Appeared in: Ras, Z. W. and Michalewicz, M., editors, Foundations of Intelligent Systems, Proc. of the 9th Int. Symposium on Methodologies for Intelligent Systems, ISMIS'96, Zakopane, Poland, LNAI 1079, Springer-Verlag, Berlin, pages 572–581, June 1996.

Dynamically Changing Behavior: An Agent-Oriented View to Modeling Intelligent Information Systems*

Can Türker Stefan Conrad Gunter Saake

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

Abstract. Although object specification technology is successfully used for modeling information systems, it is not able to get a grasp of dynamically changing behavior. Due to the fact that objects in information systems can have a very long life-span, it often happens that during the life of an object external requirements are changing (e.g. changes of laws or banking rules). Such changes often require the object to adopt another behavior. The main problem for current object specification approaches is that, in general, not all possible changes can be taken into account in advance at specification time. Therefore, a flexible extension is needed to capture this situation. The approach we present and discuss in this paper is an important step towards a specification framework based on the concept of agent by introducing a certain form of knowledge as part of the internal state.

1 Introduction

Nowadays, nearly every organization has to face the situation that in order be competitive the use of modern information systems is indispensable. Considering the frequent and dramatic changes in the international economy and politics, there is a clear demand for advanced information systems which are able to deal with highly dynamic environments, e.g. rapidly changing markets, increasing (world-wide) competition, and new trade agreements as well as (inter)national laws. In the recent years, there are obvious efforts in several computer science communities to build cooperative intelligent information systems which can deal with such aspects (see for example [HPS93]).

Today, object-oriented techniques are in general used for modeling such advanced information systems [Buc90, Bro92]. Most of the existing object-oriented approaches are successful in capturing the properties and behavior of the real-world entities. However, it seems that the concept of "object" (at least in its current understanding) cannot cover all aspects of modern information systems. Whereas structural aspects of such systems can easily be dealt with by current object-oriented approaches, these approaches succeed to cope with *dynamic* behavior only up to a certain degree. Typically, information system objects have

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

a longer life-span than application programs, environmental restrictions, etc. Therefore, we need a semantic model where the behavior specification of an object or object system may be modified during its existence, which is not expressible in current formalisms underlying traditional (object-oriented) specification languages until now.

Currently, the concept of *agent* [WJ95, GK94] which can be seen as a further development of "object" is emerging as an important concept in many fields of Information Technology, e.g. Federated Database Systems, Distributed Artificial Intelligence, or Requirements Engineering. Agents differ from conventional objects in several ways. First of all agents are not only reactive, but they are also active in the sense that they are able to act goal-driven without external stimuli. In addition, agents are flexible, e.g. they may change their behavior dynamically during system run-time. Furthermore, they are able to deal with unexpected, unpredictable situations. The properties mentioned above are especially important for modeling human behavior in an appropriate way. In summary, the behavior of an agent is not (or can not be) completely determined at compile or specification time. In order to get a grasp of such properties, we need an agent specification framework which goes beyond the existing object-oriented ones.

The paper is structured as follows. Sect. 2 surveys the current object modeling and specification technology. In Sect. 3, we briefly discuss a concept of agent as a further evolution of the concept of object. Here, we illustrate the properties of agents and point out differences to the traditional concept of object. In Sect. 4, we present a first way of extending object specification towards agent specification. Here, we focus our attention especially on the specification of dynamically changing behavior of agents. Finally, we conclude by pointing out future work.

2 Object-Oriented Modeling and Specification

Recently, object-oriented conceptual modeling of information systems has become a widely accepted approach. Meanwhile, there exists a lot of object-oriented models and specification languages (e.g. Oblog [SSG+91], or TROLL [HSJ+94, JSHS96, SJH93]) proposed for that purposes. In this section, we briefly recall the basic ideas of the concept of object, whereby we base our presentation on the object model as introduced in [SSE87].

Basically, objects are characterized as *coherent units of structure and behavior*. An object has an *internal state* of which certain properties can be observed. The internal state can be manipulated explicitly through a properly defined *event interface*. *Attributes* are the observable properties of objects which may only be changed by event occurrences. The *behavior* of objects are described by *life cycles*, which are built from sequences of (sets of simultaneously occurring) *events*. Thus, each object state is completely characterized by a *life cycle prefix*, which determines the current attribute values. The possible evolution of objects can be restricted by a set of *state constraints* which can be used to define the admissible state transitions for an object.

For textual presentation of object specifications, we use a notation close to the syntactical conventions of the object-oriented specification language TROLL. In Fig. 1 we give a TROLL specification for a class of account objects. For the

purposes of this paper, we have only specified a small number of properties an account could have. Here, we assume an account to have an (unique) account number, a bank by which it is managed, a holder, a balance, and a limit for overdrawing. Moreover, we specify some basic events like opening an account, withdrawing money from or depositing money to an account.

```
object class Account
  identification    ByAccountID: (Bank, No);
  attributes        No:           nat constant;
                    Bank:         |Bank|;
                    Holder:       |Customer|;
                    Balance:      money initialized 0.00;
                    Limit:        money initialized 0.00 restricted >= -5000.00;
                    Counter:      nat initialized 0;
  events  Open(BID:|Bank|, AccNo:nat, AccHolder:|Customer|) birth
                              changing   Bank    := BID,
                                         No      := AccNo,
                                         Holder  := AccHolder;
          Withdraw(W:money)   enabled    Balance - Limit >= W;
                              changing   Balance := Balance - W;
                              calling    IncreaseCounter;
          Deposit(D:money)    changing   Balance := Balance + D;
                              calling    IncreaseCounter;
          IncreaseCounter     changing   Counter := Counter + 1;
          Close death;
end object class Account;
```

Fig. 1. TROLL specification of an `Account` class.

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 behavior 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 a few basic specification patterns.

In the declaration section for events, we mark some events as *birth* events or as *death* events corresponding to creation and destruction of objects, e.g. `Open` and `Close` in our example in Fig. 1. The occurrence of events can be restricted by *enabling conditions* which define explicit preconditions for events, e.g. the event `Withdraw` may only occur if the the withdrawal amount is less than the balance of the account. Changes of attribute values are caused by event occurrences, e.g. the event `Withdraw` decreases the balance of an account. These attribute valuation rules can also be restricted using conditions on state attributes, e.g. we may restrict the the credit limit to maximal 5000.00. The *event calling mechanism* is used to model interaction inside (composite) objects, e.g. the event `Withdraw` enforces the event `IncreaseCounter` to occur simultaneously. Similar to attribute valuations, we may have conditional event calling, too.

Of course, TROLL-like languages offer additional language features for template description, for example operational process declaration and global behavior constraints. Moreover, further abstraction concepts and associations between objects like relationships, specialization, aggregation are supported, too. Never-

theless, the concepts presented so far have a major drawback: *they succeed in capturing dynamic behavior (of information systems) only up to a certain degree.* Indeed, languages like TROLL or Oblog are expressive enough to model even changing object behavior depending on state changes, *but these modifications have to be fixed during specification time, e.g. before object creation.* But, this is too restrictive for handling object evolution in information systems. Typically, information system objects are characterized by long life-spans. Usually, during that long time-span an object and the environment of object may change in a way that cannot be foreseen. Consequently, dynamic specification changes are needed to overcome the problem that generally not all possible future behaviors of an object can be anticipated in the original system specification.

3 A Concept of Agent

In this section we propose the *concept of agent*, which can be seen as a further evolution of the concept of object, as an adequate means for modeling information systems. We emphasize that the term agent is frequently used in several computer science disciplines, e.g. Federated Database Systems, Distributed Artificial Intelligence, or Requirements Engineering, with slightly different meanings. Basically, agents are interpreted as *dynamic* and *intelligent* objects (cf. [Sho93, GK94, WJ95]), which are assumed to be

- *autonomous* (act without direct (user) intervention),
- *cooperative* (cooperate, coordinate and negotiate with other agents),
- *reactive* (respond flexible to dynamic and unpredictable situations that occur in their environments), and
- *pro-active* (may also exhibit goal-driven behavior by taking the initiative).

In the sequel, we briefly summarize the basic characteristics of agents and point out differences to traditional objects (for a detailed discussion see [SCT95]).

Internal State. Like objects, agents have an *internal state* which is based on their history and influence their behavior. Whereas the internal state of objects is determined by the values of their attributes, agents have a more general notion of internal (mental) state: beside (conventional) attribute values it may contain disjunctive information, partial knowledge, default assumptions, etc. Thus, the state of agents includes an internal, imperfect representation of the world. Essentially, the internal state of an agent reflects the knowledge (belief, intention, obligations, goals, etc.) of that agent at a given time. In contrast to traditional object concepts, this *knowledge is not fixed at specification time, but it is changeable during the lifetime of an agent.* Due to this fact, agents are assumed to have knowledge acquisition and revision capabilities in order to be able to adapt their knowledge to changes in their environments. In order to make use of their knowledge, the states of agents may additionally contain explicit reasoning rules.

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.* Since goals are part of the internal state of agents, they may be changed during an agent's lifetime, too. They can

be extended, revised or replaced through other goals. Further, they may also be conflicting. Hence, agents must be equipped conflict resolution strategies.

Behavior, Actions and Constraints. Agents are able to (re)act and communicate by executing sequences of *actions*. Thus, agents show an external *behavior* that obeys the given *constraints*. In contrast to traditional objects, agents may

- exhibit reactive behavior as well as goal-driven (or pro-active) behavior;
- perform actions concurrently by asynchronous message passing;
- have varying, non-normative behavior;
- have varying constraints;
- have a preference ordering of alternative "correct" behaviors.

Cooperation. In most cases agents have to cooperate to achieve their goals. Especially in federated information systems, where multiple 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].

Considering all these properties agents can have, it becomes clear that current object specification technology as sketched in Sect. 2 cannot fulfill all these requirements. This is due to the fact that several concepts are not given in current object-oriented approaches. Nevertheless, the existing object specification approaches can be used as a stable basis for extensions which try to get a grasp of those agent-specific properties. By carefully extending the underlying semantic models and logics it should be possible to come closer and closer to the idea of "agents" as sketched before. The best way for allowing such extensions seems to be to consider "states as theories" in contrast to the usual way in object-oriented approaches where the state of an object is described by a simple value map assigning each attribute a corresponding value.

The "states as theories" approach is much more powerful by assuming that a state is described by a set of formulas. Depending on the underlying logic that we apply for formulating such formulas, we can then express different kinds of knowledge, for example knowledge about the future behavior of an agent as part of its own state as well as knowledge about the states of other agents. In this way, simple state changes can become changes of theories by which we can even express the change of knowledge or goals of an agent. Thereby, knowledge revision as well as dynamic knowledge acquisition can be specified. Furthermore, partial knowledge is possible and default knowledge could be integrated. However, for such an agent specification approach we need a logical framework, a *logic of agents*, in which several non-standard logics (e.g. logic of knowledge, default logic, deontic logic [Mey92, Rya93]), can be integrated. First results already show that the composition of different logics can really work [FM91]. In [SSS95] and [CS95] first steps towards the specification of dynamically changeable behavior in an object-oriented setting are presented and discussed.

4 Specifying Information Systems in Terms of Agents

In this section, we sketch the basic frame of an agent-oriented specification language by giving example specifications. We point out that in this first approach

only a few, but very important agent-specific concepts like dynamic behavior are respected.

In Fig. 2 the structure of a possible specification of an agent class `Account` is depicted. The specification language used here can be considered as an extension of the object-oriented language TROLL sketched in Sect. 2. Similar to objects, agents have attributes (e.g. `Balance`) and events (e.g. `Withdraw`). The part of the behavior 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     ByAccountID: (Bank, No);
  attributes         No:          nat constant;
                     Bank:        |Bank|;
                     Holder:      |Customer|;
                     Balance:     money initialized 0.00;
                     Limit:       money initialized 0.00;
                     Counter:     nat initialized 0;
  events             Open(BID:|Bank|, AccNo:nat, AccHolder:|Customer|) birth;
                     Withdrawal(W:money);
                     Deposit(D:money);
                     IncreaseCounter;
                     Close death;
                     Warning(S:string);
  rigid axioms       Open(BID:|Bank|, AccNo:nat, AccHolder:|Customer|)
                                      changing   Bank   := BID,
                                                 No     := AccNo,
                                                 Holder := AccHolder;
                                      calling    ResetAxioms;
                     Withdraw(W)      enabled    Balance - Limit >= W;
                                      changing   Balance := Balance - W;
                                      calling    IncreaseCounter;
                     Deposit(D)       changing   Balance := Balance + D;
                                      calling    IncreaseCounter;
                     IncreaseCounter  changing   Counter := Counter + 1;
  axiom attributes       Axioms initialized {};
  mutators               ResetAxioms;
                         AddAxioms(P:setOfAxioms);
                         RemoveAxioms(P:setOfAxioms);
  dynamic specification  ResetAxioms     changing Axioms := {};
                         AddAxioms(P)    changing Axioms := Axioms ∪ P;
                         RemoveAxioms(P) changing Axioms := Axioms - P;
end agent class Account;
```

Fig. 2. Specification of an agent class `Account`

In addition to the traditional concept of attributes as used for objects, an agent have **axiom attributes** which contain sets of axioms which are valid under certain circumstances. In our example we have axiom attribute `Axioms` which is initialized by the empty set of axioms. In case we specify several axiom attributes we have to explicitly mark one of them as current axiom set. Each formula which is included in the value of this special axiom attribute at a certain state must be fulfilled in that state. Similar to basic attributes, axiom attributes are changed by

mutators which can be seen as special events. The effect of mutators is described in the **dynamic specification** section. Here, we allow the manipulation of the axiom attribute `Axioms`. We may add further axioms to `Axioms`, remove existing axioms from `Axioms` and reset `Axioms` to the initial state.

Specification of Dynamic Behavior

As already mentioned, one main difference between agents and traditional objects is that agents may change their behavior dynamically during their lifetime. In the sequel, we show different ways how dynamic behavior can be specified.

1. By using only one dynamically changeable axiom attribute. This case is presented in the example in Fig. 2. Here, the axiom attribute must be modifiable during the lifetime of an agent in order to be able to represent changing dynamic behavior of that agent. In our example the axiom attribute `Axioms` can be manipulated by the mutators `AddAxioms`, `RemoveAxioms` and `ResetAxioms`. Whereas `AddAxioms` and `RemoveAxioms` adds further axioms to and removes existing axioms from `Axioms`, respectively, `ResetAxioms` resets `Axioms` to the initial state. Possible values for the parameter `P` of the mutator `AddAxioms` could be the following ones:

```
{ Withdraw(W) calling {W > 400.00} Warning("Withdrawal limit exceeded!"); }

{ Withdraw(W) enabled (W >= 0.00) and (Balance - W >= Limit); }

{ Withdraw(W) calling {not(occurs(Clock.NextDay)) since last
                       occurs(Withdraw(W))}
                       Warning("Two withdrawals within one day!"); }

{ Close       enabled Balance = 0.00; }
```

The values above are sets of axioms written in the syntax of our specification language. The first value contains an axiom which requires to trigger a warning if the amount of a withdrawal is larger than 400. 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` minus the current value of the attribute `Limit`. Thereby, overdrawing of an account is ruled out. The third value ensures that a warning is triggered if two withdrawals occur within one day (in this formula we refer to a `Clock` assuming that it is specified elsewhere as a part of the same system). The last value contains a formula which specifies that an account may only be closed if there is no money on this account.

2. By using a set of predefined, unchangeable axiom attributes. Here, a set of axiom attributes, which contain predefined sets of axioms and which can not be modified during the lifetime of an agent, can be defined to model dynamically changing behavior of an agent. One of these axiom attributes must be declared as the current valid set of axioms which determines the current behavior of the agent. By switching between the axiom attributes the behavior of the agent can be changed dynamically.

```
axiom attributes
  Axioms(N:nat) initialized
    N=0: {} default,
    N=1: {Withdraw(W) calling {W > Balance}
                              Warning("Account has been overdrawn"))},
    N=2: {Withdraw(W) calling {not(occurs(Clock.NextDay))
                              since last occurs(Withdraw(W))}
                              Warning("Two withdrawals within one day!");}
    ...;
mutators              SwitchAxioms(N:nat);
dynamic specification SwitchAxioms(N) changing Axioms(0) := Axioms(N);
```

In the example above we define a parameterized attribute `Axioms` (for details see [HSJ+94]) which contains different sets of axioms. Here, we declare implicitly the attribute term `Axioms(0)` to be the set with the current valid axioms. By using the mutator `SwitchAxioms` we are able to change the agent's behavior dynamically. Please notice that this approach restricts the behavior evolution of an agent to various predefined behavior pattern. This is due to the fact that the axioms sets can not be modified during the lifetime of an agent. Furthermore, note that in the rigid axioms part the common behavior of all possible behaviors are specified.

3. By using several dynamically changeable axiom attributes. Here, the ideas of the other cases are combined. We allow to specify several axiom attributes which may be modified during the lifetime of an agent. As in the second case, these attributes may be predefined and one of these attributes is marked as the currently valid one. In the following example we have specified two mutators `AddAxioms` and `RemoveAxioms` (in addition to the mutator of the example above) for adding a set of axioms to and for removing a set of axioms from a given axiom attribute, respectively.

```
mutators              ...
                      AddAxioms(N:nat, P:setOfAxioms);
                      RemoveAxioms(N:nat, P:setOfAxioms);
dynamic specification ...
                      AddAxioms(N,P)    changing Axioms(N):= Axioms(N) ∪ P;
                      RemoveAxioms(N,P) changing Axioms(N):= Axioms(N) - P;
```

We emphasize that it might be useful to combine changing as well as predefined, unchangeable axiom attributes. In such cases we have to specify for each changeable axiom attribute own mutators. Further, please note that mutator events may be equipped with enabling conditions as usual events in order to prevent arbitrary manipulations. Moreover, mutator events may also cause the occurrence of other basic as well as mutator events. This fact can be expressed by using the well-known event calling mechanism.

Representation of Goals

In classical object specifications goals are represented implicitly by the behavior specification. In case of agent specifications goals can be expressed in a similar

way. Fixed goals can be specified in terms of rigid axioms, whereas changing goals can be represented by axiom attributes.

In order to achieve its goals, an agent can execute different sequences of actions. Suppose, we have a (bank) customer who wants to invest a given amount of money such that his profit is maximal. In general there are several possible strategies to do this. The customer can transfer money from a standard account with a low rate of interest to a saving account with a higher rate of interest, or he can buy stocks or immovables, etc. In order to represent different strategies (which are changeable), we need several axiom attributes.

5 Conclusions

In this paper we have presented a first approach of a framework for modeling information systems in terms of agents. After reviewing the basics of object specification technology (in the context of the object-oriented specification language TROLL), we sketched the concept of an agent as a further evolution of the traditional concept of object. Here, we showed that the concept of agent overcomes the limitations of current object models to describe object behavior evolution. This is due to the fact that the agent paradigm allows agents to have changing goals, behavior, constraints, etc.

Our approach bases on the idea of "states as theories" as described, for instance, in [SSS95]. We proposed a two-level specification framework. The first level contains basic axioms describing usual events and their fixed effects on the specified attributes. In the second level we allow to specify (meta) axioms which describe the possible evolution of the agent specification. Thereby, we are able to consider dynamically changing behavior of agents and agent systems.

We do not want to conceal that there are several properties of agents of which we do not know at the moment how to integrate them into the framework we proposed, for example planning and conflict resolving facilities of agents, or autonomy issues (e.g. which request must be fulfilled by an agent). Besides, we have to investigate how far we can allow dynamic *signature modification*. In order to model evolutionary behavior adequately, it is necessary to allow the dynamic specification of additional events. If we allow arbitrary formulas as parameters for the mutators, it is easy to add new events into the specification during the lifetime of an agent. When defining such events we also may need the specification of additional mutators which describe the evolution of these events. On the other hand, if we do not allow arbitrary formulas as parameters, only the behavior of existing events may be changed and thus we have a restricted evolution of agents. Furthermore, we have to check if we need additionally attributes which may be integrated into the specification during the lifetime of an agent. Further, we have to find out which of the agent-specific concepts presented so far can be assumed to be implicit for each agent specification? Should we assume that a given set of axiom attributes and mutators are implicitly predefined, e.g. an axiom attribute `Axioms` and the mutators `ResetAxioms`, `AddAxioms` and `RemoveAxioms` as described in Fig. 2 in Sect. 4?

In conclusion, we can state that although there are many open questions, it is obvious that the concept of *agent* can be useful especially for modeling federated

information systems in which the single components are assumed to be partially autonomous.

References

[Bro92] M. L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Systems. In *Advanced Database Systems: Proc. 10th British National Conf. on Databases*, pp. 1–28. Springer, 1992.

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

[CS95] S. Conrad and G. Saake. Evolving Temporal Behaviour in Information Systems. In *HOA'95 — Higher-Order Algebra, Logic, and Term Rewriting (2nd Int. Workshop)*, pp. PP7:1–16. Participant's Proceedings, 1995.

[FM91] J. Fiadeiro and T. Maibaum. Towards Object Calculi. In G. Saake and A. Sernadas, eds., *Information Systems – Correctness and Reusability*, pp. 129–178. TU Braunschweig, Informatik Bericht 91-03, 1991.

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

[HPS93] M. Huhns, M. P. Papazoglou, and G. Schlageter, eds., *Proc. Int. Conf. Intelligent and Cooperating Information Systems*. IEEE Computer Society, 1993.

[HSJ+94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch. Revised Version of the Modelling Language TROLL (Version 2.0). Informatik-Bericht 94–03, Technische Universität Braunschweig, 1994.

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1996. *To appear.*

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

[Rya93] M. Ryan. Defaults in Specifications. In *Proc. Int. Symposium on Requirements Engineering*, pp. 142–149. IEEE Computer Society, 1993.

[SCT95] G. Saake, S. Conrad, and C. Türker. From Object Specification towards Agent Design. In *OOER'95 — Proc. 14th Int. Conf. on Object-Oriented and Entity-Relationship Modeling*, pp. 329–340. Springer, 1995.

[Sho93] Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, March 1993.

[SJH93] G. Saake, R. Jungclaus, and 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.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In *Proc. 13th Int. Conf. on Very Large Data Bases*, pp. 107–116. Morgan Kaufmann Publishers, 1987.

[SSG+91] A. Sernadas, C. Sernadas, P. Gouveia, P. Resende, and J. Gouveia. OBLOG — Object-Oriented Logic: An Informal Introduction. Technical Report, INESC, Lisbon, 1991.

[SSS95] G. Saake, A. Sernadas, and C. Sernadas. Evolving Object Specifications. In *Information Systems - Correctness and Reusability*, pp. 84–99. World Scientific Publishing, 1995.

[WJ95] M. J. Wooldridge and N. R. Jennings. Agents Theories, Architectures, and Languages: A Survey. In *Intelligent Agents — Proc. ECAI'94 Workshop on Agent Theories, Architectures, and Languages*, pp. 1–39. Springer, 1995.