# On Domain-Specific Declarative Knowledge Representation and Database Languages

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

## Abstract

For knowledge acquisition and engineering tasks many logic based representations formalisms are available nowadays. 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 hand domain-specific languages obtain more and more interest as a means to support user in modeling tasks. In this paper we propose to built domain-specific declarative languages using techniques from deductive databases and logic programming. This 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.



**Proceedings of the 5th KRDB Workshop**
**Seattle, WA, 31-May-1998**

(A. Borgida, V. Chaudhri, M. Staudt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10/

## 1 Introduction

Logic based knowledge representation and specification has a long tradition in AI (see e.g. [Hay77][Mac90][vHB92][Fen95a][Bri??]), so for knowledge acquisition and engineering tasks many representation formalisms are available. However, most of them do not support the elicitation and representation of domain specific knowledge. Using the syntax of 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 (e.g. relations). So to establish a communication between a knowledge engineer and a domain expert, the representation must always translated back to the domain terms. This, however is fortunately a matter of the syntax of the representation language. Therefore, similar to the development of high level programing languages, we propose to use the syntax of 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 closed world semantics exists a number of 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 inferences 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, representation and inference systems, that are easy to built, to maintain and to adjust to different tasks.

In the 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 and some interdependencies and limitations are analyzed.

In the forthcoming examples we use standard syntax with the exception, that only symbols beginning with the capital letters "X,Y,Z" or introduced by a quantifier denote variables.

## 2 Building DSDLs

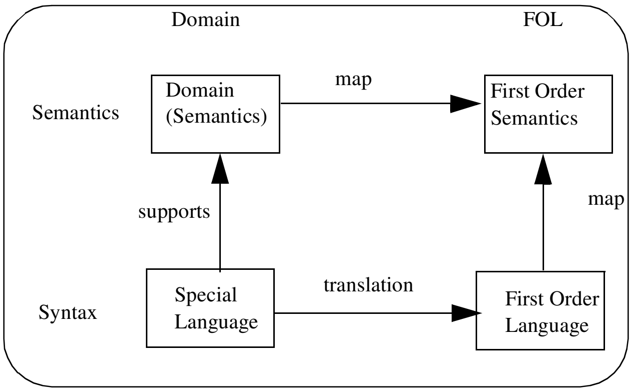


Figure 1: Relationship Between DSDL And FOL

A domain-specific declarative language should support the elicitation and representation of knowledge of a particular domain. So the domain to represent determines the semantics of the language, and for compilation reasons the domain concepts have to be translated to the usual FOL language (see figure 1 for an illustration). This suggests the following methodology:

- Clarification of the concepts of the specific domain. For object oriented systems such concepts can be e.g. classes, instances and methods or special individuals.
- Mapping these semantic concepts to the usual first order semantics. That means the concepts of the domain have to be expressed through 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 is initially (before reasoning takes places) translated from the domain syntax to the syntax of the deductive database or logic programming system, and the inference 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 parts 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 follows directly from this.

Table 1: Translation Schema For F-logic

| Object Oriented | | First Order |
|---|---|---|
| C::D | $\Rightarrow$ | sub(C, D) |
| O:C | $\Rightarrow$ | instance(O, C) |
| O[M->>V] | $\Rightarrow$ | method(O,M,V) |
| C[M=>>D] | $\Rightarrow$ | methodtype(C,M,D) |
| O:C[M->>V:D] | $\Rightarrow$ | instance(O, C) $\wedge$ method(O,M,V) $\wedge$ instance(V, D) |
| Axioms: | | |
| Reflexivity: | $\forall X \quad sub(X, X)$ | |
| Transitivity I: | $\forall X, Y, Z \quad sub(X, Z) \leftarrow sub(X, Y) \wedge sub(Y, Z)$ | |
| Transitivity II: | $\forall X, Y, Z \quad instance(X, Z) \leftarrow instance(X, Y) \wedge sub(Y, Z)$ | |
| Acyclicity: | $\forall X, Y \quad (X = Y) \leftarrow sub(X, Y) \wedge sub(Y, Z)$ | |

It seems to be 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. So using a highly optimized standard logic programming or deductive database systems should still be more efficient for modeling tasks.

In the following we analyze 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.
- method inheritance: In object oriented system the definition of methods in classes are inherited by there instances of the class. So a method definition is usable for all instances of a class, but it is still possible to overwrite it. This aspect of F-logic shows some of the limitations of our approach and will be discussed later on.

The next step is to map these object oriented semantic concepts to 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, and *"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.

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

Figure 2: Part Of A F-logic Traffic Specification

Here a part of a traffic specification is shown, defining three classes (Sign, Traffic_Light and Color), one method (status) of the class Traffic_Light and three instances (the colors). Also it is of course possible to specify rules about the objects, 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 we have also to introduce the axioms, that capture the domain specific properties of the constructs. The syntactic translation schemes and some of the axioms are shown in table 1. The shown example rules translate only a part of F-logic, but most of the rest can be handled in a similar way. In figure 3 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).
```

Figure 3: 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 [Org96] 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 form "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 4 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 be queried

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

Figure 4: Chronolog Example Specifying The Dynamics Of A Traffic Light

for the color of the light in the n-th point of time.

The syntactic translation is done as follows: 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 table 2: the first operator is associated with a constant "0", symbolizing the initial point of time. The unary function symbol "s" (for successor) is used to encode the next operator, e.g. the term "s(0)" is associated with the sequence "first next"..,

Table 2: Translation Schema For The Chronolog Example

| 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). |

### 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 5, combining the two specification about the traffic domain (figure 2 and figure 4). 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

```
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].
```

Figure 5: Combining F-logic And Chronolog

through a two step process as depicted in figure 8: in the first step the temporal annotations are ignored and just the translation as given in table 1 is performed. The result is an ordinary Chronolog program, so that the second translation table (table 2) is applicable. The result of the final translation (without the axioms specifying the behavior) is depicted in figure 6

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

Figure 6: Translation Of The Combined Specification

### 3.4 Interdependencies

There are several interdependencies between the F-logic part and the Chronolog part in the combined language:

- A F-logic molecule can be translated to a set of FOL-atoms (see table 1 for an example). In this case the temporal annotations are distributed over the atoms.
- Another dependency is the occurrence of the well known *frame problem*: introducing the notion of objects and states, one has not only to specify, which properties of an object change over time, but also, which properties don't change. However, this problem can be solved using the non-monotonic behavior oft negation in logic programming: for every predicate "p" occurring in the translation of F-logic (see table 1) two new predicate symbols "direct_p" and "nondirect_p" are introduced. In the bodies of all inference rules "p" is substituted by "nondirect_p" and in the heads of all inference rules "p" is substituted by "direct_p". For every "p" additional inferences rules as given in figure 7 are added to the final result of the compilation process. These rules handle the problem as follows: if there is a direct specification of the arguments for "p" in a certain moment of time, only the first rule is applicable. If there is a value for "X" defined in a previous moment of time, but no direct specification, only the second rule is applicable..

$$\forall \vec{X}, T \quad nondirect\_p(\vec{X}, T) \leftarrow direct\_p(\vec{X}, T)$$
$$\forall \vec{X}, T \quad nondirect\_p(\vec{X}, s(T)) \leftarrow nondirect\_p(\vec{X}, T) \wedge \neg direct\_p(\vec{X}, s(T))$$

Figure 7: Rules For Dealing With The Frame-Problem

### 3.5 Limits Of The Approach

It is well known that definite clauses with function symbols are a turing complete representation formalism. So every computable function can be compiled to definite clauses *in principle*. This means, that we can always built a meta-*interpreter*, that computes the desired semantics for an appropriate compilation result of the domain. However, this is usually not a good idea *in practice*: meta-interpretation introduces an additional level of inefficiency. So we favor a direct compilation, without the introduction of meta-interpretation. So the limits of a direct compilation of some domain specific language are exactly defined by the semantic properties of the target language, e.g. horn clauses with negation in our example: the semantic properties of the domain have to be compatible with the semantic properties of definite clauses. In practice this means, semantic concepts of a domain, that need the computation of e.g. multiple fixpoints, are not directly compilable in clauses.

An example is the multiple inheritance part in F-logic, this requires the computation of multiple fixpoints (see [Kan97] for further details). Nonmultiple inheritance, however, can be handled with rules similar to the rules depicted in figure 7..

## 4 Conclusions And Future Work

Of course also other authors have noticed, that their logical systems are often easy translatable to First Order Logic. E.g. Kifer, Lausen, and Wu argue in [KLW95] in favor 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 classical proof theory.*" As much as we agree with the first argument, as much we disagree with the second and third: if using a modeling language requires the understanding of its formal semantics, then the language is not particular useful. How many Cobol, C, C++, Java or even Prolog programmers understand a 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 requires an extraordinary effort. It might be true, that when optimizing this special inference engine, the performance of if might be better than using an inference engine for the general proof theory. However, usual no one spends the effort, whereas the general inference engines are already highly optimized. The late development of even one of the basic optimization techniques known in the area of deductive databases to be usable for F-logic supports this claim [Sch97].

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, an 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 sketched a methodology for defining domain-specific declarative language and „reengineered“ two existing knowledge representation languages. We showed how these languages can be 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. Also we have chosen logic programming and deductive databases as a target platform for knowledge representation, there are of course other possible ones, e.g.:

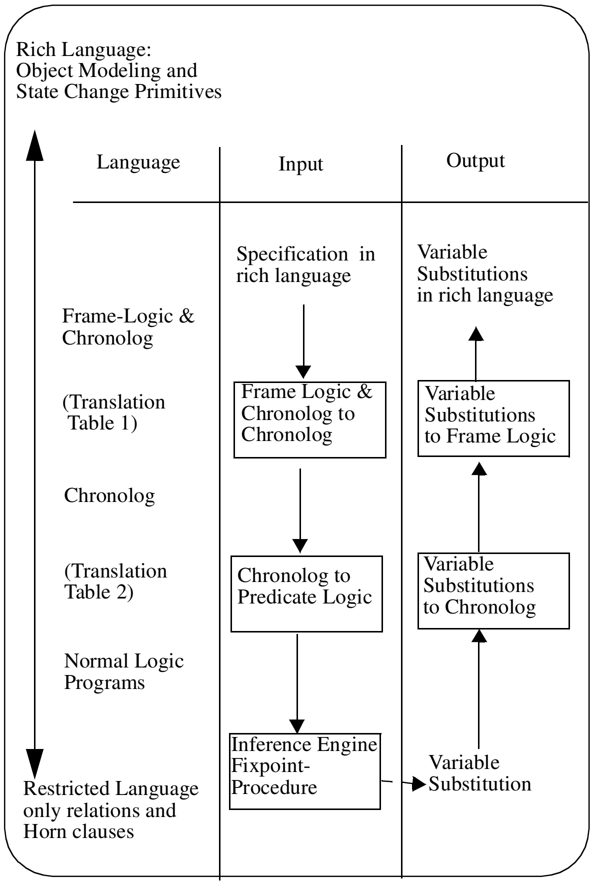


Figure 8: Processes And Languages

- Description Logics, which are usually focused on modeling hierarchies and attributes, using a decidable calculus. It seems to be an issue of further research, if this fragment of FOL can be used as a target platform for domain specific languages, but it obviously could handle hierarchy reasoning in such a language and is probably a part of a more sophisticated system.
- Full First Order Logic, which from a system point of view would mean to use a full fledged theorem prover. Although research on ATP has made remarkable progress over the last years, it is still an open question, if using a full fledged theorem prover for knowledge representation and inference tasks is feasible. However, if theorem provers are used for knowledge representation, the ideas in this paper are certainly applicable.

**Acknowledgments**

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

**References**

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

[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 *The 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*, 10(4), 1995.

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

[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 Acquisition*, 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

[Kan97] Paul-Th. Kandzia: Nonmonotonic Reasoning in Florid. In: 4th International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR'97), July 1997, Dagstuhl Castle, Germany, Springer LNCS 1265

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

[Llo87] J.W. Lloyd: *Foundations of Logic Programming*, 2nd Edition. Springer, 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 Efficiently Computing WFS. In: *Proc. of Logic Programming and Non-Monotonic Reasoning LPNMR 1997*, Dagstuhl, Germany, July 28-31, 1997. Lecture Notes 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*, University of Freiburg, 1997.

[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. In: *The VLDB Journal,* 3(2):245--288, 1994. See also: http://www.cs.mu.oz.au/~kemp/aditi/aditi.html