
*Manfred Meyer*
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

**RR-94-35**
*Rolf Backofen*
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

**RR-94-34**
*Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,*
*Günter Neumann, Hans Uszkoreit*
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

**RR-94-33**
*Franz Baader, Armin Laux*
Terminological Logics with Modal Operators
29 pages

**RR-94-31**
*Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann*
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

**RR-94-23**
*Gert Smolka*
The Definition of Kernel Oz
53 pages

**RR-94-20**
*Christian Schulte, Gert Smolka, Jörg Würtz*
Encapsulated Search and Constraint Programming in Oz
21 pages

**RR-94-19**
*Rainer Hoch*
Using IR Techniques for Text Classification in Document Analysis
16 pages

**RR-94-18**
*Rolf Backofen, Ralf Treinen*
How to Win a Game with Features
18 pages

**RR-94-17**
*Georg Struth*
Philosophical Logics—A Survey and a Bibliography
58 pages

**RR-94-16**
*Gert Smolka*
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

**RR-94-15**
*Winfried H. Graf, Stefan Neurohr*
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

**RR-94-14**
*Harold Boley, Ulrich Buhrmann, Christof Kremer*
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

**RR-94-13**
*Jana Koehler*
Planning from Second Principles—A Logic-based Approach
49 pages

**RR-94-12**
*Hubert Comon, Ralf Treinen*
Ordering Constraints on Trees
34 pages

**RR-94-11**
*Knut Hinkelmann*
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

**RR-94-10**
*Knut Hinkelmann, Helge Hintze*
Computing Cost Estimates for Proof Strategies
22 pages

**RR-94-08**
*Otto Kühn, Björn Höfling*
Conserving Corporate Knowledge for Crankshaft Design
17 pages

**RR-94-07**
*Harold Boley*
Finite Domains and Exclusions as First-Class Citizens
25 pages

**RR-94-06**
*Dietmar Dengler*
An Adaptive Deductive Planning System
17 pages

**RR-94-05**
*Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.*
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

**RR-94-03**
*Gert Smolka*
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

**RR-94-02**
*Elisabeth André, Thomas Rist*
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

**RR-94-01**
*Elisabeth André, Thomas Rist*
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

# DFKI Technical Memos

## 1996

**TM-96-01**
*Gerd Kamp, Holger Wache*
CTL — a description Logic with expressive concrete domains
19 pages

## 1995

**TM-95-04**
*Klaus Schmid*
Creative Problem Solving
and
Automated Discovery
— An Analysis of Psychological and AI Research —
152 pages

**TM-95-03**
*Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache,*
*Franz Schmalhofer*
An Environment for Exploring and Validating Declarative Knowledge
11 pages

**TM-95-02**
*Michael Sintek*
FLIP: Functional-plus-Logic Programming
on an Integrated Platform
106 pages

**TM-95-01**
*Martin Buchheit, Rüdiger Klein, Werner Nutt*
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

## 1994

**TM-94-04**
*Cornelia Fischer*
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

**TM-94-03**
*Victoria Hall*
Uncertainty-Valued Horn Clauses
31 pages

**TM-94-02**
*Rainer Bleisinger, Berthold Kröll*
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

**TM-94-01**
*Rainer Bleisinger, Klaus-Peter Gores*
Text Skimming as a Part in Paper Document Understanding
14 pages

# DFKI Documents

## 1996

**D-96-05**
*Martin Schaaf*
Ein Framework zur Erstellung verteilter Anwendungen
94 pages

**D-96-03**
*Winfried Tautges*
Der DESIGN-ANALYZER - Decision Support im Designprozess
75 Seiten

## 1995

**D-95-12**
*F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)*
Working Notes of the KI'95 Workshop:
KRDB-95 - Reasoning about Structured Objects:
Knowledge Representation Meets Databases
61 pages

**D-95-11**
*Stephan Busemann, Iris Merget*
Eine Untersuchung kommerzieller Terminverwaltungssoftware im Hinblick auf die Kopplung mit natürlichsprachlichen Systemen
32 Seiten

**D-95-10**
*Volker Ehresmann*
Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON
108 Seiten

**D-95-09**
*Antonio Krüger*
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

**D-95-08**
*Technical Staff*
DFKI Jahresbericht 1994
63 Seiten

Note: This document is no longer available in printed form.

**D-94-02**
*Markus Steffens*
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

**D-94-01**
*Josua Boon (Ed.)*
DFKI-Publications: The First Four Years 1990 - 1993
75 pages

**D-95-07**
*Ottmar Lutzy*
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

**D-95-06**
*Markus Steffens, Ansgar Bernardi*
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

**D-95-05**
*Georg Schneider*
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

**D-95-04**
*Victoria Hall*
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

**D-95-03**
*Christoph Endres, Lars Klein, Markus Meyer*
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

**D-95-02**
*Andreas Butz*
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

**D-95-01**
*Susanne Biundo, Wolfgang Tank (Hrsg.)*
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

**Note:** This document is available for a nominal charge of 25 DM (or 15 US-$).

## 1994

**D-94-15**
*Stephan Oepen*
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

**D-94-14**
*Hans-Ulrich Krieger, Ulrich Schäfer*
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

**D-94-12**
*Arthur Sehn, Serge Autexier (Hrsg.)*
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

**D-94-11**
*F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)*
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

**Note:** This document is no longer available in printed form.

**D-94-10**
*F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)*
Working Notes of the 1994 International Workshop on Description Logics
118 pages

**Note:** This document is available for a nominal charge of 25 DM (or 15 US-$).

**D-94-09**
*Technical Staff*
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

**D-94-08**
*Harald Feibel*
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

**D-94-07**
*Claudia Wenzel, Rainer Hoch*
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

**D-94-06**
*Ulrich Buhrmann*
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

**D-94-04**
*Franz Schmalhofer, Ludger van Elst*
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

**D-94-03**
*Franz Schmalhofer*
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

**Note:** This document is no longer available in printed form.

# Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96

Franz Baader, Hans-Jürgen Bürckert,
Andreas Günter, Werner Nutt (Eds.)

D-96-04
Document