Published in: M. Papazoglou, editor, Proc. of the 14th Int. Conf. on Object-Oriented and Entity-Relationship Modeling (OOER'95), Gold Coast, Australia, Pages 329–340, LNCS 1021, Springer-Verlag, December 1995.

From Object Specification towards Agent Design*

Gunter Saake   Stefan Conrad   Can Türker

Otto-von-Guericke-Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {saake|conrad|tuerker}@iti.cs.uni-magdeburg.de
Phone: ++49-391-67-18665   Fax: ++49-391-67-12020

Abstract. Nowadays, object specification technology is successfully used for modelling information systems. However, object-oriented modelling cannot cover all aspects (e.g. dynamic behaviour evolution) of such systems. A promising approach to get a grasp of these additional requirements for modelling information systems is the specification of systems as communicating, cooperating and autonomous agents. In the present paper we sketch an evolutionary extension of object specification towards agent modelling. Here, we have to emphasize that this paper is not intended to offer perfect solutions. Our aim is to discuss and fix a research agenda for a formal agent approach which promises to provide a semantically richer framework for the specification of information systems.

1 Motivation

Object-oriented modelling of information systems has become a widely accepted approach. Object-oriented data models allow to describe the structure of application domains in an adequate way. In addition, several approaches make it possible to specify the dynamic behaviour of objects and object systems. In general, versions of temporal and dynamic logic are used for this purpose. However, it seems that the concept of "object" (at least in its current understanding) cannot cover all aspects of information systems. A new concept, the concept of *agent* which can be seen as a further development of "object" is currently emerging as an important concept in many fields of Information Technology. The key facets of the concept *agent* appear to be *goal*, *state*, *behaviour*, and *heterogeneity*.

Considering the modelling of heterogeneous information systems (e.g. [Buc90, Bro92]) motivates the identification of "objects", each of them incorporating some knowledge about an application domain (like facts, rules, strategies, etc.) and being a unit of design that can be composed with other objects via aggregation and specialization hierarchies (cf. [HJS92, SJH93]). Whereas structural aspects of such systems can easily be dealt with by current object-oriented approaches, these approaches succeed to cope with dynamic behaviour only up to a certain degree.

* This research was partially supported by the ESPRIT Basic Research Working Group No. 8319 ModelAge (A Common Formal **Model** of Cooperating Intelligent **Age**nts).

The concept of "agent" is intended to deal with such aspects. In particular, an agent (as a software unit) is assumed to have an internal state which reflects its knowledge (facts, rules, etc.). In order to use its knowledge an agent needs some reasoning capabilities. Furthermore, it must be able to update or revise its knowledge. An agent has goals, i.e. its behaviour is goal-driven. The satisfaction of goals must be achieved by performing (sequences of) actions. Agents may have conflicting goals. In consequence, goal revision must be possible or there must exist a preference order on goals. In composed systems several agents cooperate. For that, some kind of cooperation and communication structure is needed. Agents are assumed to have (incomplete) knowledge about other agents and their goals. In order to satisfy a goal agents may play different roles and can delegate tasks. The concept of agent presents a unique opportunity to integrate many significant results from many different research areas, like Database Systems, Software Engineering, or Artificial Intelligence to name a few. For example, federated information systems may be built up by cooperating (autonomous) agents which are integrated in a flexible framework.

In the remainder of this paper we first discuss requirements for agent modelling (Section 2) and then the current state of object specification technology. In Section 4 we compare the concepts and modelling constructs for objects and agents. Afterwards we present a first way of extending object specification towards agent specification. Finally, we conclude by pointing out future work. It should be clear that in this paper we mainly fix the starting point for a promising research area and present already first ideas.

2 Requirements for Agent Modelling

The concept of *agent* is proposed as a generalization of the object concept to provide a more flexible and semantically richer framework for modelling heterogeneous information systems. Agents can be interpreted as *autonomous* and *intelligent* objects, which are equipped with knowledge and reasoning capabilities to satisfy various goals (cf. [BG88, GK94]). In this section we briefly characterize some important requirements for agent modelling.

Internal State. Agents have an internal *state* which includes an internal, imperfect representation of the world. The internal state of an agent reflects the knowledge of that agent at a given time. But, this knowledge is not fixed at specification time, it is changing throughout the lifetime of an agent. Therefore, agents require knowledge acquisition and revision capabilities to be able to adapt their knowledge or to react to changing environments [DT91]. Furthermore, agents need reasoning capabilities to make use of their knowledge. In addition to state information which is coded in attribute values, the concept of agent requires disjunctive knowledge as well as handling of default knowledge [BLR93].

Goals. Agents have *goals* which they try to achieve (by cooperation) under given constraints. Each agent is obliged to satisfy its goals. In general, this is done by executing a *sequence of actions*. Goals may be changed during the lifetime of an agent. They can be extended, revised or replaced through other (more important) goals. Agents may also have conflicting goals. Hence, agents must be able to resolve conflict situation in which not all goals may be achieved. In such cases agents must be able either to revise some of its goals or to satisfy only a few of its goals which are not conflicting.

Actions. Agents are able to act and communicate. Thus, they show an external *behaviour* that obeys the given constraints and results in life cycles like for objects [DDP93]. The internal state of an agent is determined by its history which is represented by its life cycle. The revision or modification of a state is the result of the execution of a *permitted* action. As already mentioned, agents are autonomous. Therefore, agents can decide independently whether to fulfill a request or not. Thus, they may show a non-normative behaviour.

Cooperating Agents. In most cases agents have to cooperate to achieve their goals. Especially in federated information systems, where agents have to interoperate in order to provide a powerful interface for application programs, appropriate structures and protocols for agent communication and cooperation are required [GK94]. Because of the fact that agents may change their behaviour and/or may even change their signature, there must exist varying communication structures. Agents are assumed to have (incomplete) knowledge about the internal state of other agents and their goals. In order to satisfy a goal, an agent may play different roles, authorize or delegate some tasks to other agents. Hence, an agent must be able to decompose a given goal into subgoals and to assign this subgoals to the various other agents [RK91].

3 Object Specification

We base our presentation on the concept of object as introduced in [SSE87]. An object is an observable process. The dynamic behaviour of objects is described by life cycles built from basic events which may modify the object state. These events can be seen as an abstraction from implemented methods. Attributes can be defined to observe the internal state of an object. An object template serves as pattern for instances of an object class. As a logical framework, we use temporal logic specification of objects using a variant of the object specification logic OSL [SSC92]. Basically, the temporal logic used is a variant of linear temporal logic as used successfully in system specification [FM91, MP92]. For textual presentation of object specifications, we use a notation close to the syntactical conventions of the languages TROLL [SJH93, JSHS95, HSJ$^+$94] and Oblog [SSE87]. Here, we focus on a few basic concepts and omit all language features not relevant for the problem handled in this paper.

3.1 Specification In TROLL-like languages, an object template specification mainly consists out of two parts: a *signature* section which lists object events and attributes together with parameter and co-domain types, and a behaviour section containing the axioms. As axioms we do not have general temporal logic formulas but special syntactic notations for typical specification patterns. For our discussion, we restrict the language features to the following list of specification patterns:

– Attribute valuation:
valuation $\{$`Counter > 0`$\}$ `==> [ Decrease ] Counter = Counter - 1;`
Attribute valuation rules expresses which effect events have on attribute values. These rules can be restricted using conditions on state attributes.
– Birth and death events:
In the declaration section for events, we mark some events as `birth` events or as `death` events corresponding to creation and destruction of objects.
– Event permission: `permissions` $\{$ `Counter > 0` $\}$ `Decrease;`
In contrast to the conditional attribute valuation rule above, a permission restricts the occurrences of events.
– Event obligations: `obligation DestroyCounter;`
Event obligations can be understood as liveness conditions, i.e. the given event must occur in the life cycle of the object. In this example the death event `DestroyCounter` has to occur. This means that we do not allow that the specified object can live forever.
– Event calling: `interaction Withdraw(m) >> IncreaseBookingCounter;`
Event calling is used to model interaction inside (composite) objects. Similar to valuations, we may have conditional event calling, too. In our example, the withdrawal of money of an account object enforces the event `IncreaseBookingCounter` to occur simultaneously.

Each of the mentioned languages supports additional language features for template description, for example attribute constraints and operational process declaration. Moreover, several abstraction concepts and associations between objects like classification, relationships, specialization, aggregation are supported in these languages, too, which we will ignore in this presentation.

3.2 Models In order to interpret object specifications object models are needed. Here, we briefly sketch the ingrediences of such an object model.

Life cycles: Object behaviour is described by life cycles. A life cycle (or trace) is a sequence of (sets of simultaneously occurring) events:

$$\underbrace{\overset{birth}{\overset{\downarrow}{\langle b\rangle}} \longrightarrow \overset{event\ snapshot}{\overset{\downarrow}{\left\langle \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix} \right\rangle}} \longrightarrow \cdots \longrightarrow \langle e_k\rangle}_{Trace\ \pi} \longrightarrow \overset{death}{\overset{\downarrow}{\langle d\rangle}}$$

An event snapshot is a set of simultaneous occurrences of events. Thereby, this way of describing object behaviour can be applied to single objects as well as to composed (aggregated) objects.

Time-varying states: Due to the temporal evolution of an object its attributes values are evolving as well. This can be described by a sequence of states corresponding to the life cycle:

$$\xrightarrow{birth} [S_1] \xrightarrow{events} [S_2] \xrightarrow{events} \cdots \xrightarrow{events} [S_n] \xrightarrow{death}$$

Observation structure: An observation structure is a mapping from states to attribute-value-pairs.

Object embedding: In order to build large systems, objects must be composed forming larger objects. This aggregation of communicating objects can be formally understood as object embedding.

Now, we can formally describe an object model as follows:

> An *Object Model* $\mathbf{ob} = (ACT, B, ATT, obs)$ consists of a set of actions ACT, an object behavior B over ACT, a set of attributes ATT, and an observation structure obs.

It is necessary to make some general assumptions about objects (i.e., to state some frame rules). For instance, only the occurrence of events may alter an observation. This appears to be reasonable because we expect attribute values to be changed only by explicit manipulation.

3.3 Logics As already indicated before, temporal logic is an adequate means for logical specification of object behaviour. Here, we present the use of first-order temporal logic by giving typical examples.

Temporal integrity constraints: The integrity constraint *"Value of* `Counter` *never decreases."* could be expressed in temporal logic as follows:

$$\forall x(\mathbf{always}(\ (\texttt{Counter} = x) \Rightarrow (\mathbf{next}(\texttt{Counter} \geq x))\ ))$$

Effect of events: The effect of event `Increase(n)` on an attribute `Counter` can be described by the following temporal logic formula:

$$\forall x \forall n(\mathbf{always}(\ ((\texttt{Counter} = x) \wedge \mathbf{occurs}(\texttt{Increase}(n))) \Rightarrow (\mathbf{next}(\texttt{Counter} = x + n))\ ))$$

Permission conditions for events: Assuming that `Decrease(n)` is only permitted to occur if the value of `Counter` is greater than `n`:

$$\forall n(\mathbf{always}(\mathbf{occurs}(\texttt{Decrease}(n)) \Rightarrow (\texttt{Counter} \geq n)))$$

Event synchronization ('event calling'): Expressing the event calling a_1>>a_2 (the occurrence of a_1 causes the simultaneous occurrence of a_2) in temporal logic yields the following formula:

$$\mathbf{always}(\mathbf{occurs}(a_1) \Rightarrow \mathbf{occurs}(a_2))$$

Of course, we can not give a complete presentation of temporal logic used for object specification at this place. More detailed presentations and discussions can be found for instance in [FM91].

3.4 Limitations of Object Specification The object specification concepts presented so far lead to certain limitations in the presence of object evolution. All specification parts restrict the object behaviour for the complete life span of an object. Conditional rules can be used, however, to model changing object behaviour. This shows us the following observation: languages like TROLL or Oblog are expressive enough to model even changing object behaviour depending on state changes, *but these modifications have to be fixed during specification time, e.g. before object creation.* As argued before, this is too restrictive for handling object evolution in information systems. Therefore, an extension of the framework underlying such object specification languages is needed to support the aspect of evolution in an adequate way.

4 Comparison of Object and Agent Approach

Up to now, we have looked at the state of the art in object specification and have listed on the other hand some requirements for designing artificial agents as software units. In the sequel we compare these two design paradigms and identify similarities as well as principal differences. This comparison will enable us to sketch the research road towards an evolutionary extension of object technology towards agent modelling. The comparison is divided into two main sections, a first to compare fundamentals concepts and design principles, and a second one for higher modelling principles. Both parts are summarized in a table. We will present the comparison following the order given by the table.

4.1 Basic Concepts

A summary of the concepts discussed in this subsection is shown in Table 1.

- Both objects and agents have an *internal state* based on their history and influencing their behaviour. For objects, this state is determined by the values of attributes. Agents on the other hand have a more general notion of state: it may contain disjunctive information, partial knowledge, default assumptions etc. Besides these extensions of the determined state concept of agents, an agent state may contain explicit reasoning rules.
- The *constraints* for object states and behaviour are fixed at specification time. For agents, the constraints may vary from time to time, and may be violated as result of non-normative behaviour.
- For *behaviour* description, we have a similar situation as for constraints.
- An *action* (or method) of an object corresponds to a fixed computed function whereas an agent's action may be more general, for example a problem solving step using heuristics, default knowledge, etc.
- *Goals* to be satisfied by an object are fixed at specification time, and may serve as formal requirements for implementing behaviour. Therefore they have to be logically consistent. The goals of agents may vary during an agent's lifetime, they may be conflicting or may not at all unsatisfiable.
- The *communication structure* for objects is fixed at specification time; for agents it may vary and can be influenced by actions.
- *Identity* and *encapsulation* are important concepts for both objects and agents.
- As *quality criterion* for object implementation we have correctness w.r.t. the specification. For agents, we have to look at a much broader spectrum of properties: a preference ordering of alternative "correct" behaviours, normative versus non-normative behaviours, minimization of the number of goal conflicts, maximal number of satisfied goals, etc.

Basic Concepts		
	Object Design	Agent Design
Internal State	determined by history, value-based	... + disjunctive information + partial + default + ...
Constraints	fixed	varying
Behaviour	fixed	varying
Actions	computed function	problem solving step
Goals	fixed	varying, conflicting
Communication Channels	fixed	varying
Identity	√	√
Encapsulation	√	√
Quality criterion	Correctness	Preferred Behaviour, Normative B., Minimal Number of Conflicts

Table 1. Object versus Agent Design: Basic Concepts

4.2 Modelling Constructs

In general, most of the modelling constructs known for objects are useful for agents, too (see Table 2). We will not go through the list in detail, but state some interesting differences only:

- In general, the agent framework sees all modelling constructs as something which is varying in time and may be modified by agents actions. Not all object design frameworks emphasize such dynamics.
- *Type systems* are seen as very important for object design formalizing for example inheritance. Agents may have different interfaces to different agents and they even may change their interfaces during their existence. Usual static type systems are in conflict with these requirements. Moreover, *inheritance* between agents is discussed on the level of knowledge rather than on the formal interface level.
- The concept of *role* seems to be very basic for agent design. It is connected with such concepts like authority, delegation, power to do something, etc. Even if roles were proposed for objects, too [WD92], it seems to us that the support of roles played by agents is a fundamental new modelling concept for agent design.
- *Aggregation* and *composition* are very fundamental concepts of object frameworks. For agents we may need other types of aggregation: Can a group of agents itself being considered as an (composed) agent? If so, what are the basic composition principles? For object design, we use the basic data structure ideas of lists, tuples, etc. for composition, and it is questionable whether this may be appropriate for agents, too.
- With regard to *modularization* it is not clear whether the object-oriented concepts (classes and class hierarchies) are sufficient in the context of agents. Here, more work has to be done to clarify this item.
- In object-oriented approaches *relationships* are fixed by explicitly naming classes such that only objects of these classes may participate. If we use re-

lationships for modelling communication structures between agents, we need a more flexible concept of relationships because communication structures may vary for agents.

Modelling Constructs		
	Object Design	Agent Design
Classification	$\surd$	$\surd$
Specialization & Inheritance	$\surd$	$\surd$
Type System	$\surd$	???
Roles	($\surd$) ?	$\surd$!
Aggregation / Composition	$\surd$	($\surd$) ?
Local versus Global / Modularization	$\surd$	?
Explicit Relationships	fixed	varying

Table 2. Object versus Agent Design: Modelling Constructs

5 Towards Agent Specification

In this section we propose an extension of current object specification technology in such a way that concepts which have been identified as indispensable for agents can be integrated. Instead of inventing a new specification language, we aim at a sound extension of existing object-oriented specification languages like TROLL (cf. Section 3).

In current object specification approaches object states are given by a mapping from attributes to values. From our point of view, such value maps are not flexible enough for capturing further requirements of agents. Instead of a value map it seems to be more adequate to describe the state of an agent by a set of formulas, i.e. a state is given by a logical theory (this point of view has already been used in e.g. [Rei84]). In addition, state changes have to be considered as theory revision, i.e. adding or removing of formulas.

Taking logical theories for states we do not loose any possibility of current object specification. This is due to the fact that value maps for attributes can easily be expressed in terms of standard logic.

Moreover the theory approach to states allows us to introduce new agent-specific concepts in a rather straightforward way. Beside the current values of attributes it is now possible to explicitly describe the agent's current knowledge (as logical rules). Furthermore we may now change the behaviour of an agent during its lifetime by adding or removing formulas from its current "state".

Example: A possible specification of an `Account` agent class is depicted in Fig. 1. The specification language used in this example can be considered as an extension of the object-oriented language TROLL sketched in Section 3. Similar to objects, agents have attributes (here: `Balance`) and events (e.g. `Withdraw`). The part of the behaviour specification which must not be changed is specified in the **rigid axioms** section. In our example the effect of the events `Withdraw` and `Deposit` on the attribute `Balance` is fixed.

```
agent class Account
  identification    ByAccNo:         AccNo;
  attributes        AccNo:           nat constant;
                    Balance:         money initialized 0.00;
  events            Open(H:|Holder|, AN: nat) birth;
                    Withdraw(W:money);
                    Deposit(D:money);
                    Alarm(S:string);
  mutators          AddAxioms(P:setOfAxioms);
                    ResetAxioms;
  rigid axioms      [Withdraw(W)]    Balance := Balance - W;
                    [Deposit(D)]     Balance := Balance + D;
  axiom attributes  ControlAxioms    initialized {true};
  dynamic specification
                    [AddAxioms(P)]   ControlAxioms := ControlAxioms ∪ P;
                    [ResetAxioms]    ControlAxioms := {true};
end agent class Account;
```

Fig. 1. Specification of an `Account` agent class.

In addition to the concepts used for objects, agents may have **axiom attributes**, i.e., attributes that have sets of axioms as values. For instance, Account agents have an axiom attribute `ControlAxioms` which is initialized by the set containing the formula `true`. Because there is only one axiom attribute specified, this contains the set of currently valid axioms for the agent. In case we specify several axiom attributes we have to explicitly mark one of them for that purpose. Each formula which is included in the value of this special axiom attribute at a certain state must be fulfilled in that state.

The values of axiom attributes may be manipulated by a special kind of events, called **mutators**. For our Account agents we have introduced two mutators: `AddAxioms` and `ResetAxioms`. The effect of mutators is described in the **dynamic specification** section. Here, we restrict the manipulation of the axiom attribute `ControlAxioms` to adding of further axioms and to resetting to the initial state. Please note that adding axioms means to restrict the possible behaviour of the specified agent. The mutator `ResetAxioms` removes all these additional restrictions.

In Fig. 2 some possible values for the parameter `P` of the mutator `AddAxioms` are listed. These values are sets of formulas written in the syntax of our specification language. The first value contains a formula which requires to trigger an alarm if the amount of a withdrawal is larger than 300. In the next value there is an additional restriction saying that a `Withdraw` event may only occur with an amount smaller than the current value of the attribute `Balance`. Thereby, overdrawing of an account is ruled out. The third listed value contains a formula which specifies the following behaviour: in case that a second withdrawal is tried within one day an alarm must be triggered (in this formula we refer to a `Clock` assuming that it is specified elsewhere as a part of same system).

```
{ interaction
    { W > 300 } Withdraw(W) >> Alarm("Withdrawal limit exceeded!"); }

{ permission    { W ≤ Balance } Withdraw(W); }

{ interaction
    { not(occurs(Clock.NextDay)) since last occurs(Withdraw(W)) }
      Withdraw(W) >> Alarm("Two withdrawals within one day!");     }
```

Fig. 2. Examples for possible values for P in `AddAxioms(P)`.

In the classical object-oriented view temporal evolution is only seen as value changes of attributes. In this way we can only manipulate attribute values. The behaviour of an object is fixed (*at specification time*). For systems which are intended to run over a long time this does not seem to be adequate.

In addition to the change of attribute values the theory approach to states facilitates another view to temporal evolution of a system. The change of theories (instead of attribute values) can be seen as knowledge revision and knowledge acquisition. Default knowledge may be overwritten (*at run-time*) by more specific knowledge, partial knowledge may be completed.

The idea of "states as theories" for object specification with evolving behaviour is described in more detail in [SSS95] where other aspects like evolving knowledge of an object or agent are not considered.

In order to support this more powerful approach of states as theories, we need adequate logics for dealing with the new possibilities. Obviously, classical temporal logic used in current object specification approaches is not sufficient.

Logics are needed which cope with belief revision, knowledge evolution and revision, change of defaults (default knowledge), and normative behaviour.

A large number of non-standard logics already exists which are tailored for these purposes, e.g. [Mey92, Rya93, JS93]. Here, we can choose logics having those properties we are interested in. To come as close as possible to the idea of "agent" presented before, choosing one or two logics is not enough. They must be integrated into one logical framework, into a logic of agents.

Especially, we think that a temporal logic for specifying temporal evolution, a logic of belief or knowledge, a default logic, and a deontic logic for specifying normative and non-normative behaviour could be a useful combination. First results already show that the composition of different logics can really work [FM91]. However the work in this area is still at the beginning.

6 Conclusions and Outlook

In this paper we have presented a framework for agent design. After describing some important requirements for modelling of agents, we took a brief look at the current state of object specification technology. For that, we reviewed the basics of object specification (in the context of the object-oriented specification language TROLL), models, and logics. Then we compared the two approaches object specification and agent design. We pointed out that on the one hand both approaches have many similarities, but on the other hand there also exist some serious differences between these two approaches. Especially, we show that the concept of agent overcome the limitations of current object models to describe object behaviour evolution. The main reason for that is due to the fact that the agent paradigm allows agents to have changing goals, behaviour, constraints and communication structures.

Further, we present first ideas of extending current object specification framework towards agent specification. We think that this extension can be done in several smaller steps, integrating one agent-specific concept after the other. In fact, a first step of extending an object specification approach to capture temporal evolution of object behaviour is described in [SSS95]. In this approach the idea of *states as theories* is used. The next steps will be the integration of concepts like normative behaviour specification (i.e. some kind of deontic logic) and knowledge and goal reasoning capabilities.

We do not want to conceal that there are several properties of agents for which we do not know at the moment how to put them into an adequate logic. For instance, current object-oriented approaches in general use a fixed communication structure between objects which can easily be represented in a logic. But we want to allow a more flexible communication between agents. Furthermore, we would like to distinguish different states of activities for agents. For that we have to express that an agent reacts only on demand or is active in a sense that it initiates actions in order to fulfill its goals.

Although there are many open question, it is obvious that the concept of *agent* may be useful for modelling federated information systems in which the single components are assumed to be autonomous. In order to describe such components in an adequate way the concepts of current object-oriented approaches are often not sufficient. We are aware of the fact that the ideas presented in this paper are in a preliminary state. The presentation aims at showing current research efforts rather than stable results.

References

[BG88] A. H. Bond, L. Gasser, eds. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1988.

[BLR93] S. Brass, U. W. Lipeck, P. Resende. Specification of Object Behaviour with Defaults. In U. W. Lipeck, G. Koschorreck, eds., *Proc. Int. Workshop on Information Systems – Correctness and Reusability (IS-CORE'93), Tech. Rep. 01/93, University of Hanover*, pp. 155–177, 1993.

[Bro92] M. L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Systems. In P. M. Gray, R. J. Lucas, eds., *Advanced Database Systems (Proc. BNCOD-10)*, pp. 1–28. Springer LNCS 618, 1992.

[Buc90] A. P. Buchmann. Modeling Heterogeneous Systems as an Active Object Space. In A. Dearle, G. M. Shaw, S. B. Zdonik, eds., *Proc. 4th Int. Workshop on Persistent Object Systems*, pp. 279–290. Morgan Kaufmann, 1990.

[DDP93] E. Dubois, P. Du Bois, M. Petit. O-O Requirements Analysis: An Agent Perspective. In O. Nierstrasz, ed., *ECOOP'93 – Object-Oriented Programming*, pp. 458–481. Springer LNCS 707, 1993.

[DT91] R. Dieng, P. A. Tourtier. A Composite System for Knowledge Acquisition and User Assistance. In *Proc. AAAI Spring Symp. Design of Composite Systems*, 1991.

[FM91] J. Fiadeiro, T. Maibaum. Temporal Reasoning over Deontic Specifications. *Journal of Logic and Computation*, 1(3):357–395, 1991.

[GK94] M. R. Genesereth, S. P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, 1994.

[HJS92] T. Hartmann, R. Jungclaus, and G. Saake. Aggregation in a Behavior Oriented Object Model. In O. Lehrmann Madsen, editor, *Proc. of the European Conf. on Object-Oriented Programming (ECOOP'92)*, pages 57–77. Springer-Verlag, LNCS 615, Berlin, 1992.

[HSJ+94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, J. Kusch. Revised Version of the Modelling Language TROLL (Version 2.0). Report 94–03, TU Braunschweig, 1994.

[JS93] A. Jones, M. Sergot. On the Characterisation of Law and Computer Systems: The Normative System Perspective. In J.-J. Ch. Meyer, R. J. Wieringa, eds., *Deontic Logic in Computer Science: Normative System Specification*, chapter 12. Wiley, 1993.

[JSHS95] R. Jungclaus, G. Saake, T. Hartmann, C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1995. *To appear.*

[Mey92] J.-J. Ch. Meyer. Modal Logics for Knowledge Representation. In R. P. van de Riet, R. A. Meersman, eds., *Linguistic Instruments in Knowledge Engineering*, pp. 251–275. North-Holland, 1992.

[MP92] Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems. Vol. 1: Specification*. Springer, 1992.

[Rei84] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopoulos, J. W. Schmidt, eds., *On Conceptual Modeling*, pp. 191–239. Springer, 1984.

[RK91] E. Rich, K. Knight. *Artificial Intelligence*. McGraw-Hill, 1991.

[Rya93] M. Ryan. Defaults in Specifications. In A. Finkelstein, ed., *Proc. of the IEEE Conf. on Requirements Engineering (San Diego)*, 1993.

[SJH93] G. Saake, R. Jungclaus, T. Hartmann. Application Modelling in Heterogeneous Environments Using an Object Specification Language. *Int. Journal of Intelligent and Cooperative Information Systems*, 2(4):425–449, 1993.

[SSC92] A. Sernadas, C. Sernadas, J. F. Costa. Object Specification Logic. Internal Report, INESC, University of Lisbon, 1992.

[SSE87] A. Sernadas, C. Sernadas, H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P. M. Stocker, W. Kent, eds., *Proc. VLDB'87*, pp. 107–116. Morgan Kaufmann, 1987.

[SSS95] G. Saake, A. Sernadas, C. Sernadas. Evolving Object Specifications. In R. Wieringa, R. Feenstra, eds., *Information Systems - Correctness and Reusability. Selected Papers.* World Scientific Publishing, 1995.

[WD92] R. Wieringa, W. De Jonge. The Identification of Objects and Roles – Object Identifiers Revisited. Report IR–267 (Vers. 2), VU Amsterdam, 1992.