# On Domain-Specific Declarative Knowledge Representation Languages[1]

**Stefan Decker**
**Institute AIFB, University of Karlsruhe (TH),**
**D-76128 Karlsruhe**
**e-mail: decker@aifb.uni-karlsruhe.de**

Abstract

For todays knowledge acquisition and engineering tasks many logic based representations formalisms are available. However, most of them do not support the elicitation and representation of domain specific knowledge. On one hand building special systems, which supports these elicitation and representation tasks is a costly and cumbersome process. The direct implementation of inference mechanisms reflecting the semantics available in a specific domain leads to an inflexible system: a small change in the input language results in changes of the whole system. On the other side, domain-specific languages obtain more and more interest. In this paper we propose to built domain-specific declarative languages using approved techniques from deductive databases and logic programming. The enables the usage of existing efficient and elaborated systems. The ideas are illustrated by a reengineering and combination of two declarative representation languages: Frame-Logic and Linear Temporal Logic.

## 1 Introduction

Logic based knowledge representation and specification has a long tradition in AI (see e.g. [Hay77][Mac90][vHB92][Fen95a][Bri??]), so for todays knowledge acquisition and engineering tasks many representations formalisms are available. However, most of them do not support the elicitation and representation of domain specific knowledge. Using First Order Predicate Logic (FOL) for knowledge representation tasks is comparable to using a Turing Machine for programming tasks: it is a sufficient formalism for studying theoretical properties, but does not support real life tasks. Of course a turing machine is powerful enough from a *principal* point of view, but not from a *practical* one: the encoding is often to expensive. The reason for this is, that the language (FOL) does not support thinking in domain terms. Instead, the domain terms must be translated to the terms of the representation language. Therefore, similar to the development of high level programing languages, we propose to use First Order Logic as a kind of low level language for building domain-specific declarative languages (DSDLs) (see [DSL97]), such that higher level representation languages can be compiled into this language. Especially for horn logic with "negation as failure" exists a number of implemented efficient systems (e.g. CORAL [RSS+94], XSB [RSS+97], ADITI [VRK+94], ConceptBase [JGJ+95], LogicBase [HLX94] among others), which can be used as a kind of "virtual machine" for inferencing and execution, getting a formal and operational semantics for domain-specific languages for free. The expressive power of the logics used in the former mentioned systems is well understood and several semantics are defined (for an overview see e.g. [BrD96][Llo87]). Describing this approach using Brachmans stratification for knowledge representations [Bra79], the implementation level are data structures and algorithms used in the deductive database or logic programming system, the logic level is Horn logic with negation, and the epistemological level realizes adequate domain dependent representation primitives through compilation to the lower levels. This approach is similar to approaches in compiler construction: usually an initial language is not compiled directly to a target language, but through several intermediate stages and languages. This helps bridging the conceptual gap between the initial language and the target language (e.g. the first C++ compiler was nothing more than a preprocessor for C programs). The direct implementation of the semantics used in a specific domain leads to an inflexible system: a small change in the input language results in changes of the whole inference system. We show, how this idea can be exploited in knowledge representation, leading to special purpose knowledge acquisition and representation systems, that are easy to built, to maintain and to adjust to different tasks.

We following sections we illustrate the general approach through a kind of reengineering of two special representation languages: Frame-Logic [KLW95], that is used to model object orientation, and Chronolog (see e.g. [LiO96][OrM94]), that realizes Linear Temporal Logic. Then the combination of both approaches is sketched.

---

1. The work reported in this paper is supported by the BMBF (German Ministry of Education, Science, Research and Technology) under grant number 01HS014

## 2 Building DSDLs

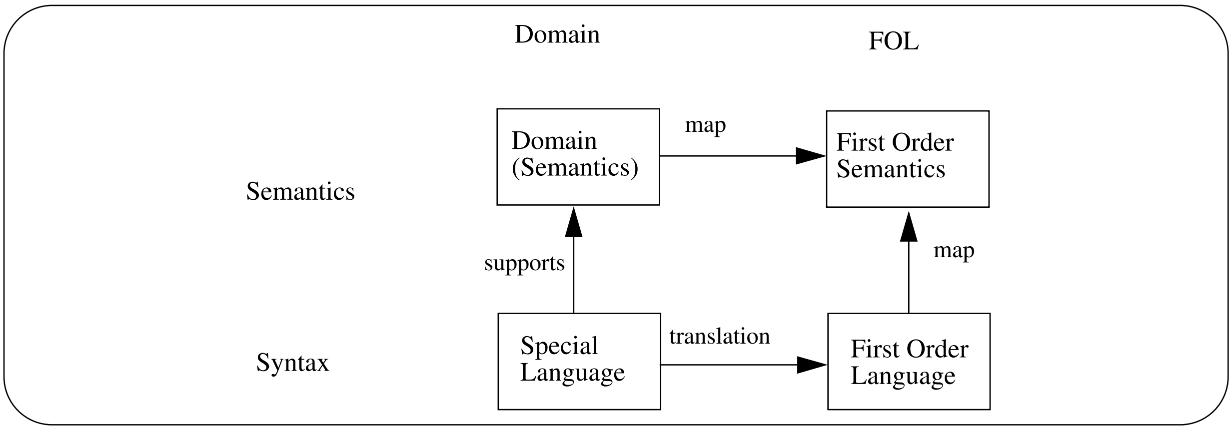


**Fig. 1 .** Relationship between DSDL and FOL

A domain-specific declarative language should support the elicitation and representation of knowledge of a particular domain. For building such a language the transfer of the usual concepts of syntax and semantics in FOL is useful (see figure 1): a domain specific language should help to model aspects of the domain. So the domain to represent determines the semantics of the language. To map this language appropriate to first order syntax, at first the semantics has to be mapped. This suggests the following methodology:

- The first thing to do, when building a DSDL is to clarify the concepts of the specific domain. For object oriented systems such concepts can be e.g. classes, instances and methods or special individuals.
- The next step is to map these semantics concepts to the usual first order semantics. That means the concepts of the domain have to be expressed through e.g. predicates, functions, constants or formulae.
- The most important and most difficult step is to identify syntactic constructs, that support the elicitation and representation of knowledge in an appropriate way. There are several possibilities: special operators can be identified, such that relations from the domain are now expressed with operators. If these operators have a built-in semantics, these semantics can be captured with appropriate axioms. Other domain concepts can be captured via special function symbols. Some useful principles are illustrated later on.
- According to the identified mapping from the domain semantics to the FOL semantics an appropriate translation from the DSDL syntax to FOL syntax can be defined. What was identified as a relation in the domain analysis is translated to a predicate. Special constants from the domain language are translated to special constants in the target language. This translation is needed in both directions: the set of formulas have initially translated from the domain syntax to the syntax of the deductive database or logic programming system, and the inferencing results (usually variable substitutions) have to be translated back to the domain syntax.

## 3 Examples

We illustrate the general approach through a reengineering of two special representation languages: Frame-Logic, that is useful for modeling in an object oriented style, and Chronolog, that realizes Linear Temporal Logic and allows to model states and state changes. Although both languages have a well defined model and proof theory and there exists implemented inference engines, we show that inference engines for these languages can also be obtained via a translation approach and the usage of one of the inference engines for standard horn logic. So the languages can be adapted to different purposes, can be combined, and techniques available for deductive databases are directly applicable.

### 3.1 Frame-Logic

We start with analyzing part of the semantic structure of Frame-Logic (abbr. F-logic) (see [KLW95]). F-logic is designed as a declarative language, that accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages ([StB86][RBP+91]). F-logic has its own model and proof theory and a specialized inference engine is available ([FHK+97]). The structure of F-logic formulae are similar to usual FOL-formulae. The main difference are the atomic expressions, which are called molecules in F-logic. Therefore we give translations only for the molecular expressions of F-Logic, everything else can be used without translation.

It seems difficult to transfer results available in the deductive database area to the inference engine for F-logic: only recently, a semi-naive evaluation strategy for F-logic was defined in [Sch97] and optimized evaluation strategies such as magic sets are still missing. In the following we analyse some of the most important semantic concepts used in F-logic:

- subclassing: This is an important concept in object oriented languages (see [StB86][RBP+91]). Properties of subclassing are e.g. transitivity, reflexivity.
- class membership: This concepts allows to identity, which element of the universe of discourse is a member (or instance) of another element.
- methods: This concepts identifies, when a element of the universe of discourse can be interpreted as a method of other elements and what result is returned on invocation of the method.
- method definition: This concept identifies, which classes understand which methods, and what is the class of the result.

The next step is to map these object oriented semantic concepts on usual first order concepts: In these cases all concepts are mapped to relations of the target semantics, e.g. subclasses, class membership and methods are identified through special relations of the usual first order interpretations.

There are of course several possibilities to support these concepts syntactically in a special purpose language. However, the developers of F-logic have adopted notations that are similar to other languages:

- "C :: D" for subclassing, meaning that class C is a subclass of class D
- "O : C" for class membership, meaning that O is an instance of class C
- "O[M->V]" for method invocation, meaning that the instance O has a method M with value V.
- "C[M=>D]" for method definition, meaning that all instances of the class C understand the method M and the class of the result returned is D.
- Furthermore, to allow a more flexible handling, several syntactic constructs can be combined, e.g., "O:C[M->>V:D]" is an allowed construct.

An example F-logic specification is depicted in figure 2. Here a part of a traffic specification is shown, defining two classes (Traffic_Sign and Color), one method (status) of the class Traffic_Sign and three instances (the colors). Also it is of course possible to specify rules about the objects, but it is very cumbersome to specify the dynamic behavior and change of objects without any possibilities to express the change of states. Translation has not only to adopt the relational structure of the concepts, but also the domain specific properties of the domain, e.g. transitivity of the subclass relationship. Therefore we have not only to do the syntactic translation, but also the axioms, that capture the domain specific properties of the constructs.

```
Sign :: Object. Color :: Object.
Traffic_Light::Sign[status=>Color].
green : Color. amber : Color.
red : Color.
```

**Fig. 2 .** Part of a F-logic Traffic Specification

| Translation: | Object Oriented | | First Order |
|---|---|---|---|
| | C::D | "" ⇒ "" | sub(C, D) |
| | O:C | "" ⇒ "" | instance(O, C) |
| | O[M->>V] | "" ⇒ "" | method(O,M,V) |
| | C[M=>>D] | "" ⇒ "" | methodtype(C,M,D) |
| | O:C[M->>V:D] | "" ⇒ "" | instance(O, C) ∧ method(O,M,V) ∧ instance(V, D) |

| Axioms: | | | |
|---|---|---|---|
| | Reflexivity: | ∀X | sub(X, X) |
| | Transitivity I: | ∀X, Y, Z | sub(X, Z) ← sub(X, Y) ∧ sub(Y, Z) |
| | Transitivity II: | ∀X, Y, Z | instance(X, Z) ← instance(X, Y) ∧ sub(Y, Z) |
| | Acyclicity: | ∀X, Y | (X = Y) ← sub(X, Y) ∧ sub(Y, Z) |

**Fig. 3 .** Simplified Translation Table for F-logic

The syntactic translation schemes and some of the axioms are shown in figure 3. The shown example rules translate only a part of F-logic, but the rest can be handled in a similar way. In figure 4 the translation is depicted.

```
sub(Sign,Object). sub(Color,Object).
sub(Traffic_Light,Sign).
methodtype(Traffic_Light,status,Color).
instance(green,Color). instance(amber,Color).
instance(red,Color).
```

**Fig. 4 .** Translation of the F-logic Traffic Specification

### 3.2 Chronolog

Temporal logic has been widely used as a formalism for several purposes, e.g. program specification and verification. Chronolog is a language based on linear-time temporal logic. Semantic concepts are the *initial state* and the *next state*. The semantic structures of Chronolog are described as *temporal Herbrand models* [LiO96], such that every element of the temporal Herbrand model is true in exactly one moment in time. The semantic translation is as follows: every relation from the temporal Herbrand model is extended with an extra argument. This argument contains a counter, that stands for the moment in time, in which this fact is valid. The syntactic support for specification is achieved through two temporal operators, *first* and *next*, which refer to the initial and the next moment in time respectively. An atom can have a prefix of the from "`first next`$^n$", "`next`$^n$", or none prefix at all, meaning that this is valid in all moments of time.

The example depicted in figure 5 is taken from [OrM94]. The program specifies the simulation of a traffic light. At the initial state the color of the traffic light is green. The clauses then define the value of the color in the next states. The series goes `green, amber, red, green, amber, red, green` and so on. This example can queried for the color of the light in the nth point of time.

```
first light(green).
next light(amber) <- light(green).
next light(red) <- light(amber).
next light(green) <- light(red).
```

**Fig. 5 .** Chronolog example specifying a traffic light

The syntactic translation is similar to the semantic translation: every atom is extended with an extra argument. This extra argument is used as a time counter and the first-next sequence is coded into this extra argument according figure 6: the first operator is associated with a constant 0, symbolizing the initial point of time. The unary function symbol s if used to encode the next operator.

| Chronolog | First Order Logic |
|---|---|
| `first light(green).` | `light(green,0).` |
| `next light(amber) <- light(green).` | `light(amber,s(X)) <- light(green,X).` |
| `next light(red) <- light(amber).` | `light(red,s(X)) <- light(amber,X).` |
| `next light(green) <- light(red).` | `light(green,s(X)) <- light(red,X).` |

**Fig. 6 .** Translation schema for the Chronolog example

### 3.3 Combining F-logic and Chronolog

Taking these two translation tables as a starting point, both approaches, F-logic and Chronolog can be combined, allowing declarative and executable specifications about objects changing their properties over time. As an example see figure 7, combining the two specification about the traffic domain (figure 2 and figure 5). This is in the same spirit as described in [MSL97] however, we have no need to built a special purpose inference engine. Instead, we just make a translation to usual deductive database formalisms. For performing this kind of translations, the different translation tables have to be combined. This can be done through a two step process as depicted in figure 9: in the first step the temporal annotations are ignored and just the translation as given in figure 3 is performed. If some F-logic molecules are translated to several atoms, the temporal annotations are distributed over the atoms. The result is an ordinary Chronolog program, so that the second translation table (figure 6) is applicable. The result of the final translation (without the axioms specifying the behavior) is depicted in figure 8

```
Sign :: Object. Color :: Object.
Traffic_Light :: Sign[status=>Color].
green : Color. amber : Color.
red : Color.
tl : Traffic_Light.
first tl[status->green].
next tl[status->amber]<-tl[status->green].
next tl[status->red]<-tl[status->amber].
next tl[status->green]<-tl[status->red].
```

**Fig. 7 .** Combining F-logic and Chronolog

```
sub(Sign,Object). sub(Color,Object).
sub(Traffic_Light,Sign).
methodtype(Traffic_Light,status,Color).
instance(green,Color). instance(amber,Color).
instance(red,Color). instance(tl, Traffic_Light).
method(tl,status,green,0).
method(tl,status,amber,s(X))<-method(tl,status,green,X).
method(tl,status,red,s(X))<-method(tl,status,amber,X).
method(tl,status,green,s(X))<-method(tl,status,red,X).
```

**Fig. 8 .** Translation of the combined Specification

## 4 Related Work

Although Knowledge Representation is an important aspect for building knowledge based system, there has been not much work in domain-specific declarative representation languages, e.g. F-logic and Chronolog are not developed for domain-specific modeling: instead both have a elaborated proof and model theory, which makes it difficult to reuse and to extend the languages for different purposes quickly. The languages KARL, used to model KADS-models of expertise ([AFS94][Fen95b]) uses part of F-logic for modeling purposes. Although the specification of the operationalisation defines a direct semantics for the language ([Ang93]), the interpreter is realized through a compilation approach. $(ML)^2$ ([vHB92]) uses only predicate logic for knowledge representation, and no special epistemological primitives are possible for domain modeling. The same is true for DESIRE ([LPT92] ), although temporal logic is used for modeling purposes, but this logic is hardwired into the system.

Of course also other authors have noticed, that their logical systems are often easy translatable to First Order Logic. E.g.

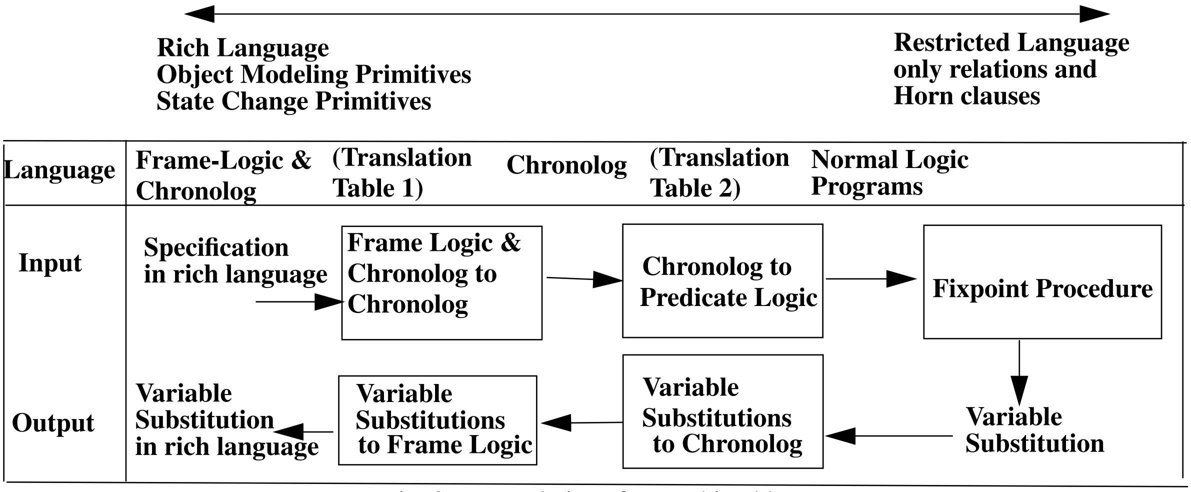


Fig. 9. Translations for combined language

Kifer, Lausen, and Wu argue in [KLW95] in fovour of a direct semantics for F-logic:"*The syntax of a programming language is of paramount importance, as it shapes the way programmers approach and solve problems. However, syntax without semantics is not conducive to programming. Third, a direct semantics for a logic suggests ways of defining proof theories tailored to that logic. Such proof theories are likely to be a better basis for implementation that the general-purpose classicial proof theory.*" As much as we agree with the first argument, as much we disagree with the second and third: if a using a modelling language requires the understanding of its formal semantics, then the language is not particulary useful. How many Cobol, C, C++, Java or even Prolog programmers understand the formal semantics of the language they use? In spite of this they are doing many useful things with the languages. Furthermore, building a specialized inference engine for a special semantics require and extraordinary effort. It might be true, that when optimizing this special inference engine, the performance of if might be better than using a inference engine for the general proof theory. However, usual no one spends the effort, whereas the general inference engines are allready highly optimized. The late development of even one of the basic optimization techniques known in deductive databases to be usable for F-logic supports this claim [Sch97].

## 5 Conclusions and Future Work

Domain specific languages support the thinking in domain terms, instead of thinking in terms of the representation language: when modeling in an object oriented style, a object oriented language supports thinking in terms of classes, objects, and methods instead of relations and functions. Although the given examples are applicable in many domains (and by this not really domain depended) they show how to built more specific languages. We presented a methodology for defining domain-specific declarative language and „reengineered“ two existing knowledge representation languages. We showed how these languages can be easily combined to obtain a richer one. This approach is similar to approaches used in programming languages design: to bridge the conceptual gap between the start language and the target language several intermediate languages are used.

Although some part of the work described above are used in the realization of the Ontobroker system ([FDE+98]), future work includes to built a library of domain specific language parts, which can then be combined in the described way.

**Acknowledgment.**

Discussions with Jürgen Angele, Michael Erdmann, Dieter Fensel, Rainer Perkuhn, and Rudi Studer are gratefully acknowledged.

## 6 References

[Ang93] J. Angele: Operationalisierung des Models der Expertise mit KARL, Infex, St. Augustin, 1993

[AFS96] J. Angele, D. Fensel, and R. Studer: Domain and Task Modeling in MIKE. In: Proceedings of the IFIP WG 8.1/13.2 Joint Working Conference, Domain Knowledge for Interactive System Design, Geneva, May 1996.

[AFS94] J. Angele, D. Fensel, and R. Studer: The Model of Expertise in KARL. In: Proceedings of the 2nd World Congress on Expert Systems, Lisboa/Estoril, Portugal, January 1994.

[Bra79] Brachman R. J., "On the Epistemological Status of Semantic Networks", in Associative Networks:Representations and Use of Knowledge by Computers, Findler, N.V: Academic Press, 1979, pp. 3-50.

[Bri??] Selmer Bringsjord: Logic and Artificial Intelligence: Still Married, Divorced, Separated...? forthcoming in: Minds and Machines. See also: http://www.rpi.edu/~brings/LOG+AI/lai/lai.html

[BrD96] G. Brewka and J. Dix . Knowledge representation with logic programs. Technical report, Tutorial Notes of the 12th European Conference on Artificial Intelligence (ECAI '96), 1996. Also appeared as Technical Report 15/96, Dept. of CS of the University of Koblenz-Landau. Will appear as Chapter 6 in em Handbook of Philosophical Logic, 2nd edition (1998), Volume 6, Methodologies.

[DSL97] Proceedings of DSL'97: First ACM-SIGPLAN Workshop on Domain-Specific Languages. University of Illinois Computer Science Report, URL: www-sal.cs.uiuc.edu/~kamin/dsl

[Fen95a] D. Fensel: Formal Specification Languages in Knowledge and Software Engineering, The Knowledge Engineering Review, vol 10(4), 1995.

[Fen95b] D. Fensel: The Knowledge Acquisition And Representation Language KARL, Kluwer Academic Publisher, Boston, 1995.

[FDE+98] Dieter Fensel, Stefan Decker, Michael Erdmann, and Rudi Studer: Ontobroker: Transforming the WWW into a Knowledge Base, submitted to KAW'98.

[FHK+97] J. Frohn, R. Himmeröder, P.-Th. Kandzia, G. Lausen, C. Schlepphorst: FLORID - A Prototype for F-Logic. In:Proc. Intl. Conference on Data Engineering (ICDE, Exhibition Program), Birmingham, 1997; IEEE Computer Science Press. See also: http://www.informatik.uni-freiburg.de/~dbis/flogic-project.html

[vHB92] F. van Harmelen and J. Balder: ($ML^2$): A Formal Language for KADS Conceptual Models. In: Knowledge Acqusition, 4(1).

[JGJ+95] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, S. Eherer: ConceptBase - a deductive object base for meta data management. In Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases, Vol. 4, No.2, 167-192, 1995

[KLW95] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, vol 42, 1995.

[LPT92] I. van Langefelde, A. Philipsen, and J. Treur: Formal Specification of Compositional Architectures. In: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria, 1992

[Llo87] J.W. Lloyd: Foundations of Logic Programming, 2nd Edition. Springer-Verlag, Berlin, 1987

[HLX94] J. Han, L. Liu and Z. Xie: LogicBase: A Deductive Database System Prototype. In: Proc. 3rd Int'l Conf. on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, Nov. 1994, pp. 226-233. See also: http://db.cs.sfu.ca/LogicBase/

[Hay77] Pat Hayes: In defense of logic.In: Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1977, pp. 107-114.

[LiO96] Chuchang Liu and Mehmet A. Orgun: Clocked Temporal Logic Programming. In: Proc. of the 19th Australasian Computer Science Conference, Melbourne, Australia, 1996

[MSL97] W. May, C. Schlepphorst, G. Lausen: Integrating Dynamic Aspects into Deductive Object-Oriented Databases.In: 3rd International Workshop on Rules in Database Systems, (RIDS'97) Skövde, Sweden, 1997, Springer LNCS 1312, pp. 20-34.

[Mac90] MacGregor: LOOM Users Manual, ISI/WP-22, USC/Information Sciences Institute

[Org96] M. A. Orgun: On temporal deductive databases. Computational Intelligence, Vol.12, No.2,pp.235-259. 1996

[OrM94] M. A. Orgun and W. Ma: An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach (eds), Proc. of ICTL'94: The First International Conference on Temporal Logic (Gustav Stresemann Institut, Bonn, Germany, July 11-14). LNAI 827, Springer-Verlag Berlin Heidelberg, pp.445-479.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs 1991.

[RSS+97] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David Scott Warren, Juliana Freire: XSB: A System for Effciently Computing WFS. In: Proc. of Logic Programming and Non-Monotonic Reasoning, LPNMR 1997, Dagstuhl, Germany, July 28-31, 1997. Lecture Nortes in Computer Science, Vol. 1265, Springer 1997, pp.: 431-441

[RSS+94] Raghu Ramakrishnan, Divesh Srivastava, S. Sudarshan and Praveen Seshadri: The CORAL Deductive System. In: The VLDB Journal, Special Issue on Prototypes of Deductive Database Systems. 3(2), pp.: 161-210, 1994. See also: http://www.cs.wisc.edu/coral/

[Sch97] C. Schlepphorst: Semi-naive Evaluation of F-logic Programs, Technical Report 85 , Universität Freiburg, 1997.

[SWH+94] A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde: CommonKADS: A Comprehensive Methodology for KBS Development. In: IEEE Expert, December 1994, 28-37.

[StB86] M. Stefik and D.G. Bobrow:Object-Oriented Programming: Themes and Variations. In: The AI Magazine, Vol. 6, No. 4, Winter 1986, pp. 40 - 62

[VRK+94] Jayen Vaghani, Kotagiri Ramamohanarao, David B. Kemp, Zoltan Somogyi, Peter J. Stuckey, Tim S. Leask, and James Harland: The Aditi deductive database system. The VLDB Journal 3(2):245--288, 1994. See also: http://www.cs.mu.oz.au/~kemp/aditi/aditi.html