A Layer Architecture for the Integration of Rules, Inheritance, and Constraints

Andreas Abecker and Holger Wache
DFKI
Postfach 2080, 67608 Kaiserslautern (Germany)
E-mail: {aabecker|wache}@dfki.uni-kl.de

Abstract

We informally introduce **TaxLog**, a close integration of logic programming and terminological reasoning.

Terminological systems are handling declarative logic-based descriptions of conceptual knowledge. Most of them restrict their expressiveness and focus on (efficient) reasoning algorithms for certain services. This imposes principal restrictions on the expressivity of such a formalism.

To get maximal benefit of terminological reasoning, while being able to overcome these expressiveness deficiencies, the terminological system **Taxon** has been integrated with logic programming by applying a CLP scheme to its assertional formalism.

Because **Taxon** itself is an amalgamation of an abstract concept language by *concrete domains* (such as predicates over rational numbers) we have a three-layered system architecture: 1) rules on the basis of a tuned vocabulary which is formulated in 2) a concept language that is grounded by 3) concrete domains.

We will discuss this layered architecture as a main prerequisite of several advantages in software/knowledge engineering, thus promoting a new discussion about principles of hybrid systems design (which is necessary when trying to gain practical relevance for declarative programming).

1 Motivation

Newer results in knowledge representation (KR) and expert system technology suggest to turn our attention away from the idea of a uniform general-purpose language of declarativity towards heterogeneous hybrid systems. These offer several distinct representation formalisms, each of them especially tailored for natural description (and efficient processing) of certain aspects of the domain knowledge.

Usually, hybrid expert system shells provide mechanisms such as rules, inheritance, constraints etc. (cf. [13]). But most of such shells are rather pragmatic couplings of pragmatic components. A mature understanding of some subsystems (like e.g. constraints, forward chaining rules) has been gained in just recently (see for example [7]), but the overall semantics of complex systems is still far from being clear.

Another trend is to see declarative expert system shells from a practical point of view as very high-level programming languages that gain remarkable advantages for rapid prototyping and software maintenance [23] from their level of abstraction. Powerful new analysis and inference mechanisms on knowledge bases (KBs) promise excellent chances for debugging [21], evolution [19], and reuse of KBs. But such analyses need formally well-understood, highly-integrated and perhaps, not too powerful representation languages [8].

Thus, a useful declarative representation language has to find an appropriate trade-off between a comfortable heterogeneous representation (user-friendly, but with the problem of hard to understand loose integration mechanisms) and a highly-integrated small language (good to analyze, but probably enforcing unnatural encodings of knowledge). Two questions arise: *what* representation means should be employed, and *how* should they be integrated?

In this paper, we will sketch some of the above-mentioned software/knowledge engineering (SE/KE) aspects presenting the design decisions we made for the **TaxLog** system, a close integration of terminological reasoning and logic programming (LP).

But a main concern of the paper can be seen independent from **TaxLog**: make declarative language designers aware of such pragmatic considerations as knowledge reusability or appropriateness of representation formalisms for the differents kinds of knowledge coming up in a specific application domain. Putting such pragmatic considerations into the center of interest may promote the spirit of Brachman [10], who blamed the KR community for having lost sight of their users being caught in technical details.

2 TaxLog: A Close Integration of Terminological Reasoning and Logic Programming

First, let us introduce the language **TaxLog**, also presented in [2, 1].

A **TaxLog** *program clause* is a definite function-free horn clause
$\mathtt{p(x_0)\ :\text{-}\ q_1(x_1),\ldots,q_n(x_n)\quad \&\ ABox(x_0,\ldots,x_n).}$

The expression $\mathtt{ABox(x_0,x_1,\ldots,x_n)}$ denotes a finite collection of *assertional axioms* of the form:

membership assertion:	conceptname(x)
role filler assertion:	(x rolename y)
equality assertions:	(x = y), (x $\neq$ y)
concrete constraint:	concrete_predicate(x_1,...,x_k)

Assertions constrain possible interpretations for a program by giving necessary conditions for the variables involved. These conditions are specified wrt. a terminological background theory that describes the structure of an order-sorted domain for an interpretation.

In such a terminological theory we can declare *primitive concepts, roles*, and *attributes* of our application domain (intended semantics: individual sets, binary relations between individuals, and functional, i.e. one-valued binary relationships). Starting with such primitives, we can compose complex concepts out of more primitive ones using a *concept description language* containing e.g. boolean connectives or quantification over role values. Some examples for *terminological axioms*:

(PRIM female human phd)
(CONC woman $=$ female $\sqcap$ human)
(ROLE child)
(ATTR age)
(CONC mother $=$ woman $\sqcap$
$\quad\quad\quad\quad\quad\quad\exists$ (child).human)
(CONC proud_mother
$\quad\quad\quad\quad\quad = $ mother $\sqcap$
$\quad\quad\quad\quad\quad\quad\forall$ (child).phd) $\sqcap$
$\quad\quad\quad\quad\quad\quad\forall$ (child age).$\geq_{18}$)

Here, we define *female* beings, *humans* and people having a *Ph.D.* degree as primitive, i.e. not further described concepts; we compose the *woman* concept as the set of all humans that are female, and a *mother* as a woman with *atleast one* female filler for the *child* role. In the definition of a *proud_mother*, we use both a concept name and a concrete predicate for value restriction: *all* fillers of the child role have to be members of the *phd* concept and the value of the *age* attribute for all children has to be a member of the concrete domain of real numbers. This *concrete domain* provides a predicate testing its input for being greater than 18.

[5, 15] describe the terminological system (TS) **Taxon** and give a formal specification of the concrete domain extension: concrete domains (CDs) consist of

(i) a set of concrete individuals

(ii) a set of concrete predicates (this set has to be closed under negation), and

(iii) a decision procedure for the satisfiability problem for conjunctions of partial instantiated predicates.

Concrete domains and abstract domain of discourse have to be disjoint. Concrete individuals can be referred to as attribute values of abstract individuals. Using attributes as a link, we can integrate such things as e.g. real numbers, relational database objects, some notion for time or space, or a finite domain constraint system into our logic-based framework.

Since a concrete domain has to provide only a satisfiability test, we can use efficient decision procedures encapsulated in black-boxes for concrete domain implementations.

A main service of terminological systems is e.g. the computation of the subset-superset hierarchy of concepts based on the intensional concept descriptions.

Up to now, we have only regarded the terminological module (called TBox) for defining and processing concept descriptions. Additionally, TSs usually have an *assertional module* (ABox) for individual reasoning. Here we can state ground instantiations (assertional axioms) of the structures defined in the TBox and reason about single individuals. Important ABox services are the satisfiability test for collections of assertions (also called ABoxes) or the realization of individuals: find the most specific concept an individual belongs to (wrt. the given ABox and TBox).

A central point is that (in contrast to most other TS) **Taxon** supports complete and terminating algorithms for all services mentioned. Thus, the ABox formalism can serve as constraint theory implementation for the constraint logic programming scheme by Höhfeld & Smolka [20].

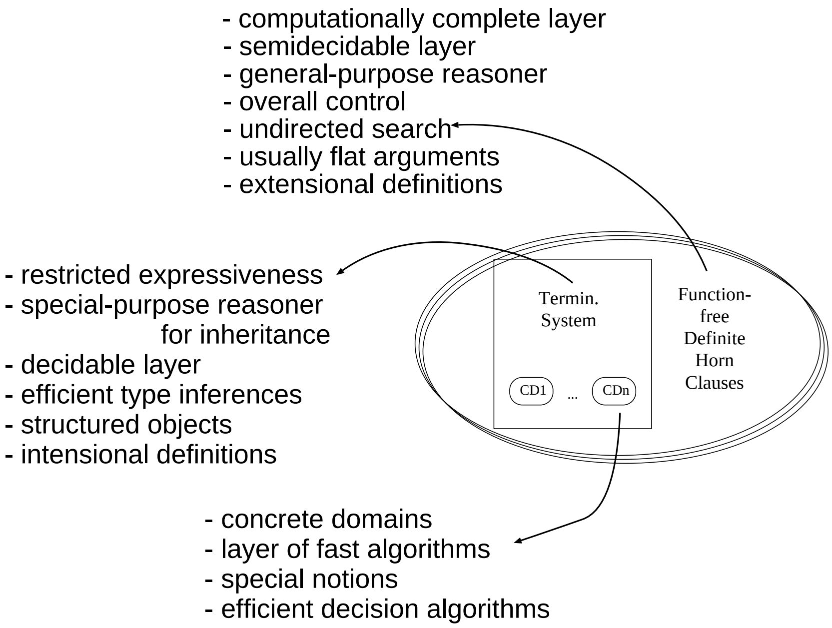
This results in a rather expressive CLP language because we have both structured types (with some kind of inheritance, object-centered KR resp.) and several constraint solvers integrated by the terminological layer. Thus, the LP part is enriched by a powerful semantical typing facility whereas the TS (that paid for decidability and efficiency with some representational deficiencies [17]) obtains general computational power by the integration into the rule formalism.

TaxLog programs are executed by the following interpreter:

solve(Program,R&ABox) :=	
1. IF ABox not consistent THEN RETURN 'fail	Consistency Test
2. IF R=empty THEN RETURN ABox	
3. select don't care an atom p(X) from R	Goal Selection
4. select don't know a (renamed) program clause D = p(X) :- G & DABox IF there is no such clause THEN RETURN 'fail	Clause Selection
5. NewR := (R - p(X)) + G NewABox := ABox + DABox	Goal Reduction (Backward Chaining)
6. solve(Program,NewR&NewABox)	

It implements a standard evaluation procedure for CLP programs where the consistency (satisfiability) test of **Taxon** ABoxes replaces unification and **Taxon** ABoxes are used to represent *conditional answers*. Because we will concentrate on the **TaxLog** architecture, see [2, 1] for more details of syntax and semantics.

The architecture is presented in the following picture:



Next we will see some characteristics of our language design comparing it with the most important similar approaches in LP and KR.

3 Some Related Work

The most related work in LP are *feature logics* that introduce order-sorted types and record-like structured objects in LP by modifying the unification procedure (see e.g. [3]). It is easy to see that the provided representation formalism is a subset of our ABox formalism. This shows a main point of the **TaxLog** philosophy: our typing instrument is very expressive; in some sense it has the maximum senseful expressiveness because a further extension of the terminological formalism would result in undecidability [6]. We will discuss this feature later in more detail. This expressiveness is also a main difference to most conventional LP typing approaches that remain at a purely syntactical level.

In KR there have been implemented several powerful TSs [9, 22] extending the *classification-based paradigm of computation* of TSs step by step, each occuring representational deficiency (see for example [12]) answering with an ad-hoc extension of the TS's representation and reasoning capabilities. These systems may have a similar computational power as our approach, but they are pragmatic developments without proovable features and without complete algorithms. They do not really support a *hybrid* formalism but have brought an object-centered view on reasoning to some perfection. Our approach is different in spirit: we do not go beyond the cliff of undecidability in the TS, but we also do not use the terminological system as the general problem solver.

4 The Effects of the TaxLog Architecture for Hybrid Knowledge Processing

We see two important features of the **TaxLog** architecture:

1. the layered language design
2. the very expressive constraint instrument

Now let us explain why this can facilitate declarative programming.

One main reason is the *modularity* naturally coming with the layered representation structure:

(1) It is more convenient for a user to work with such a layered language, because he/she can formulate several kinds of knowledge, each of them in an especially tailored formalism:

- Efficient non-declarative reasoning mechanisms (e.g. coming from operations research or databases) can be isolated in concrete domains without affecting the declarative part.
- The domain vocabulary can be formulated in the terminological formalism that is rather similar to natural language formulations. But in **Taxon** the concept description language itself can be tailored for a specific application domain by integrating the appropriate concrete domains. Thus, even domain specific notations can be used instead of unnatural encodings.
- Finally, we have a rule language with user-defined, domain-specific types, not only simple inflexible types as in most conventional programming languages.

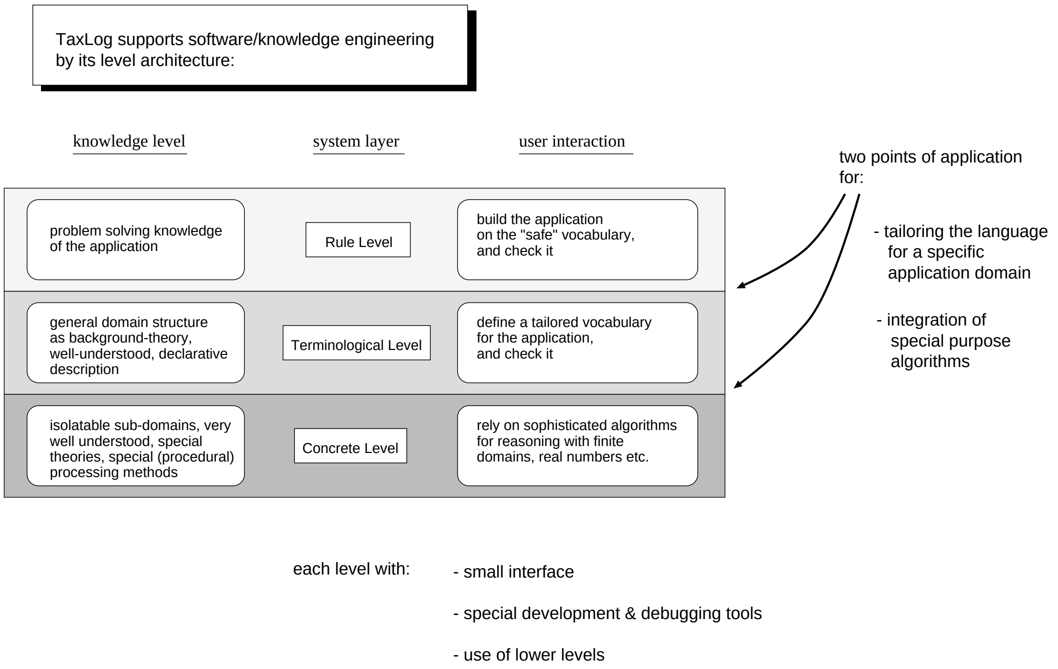
At each level there may exist specialized development and debugging tools that facilitate more effective application programming. For example, concrete domains can be implemented as Lisp programs[1] using comfortable Lisp programming environments, defining the terminological part can be supported by graphical taxonomy browsers and editors, and the rule part can be validated using expert system verification techniques [21] adapted to the **TaxLog** formalism [1].

Furthermore, modularity permits separate development by different people; a single programmer has not to worry about the whole system; he/she has not to know all details of a complex KR language but is only concerned in one level of description. Such a divide-and-conquer strategy applied to KBs may also simplify automatic or partly automatic knowledge acquisition. For example, we could obtain the domain vocabulary from one source (e.g. from text books or by translation of earlier used frame hierarchies) and rules (using this vocabulary as types) from other sources (e.g. from domain experts or from previous cases).

[1]**TaxLog** has been implemented in Common Lisp. Thus, concrete domains have to be implemented as Lisp packages.

(2) The **TaxLog** architecture reflects a similar organisation of knowledge available in an application domain.

Existing procedural knowledge and interfaces to other modules of an information system can be integrated via CDs, the domain dependent vocabulary, well understood and easy to formulate can be represented in the terminological module, and finally, the problem solving knowledge can be expressed by typed rules. [26] shows that such an organisation of the KB is especially appropriate in a technical configuration domain.



(3) The several parts of a KB can not only be implemented independently, they can also be *reused* or *shared* separately. There are concrete domain repositories possible, with some CDs very specialized, others of universal utility. Domain vocabulary (expressed in concept hierarchies) should be application independent too, and thus reusable for other applications.

In addition to these advantages of partitioning the involved knowledge and distributing it to appropriate formalisms, we have another main group of advantages based on the idea that in a layered system each level can benefit from the services of the underlying levels.

Obviously, a rule formalism with an SLD-like interpreter can remarkably improve its efficiency profiting from the pruning power of a type processor that early detects failing branches because of type inconsistencies. This idea is realized in our system in a twofold way. First, the modified tableaux calculus of the terminological system is tuned by its special purpose reasoners for CDs. Second, the terminological level serves as a special purpose reasoner for types and inheritance integrated into the rule formalism. Thus, we have two steps similar to the evolution from LP to CLP.

Please note, that we have extended this feature as far as possible because our typing instrument itself represents a large part of the application knowledge. We can state that at the term level (and thus, also at the level of a single rule) we can express by our ABox formalism much more knowledge than by ordinary Herbrand terms (or even typed Herbrand terms). This is clear seeing that Herbrand terms could be simulated in **Taxon** using attributes and primitive concepts, whereas in **TaxLog** we have additionally arbitrary roles, quantification over fillers and concrete domains.

Clearly, we could encode the same information by a conventional rule formalism. What is the advantage of having it isolated in a terminological system? First, we have specialized algorithms that can really *prune* the search space while additional premises in rules will enlargen it. Second, we can precompute important terminological relationships and store them in special data structures (especially for the subsumption relations or disjointness of concepts). Third, and most important, we can do KB analyses (e.g. for indexing, compilation, or validation of rules) more locally because a *single* **TaxLog** rule contains information that would been formulated in a conventional rule language using a *set* of rules. This decreases complexity when examining the KB for features such as inconsistent information.

Moreover, it offers with the terminological concept hierarchy a more fine-grained structure for propagating variable modes doing *abstract interpretation*, or for weakening/strengthening rule premises doing *KB refinement* [14]. [18, 19] show the link between expressive rule formalisms (such as **TaxLog**) and powerful *KB evolution* scenarios.

5 Conclusions

We gave an overview of possible advantages of the **TaxLog** architecture, a hierarchical integration of rules, types, and constraints. Several constraint solvers can be integrated via the terminological type system into the rule component.

From a formal point of view, this can be seen as a special instance of the CLP scheme by Höhfeld and Smolka.[2] This results in a rather natural combination of object-centered KR and relational LP. Such integrations have been proposed several times (see e.g. [4]), but, to our knowledge, never have been implemented, nor further investigated. Since most conventional constraint solvers can be used as concrete domains, our approach allows for the elegant combination of several constraint solvers plus expressive structured types. Because the terminological part can be seen as a sublanguage of first order logic, its extension by concrete domains is rather similar in spirit to the step from LP to CLP. So we have in some sense a two-step CLP scheme.

But the main concern of this paper was not to promote the **TaxLog** language but to talk about relevant issues when trying to bring declarative programming to practice. In our opinion, such aspects could be: user friendliness, manyfold representation means, interfaces to procedural programs or databases, knowledge sharing and reuse, knowledge validation and evolution. All these issues are affected by the language design and become increasingly

[2]This scheme is characterized by its possible world semantics: a constraint theory like our terminological theory can have several models, whereas in the scheme by Jaffar and Lassez a constraint theory is given by a single model.

more difficult when providing rich, expressive representation languages.

We have described three design decisions in order to tackle this effect: (1) use a terminological system, (2) use a highly integrated layered language architecture, and (3) try to let the several layers as expressive as possible.

Let us elaborate these three points a little bit more in the following:

(1) The LP community should consider both terminological logics and constraints as possible extensions. TSs could be especially worthful when being forced to process large amounts of data (cf. [24]) – what is the case in each real world application. Seen in this light it seems to be clear that the focus of interest within the TL community should shift from the TBox to efficient ABox reasoning incorporating large databases. The integration of special purpose reasoners in terminological systems (TS) is an indispensible need. Our scheme [5] gives a formally clear foundation to do this. In our opinion, the extension of the classification-based reasoning paradigm to a universal problem solver is one way for using TSs, but not the only relevant one. The way proposed in this paper: leave the TS restricted in both representational and inferential power in order to get efficient decidable reasoning services and use the TS as a subordinate component offering its services in an overall heterogeneous system (although being the core component, as in **TaxLog**) is more according to the *restricted language architecture* idea of Vilain [25].

(2) Our decision for a layered arrangement of system components in contrast to flat integrations as in some expert system shells, is the main characteristic of **TaxLog**. It seems to provide essential advantages concerning software/knowledge engineering aspects (that should move more to the center of our interest). Of course, most of the potential advantages must be proven by evaluation in practical use, and some of the possible application areas (KB validation and evolution, knowledge sharing and reuse) have become the object of scientific research just recently (see for example [19]).

(3) The second main design decision was: when using a layered architecture, nevertheless try to let the integrated parts as powerful as possible. This is the prerequisite for an optimum success of the above mentioned advantages. Especially, it results in a higher program analysis potential for compilation and validation which benefit from a more comprehensive expressivity at the term level and from precomputed information concerning the terminological part.

The proposed **TaxLog** system has been prototypically implemented using Common Lisp. Furthermore, Wache [26] implemented a system for configuration in a technical domain along the **TaxLog** philosophy. However, other aspects like e.g. compilation or validation tools have not yet been implemented.

Another theoretical point that could not be discussed in this paper is that even couplings of several rather different subsystems can be done semantically clear using CLP schemata, but that probably we have to learn something about "strange new logics". Very general CLP approaches bring us near to abductive reasoning [11], and [16] even must use epistemological logic to give a semantics to a further generalisation of the rule formalism presented here.

Acknowledgement

The work presented here is essentially based on ideas by Philipp Hanschke. Parts were supported by the German "Bundesministerium für Forschung und Technik" under grant ITW 8902 C4.

References

[1] A. Abecker. TaxLog: Taxonomische Wissensrepräsentation und logisches Programmieren. Diploma thesis, University of Kaiserslautern, 1994. In German.

[2] A. Abecker and Ph. Hanschke. TaxLog: A flexible architecture for logic programming with structured types and constraints. In M. Meyer, editor, *Proc. WS Constraint Processing, CSAM'93*. DFKI RR-93-39, 1993.

[3] Hassan Aït-Kaci and Roger Nasr. Login: a logic programming language with built-in inheritance. *The Journal of logic programming*, 3:185–215, 1986.

[4] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J.H. Siekmann. Concept logics. DFKI RR-90-10, 1990.

[5] F. Baader and Ph. Hanschke. A scheme for integrating concrete domains into concept languages. In *IJCAI'91*, 1991. Also as DFKI-RR-91-10.

[6] F. Baader and Ph. Hanschke. Extensions of concept languages for a mechanical engineering application. In *GWAI-92*, 1992.

[7] H. Boley, Ph. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: a hybrid compilation laboratory. In *3rd Int. WS on Data, Expert Knowledge and Decisions*, 1991. Also as DFKI RR-93-03.

[8] H. Boley, Ph. Hanschke, K. Hinkelmann, M. Meyer, and M.M. Richter. VEGA - knowledge validation and exploration by global analysis. Project Outline, 1992.

[9] R.J. Brachman, D.L. McGuiness, P.F. Patel-Schneider, L. Alperin Resnicka, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In *J. Sowa: Principles of Semantic Networks*. Morgan Kaufmann, 1990.

[10] R.L. Brachman. The future of knowledge representation. In *AAAI'90*. The MIT Press / AAAI Press, 1990.

[11] H.-J. Bürckert and W. Nutt. On abduction and answer generation through constrained resolution. DFKI RR-92-51, 1992.

[12] J. Doyle and R.S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *AI*, 48:261–297, 1991.

[13] M.W. Firebaugh. *Artificial Intelligence, A Knowledge-Based Approach.* PWS-KENT Publisher Company, Boston, 1988.

[14] A. Ginsberg, Sh. Weiss, and P. Politakis. SEEK2: a generalized approach to automatic knowledge base refinement. In *IJCAI'85*, 1985.

[15] Ph. Hanschke. Specifying role interaction in concept languages. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92*. Morgan Kaufmann, 1992.

[16] Ph. Hanschke. *A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning.* PhD thesis, University of Kaiserslautern, 1993. Also as DFKI RR-93-46.

[17] Ph. Hanschke and K. Hinkelmann. Combining terminological and rule-based reasoning for abstraction processes. In *GWAI-92*. Springer LNAI 671, 1992. Also as DFKI RR-92-40.

[18] Ph. Hanschke and M. Meyer. An alternative to theta-subsumption based on terminological reasoning. In C. Rouveirol, editor, *WS on Logical Approaches to Machine Learning, ECAI-92*, 1992. Also as DFKI RR-92-38.

[19] K. Hinkelmann, M. Meyer, and F. Schmalhofer. Knowledge-base evolution for the manufacturing with new materials. *AICOM*, 1994. To appear.

[20] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. Lilog-Report 53, IBM Deutschland, 1988.

[21] B. Lopez, P. Meseguer, and E. Plaza. Knowledge based systems validation: A state of the art. *AICOM*, 3(2), 1990.

[22] R. MacGregor and M.H. Burstein. Using a description classifier to enhance knowledge representation. *IEEE Expert*, June 1991.

[23] C. Moss. Commercial applications of large Prolog knowledge bases. In H. Boley and M.M. Richter, editors, *Proc. of the Int. WS on Processing Declarative Knowledge (PDK'91)*, LNAI 567. Springer, 1991.

[24] B. Nebel and Chr. Peltason. Terminological reasoning and information management. In D. Karagiannis, editor, *Information Systems and Artificial Intelligence*. Springer.

[25] Marc Vilain. The restricted language architecture of a hybrid representation system. In *IJCAI'85*, August 1985.

[26] H. Wache and P. Tsarchopoulos. Ein erweitertes CLP-Schema für eine hybride Wissensverarbeitung. In H. Boley, F. Bry, and U. Geske, editors, *Proc. WS Neuere Entwicklungen der deklarativen KI-Programmierung auf der KI-93*. DFKI RR-93-35, 1993. In German.