Towards an Agent-Oriented Framework for Specification of Information Systems*

Stefan Conrad Gunter Saake Can Türker

Otto-von-Guericke-Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {conrad|saake|tuerker}@iti.cs.uni-magdeburg.de
Phone: ++49-391-67-18066/18800/12994 Fax: ++49-391-67-12020

Abstract

Objects in information systems usually have a very long life-span. Therefore, it often happens that during the life of an object external requirements are changing, e.g. changes of laws. Such changes often require the object to adopt another behavior. In consequence, it is necessary to get a grasp of dynamically changing object behavior. Unfortunately, not all possible changes can in general be taken into account in advance at specification time. Hence, current object specification approaches cannot deal with this problem. Flexible extensions of object specification are needed to capture such situations. The approach we present and discuss in this paper is an important step towards a specification framework based on the concept of agents by introducing a certain form of knowledge as part of the internal state of objects. Especially, we concentrate on the specification of evolving temporal behavior. For that, we propose an extension (called Evolving Temporal Logic) of classical temporal logic approaches to object specification.

Keywords: modeling information systems, agent-oriented specification, dynamically changing behavior, evolving temporal logic.

1 Introduction

Currently, nearly every enterprise or 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 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 [Buc91, 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

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

only up to a certain degree. Typically, information system objects have 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.

The concept of *agent* [WJ95, GK94] which can be seen as a further development of the concept of object seems to provide a more adequate basis for modeling such information system dynamics. In comparison to traditional objects, agents are flexible in that sense they may change their behavior dynamically during system run-time, i.e. 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-oriented specification framework which goes beyond the existing object-oriented ones.

Therefore, we propose and discuss several extensions for object specification languages. These extensions are intended to be first steps towards an own agent-oriented specification framework. For that, we present a first formalization based on an extended temporal logic.

The remainder of this paper is organized as follows. Section 2 starts with a brief presentation of current object specification technology for modeling information systems. Further, we introduce the concept of agent as a further evolution of the concept of object. In Section 3, we propose first extensions of existing object specification languages for capturing dynamically changing behavior. An extended temporal logic, called Evolving Temporal Logic, as formal basis is sketched in Section 4. Finally, we conclude by summarizing and pointing out future work.

2 From Object Specification to Agent Design

In the recent years, 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 [SSE87, SSG$^+$91], LCM [FW93] or TROLL [HSJ$^+$94, JSHS96, SJH93]) proposed for those 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*. Objects can be considered as *observable processes*. *Attributes* are the observable properties of objects which may only be changed by event occurrences. The *behavior* of objects are described by *life cycles (or traces)*, which are built from sequences of (sets of simultaneously occurring) events. Thus, each object state is completely characterized by a *life cycle prefix (or event snapshot)*, 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 Figure 1 we introduce an example of a TROLL specification. For the purposes of this paper, we have chosen a small universe of discourse (UoD) consisting of one or more account objects. 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.

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-

```
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;
```

Figure 1: TROLL specification of an `Account` class.

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.

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`. The occurrence of events can be restricted by *enabling conditions*, which are formulae built over attributes and event parameters. In connection with temporal quantifiers these conditions may refer to object histories. Changes of attribute values are caused by event occurrences, i.e. the event `Withdraw` decreases the balance of an account. The allowed values for object attributes may also be restricted, e.g. we may constrain the credit limit to maximal 5000.00. Interactions inside (composite) objects are expressed by the *event calling mechanism*, e.g. a withdrawal event enforces the event `IncreaseCounter` to occur simultaneously. Similar to attribute valuations, conditional event calling is supported, too.

The object specification 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 in advance. 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. In order to support the aspect of object and object system evolution, respectively, in an adequate way, we need an extended, logic-based framework where object class descriptions may be modified during system run-time.

Recently, the *concept of agent*, which can be seen as a further evolution of the concept of object (cf. [Sho93, GK94, WJ95]), is proposed as an adequate means for modeling information systems. Basically, an agent may be seen as an *intelligent* and *evolutionary* object which is equipped with knowledge and reasoning capabilities and is able to deal with dynamic aspects, e.g. to change its state as well as its behavior dynamically.

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 state: beside (conventional) attribute values it may contain disjunctive information, partial knowledge, default assumptions, etc. 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.* In conclusion, we can state that the internal state of an agent contains strict knowledge (which is fixed at creation time and may not be revised) as well as some changeable knowledge (which may be revised or replaced under given constraints during the agent evolution).

Agents have *goals* which they try to achieve (by cooperation) under given constraints. Each agent is obliged to satisfy its goals. 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 (more important) goals. In contrast, goals to be satisfied by traditional objects are fixed at specification time, and may serve as formal requirements for implementing behavior. Therefore they have to be logically consistent. On the other hand, the agent's goals may also be conflicting. Hence, agents must be able to resolve conflict situations in which not all goals may be achieved. In such cases, agents must be able either to revise some of their goals or to decide to satisfy only a few of their goals which are not conflicting.

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 exhibit reactive behavior as well as goal-driven (or pro-active) behavior. Because agents are assumed to be autonomous, they are able to act without direct (user) intervention.

In most cases agents have to cooperate to achieve their goals. Because of the fact that agents may change their behavior and/or may even change their signature, there must exist varying communication structures. For cooperation reasons agents require knowledge about other agents, i.e. their capabilities and goals, respectively. However, agents have in general not the same and complete knowledge about other agents. In such cases, agents have to deal with partial or incomplete knowledge.

Considering all these properties agents can have, it becomes clear that the current object specification technology as sketched in the beginning of this section 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. A detailed discussion on the differences between traditional object concepts and the presented concept of agent can be found in [SCT95, TCS96].

In the following section, we propose a first agent specification language in which some of the agent-specific concepts are respected. This language is an extension of an existing object-oriented specification language. Instead of inventing a completely new specification language the extension of an existing and well-understood specification language offers us the possibility to experiment on a stable and well-understood basis.

3 Towards an Agent-Oriented Specification Language

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.

Our starting point is the idea of "*considering states as theories*" (a similar approach was taken in [Rei84]). In comparison to usual 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.

```
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:Formula);
                       RemoveAxioms(P:Formula);
  dynamic specification ResetAxioms     changing Axioms := {};
                       AddAxioms(P)    changing Axioms := Axioms ∪ P;
                       RemoveAxioms(P) changing Axioms := Axioms - P;
end agent class Account;
```

Figure 2: Specification of an agent class `Account`

We propose a two-level specification framework for modeling of information systems in terms of agents. The first level contains usual attributes and events, which describe the fixed behavior of an agent. In the second level, the possible evolution of the agent specification is specified.

In Figure 2 the structure of a possible specification of an agent class `Account` is sketched. The specification language used here can be considered as an extension of the object-oriented language TROLL sketched in Section 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. In addition to the concepts used for objects, an agent have **axiom attributes** which contain sets of axioms which are valid under certain circumstances. In our example we have the 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 the 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. There are several different ways how dynamic behavior can be specified:

1. **Using only one dynamically changeable axiom attribute:** This case is presented in the example in Figure 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 listed value contains a formula which specifies that an account may only be closed if there is no money on this account.

2. **Using a set of predefined, unchangeable axiom attributes:** Here, a set of axiom attributes, which contain predefined sets of axioms and which cannot 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
  ResetAxioms;
  SwitchAxioms(N:nat);
dynamic specification
  ResetAxioms     changing Axioms(0) := {};
  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. **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.

However, for the agent specification approach presented so far 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, Rya94, JS93]), 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. The following section gives a first formalization of dynamically changing behavior based on an extended temporal logic.

4 Evolving Temporal Logic

In this section we present the basic ideas for formalizing an extension of temporal logic we need for capturing the properties sketched in the previous section. We will call this extension *Evolving Temporal Logic* (ETL). Afterwards, we show how the example given in the previous section is formulated in ETL.

4.1 Basic Ideas for Formalization

Temporal Logic. The starting point is a first-order, discrete, future-directed linear temporal logic for objects which can be considered as a slightly modified version of the *Object Specification Logic (OSL)* which is presented in full detail in [SSC92]. In [Jun93] a comprehensive translation of Troll object specifications into OSL is given. The following basic types of elementary propositions are used in the logic:

1. $o.\mathrm{Attr} = v$ expresses that the attribute Attr of an object o has the value v (we have adopted this form from the specification language used for our example; instead we could also take a predicate expression like $\mathrm{Attr}(o, v)$).

2. $o.\nabla e$ stands for the occurrence of event e in object o.

With these elementary propositions we may build formulas in the usual way: for this we may use for instance the boolean operators $\neg$ (negation) and $\wedge$ (conjunction) as well as all operators which can be defined by these ones. Furthermore, we have the future-directed temporal operators $\bigcirc$ (next), $\Box$ (always in the future), and $\Diamond$ (sometime in the future; defined as $\Diamond f \equiv \neg\Box\neg f$). By introducing variables and quantifiers we obtain a first-order variant of linear temporal logic: provided x is a variable and f a formula, then $\forall x : f$ and $\exists x : f$ are formulas.

The semantics of temporal logic formulas is defined w.r.t. life cycles which are infinite sequences of states: $\lambda = \langle s_0, s_1, s_2, \ldots \rangle$. We define λ^i as the life cycle which is obtained by removing the first i states from λ, i.e. $\lambda^i = \langle s_i, s_{i+1}, s_{i+2}, \ldots \rangle$. Each state in a life cycle is assumed to be equipped with a mapping assigning a truth value to each elementary proposition. Based on that we can define the semantics of composed formulas in the usual way. For instance, the semantics of temporal operators is defined as follows ($\lambda \models \phi$ means that ϕ is satisfied in λ):

$\lambda \models \Box f$ if for all $i \geq 0$: $\lambda^i \models f$.
$\lambda \models \bigcirc f$ if $\lambda^1 \models f$.

For brevity we omit the treatment of variables. This can be done in the usual straightforward way. All variables which are not explicitly bound by a quantifier are assumed to be universally quantified. Fully-fledged definitions of syntax and semantics of first-order order-sorted temporal logics for object specification can be found for instance in [SSC92] or [Con96].

Example. Here, we only present some temporal logic formulas representing properties of the objects described in Fig. 1. We start with the effect an event occurrence has on attributes. For instance the effect of `Open` events for account objects is represented by the following temporal logic formula:

$$\Box(\ a.\nabla\mathrm{Open}(B,N,H) \rightarrow \bigcirc(a.\mathrm{Bank} = B \wedge a.\mathrm{No} = N \wedge a.\mathrm{Holder} = H)\ )$$

Due to the fact that `Open` is a birth event it may only occur once in the life of an object. This property being inherent to the object model of the specification language Troll can be expressed by:

$$\Box(\ a.\nabla\mathrm{Open}(B,N,H) \rightarrow \bigcirc\Box\neg(\exists B',N',H' : a.\nabla\mathrm{Open}(B',N',H'))\ )$$

Event calling as it may be specified for `Transfer` events in bank objects could be expressed by temporal logic formulas as follows (where b refers to a bank object):

$$\begin{aligned}\Box(\ b.\nabla\mathrm{Transfer}(A_1,A_2,M) \rightarrow (\ & \mathrm{Account}(A_1).\nabla\mathrm{Withdrawal}(M)\wedge \\ & \mathrm{Account}(A_2).\nabla\mathrm{Deposit}(M)\qquad)\ )\end{aligned}$$

Evolving temporal Logic (ETL). Based on the linear temporal logic described before we have to find an extension for the treatment of the special attribute having sets of first-order formulas as values. In order to represent this special property we introduce a corresponding predicate $\mathcal{V}$ into our logic. This predicate is used to express the current validity of the dynamic behavior axioms. For simplicity, we restrict our consideration to one special predicate over first-order temporal formulas.[1]

This predicate is used to express the state-dependent validity of first-order formulas: $\mathcal{V}(\widetilde{\varphi})$ holds in a state (at an instant of time) means that the specification φ is valid w.r.t. that state.

In a more formal way we can express this as follows: if $\mathcal{V}(\widetilde{\varphi})$ holds for a (linear) life cycle λ (i.e., $\lambda \models \mathcal{V}(\widetilde{\varphi})$) then φ holds for λ as well:

$$\lambda \models \mathcal{V}(\widetilde{\varphi}) \quad \textbf{implies} \quad \lambda \models \varphi$$

In order to avoid severe problems especially caused by substitution we assume $\mathcal{V}$ to work only on syntactic representations of first-order temporal formulas instead of the formulas themselves. Here, we use the notation $\widetilde{\varphi}$ to distinguish such a syntactic representation from the formula φ. For a correct formal treatment we have to define an abstract data type `Formula` for first-order temporal formulas as possible parameter values for $\mathcal{V}$. In addition a function translating values of this abstract data type into corresponding formulas is needed.

W.r.t. the reflection of $\mathcal{V}(\widetilde{\varphi})$ on the first-order level, we may establish the following axiom for ETL:

$$\mathcal{V}(\widetilde{\varphi}) \rightarrow \varphi$$

[1] For dealing with several objects having different sets of currently valid behavior axioms, we could extend this view to several predicates or to introduce an additional parameter to the predicate for referring to different objects. In the same way, we can deal with the case that one object has several of these attributes.

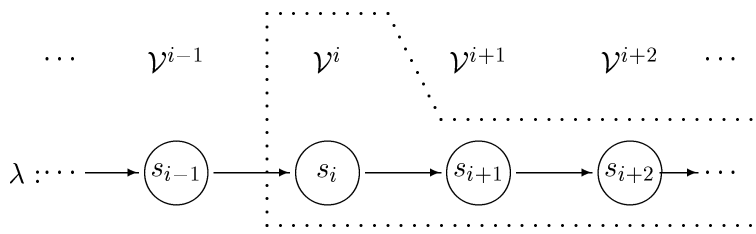

Figure 3: Interpreting Evolving Temporal Logic.

By the predicate $\mathcal{V}$ we simulate the finite set of behavior axioms which are currently valid. Thus $\mathcal{V}(\widetilde{\varphi})$ can be read as "$\widetilde{\varphi}$ is in the set of currently valid behavior axioms". Due to $\mathcal{V}(\widetilde{\varphi}) \rightarrow \varphi$, it is sufficient that $\mathcal{V}$ holds only for a finite set of specification axioms because the theory induced by these axioms is generated on the first-order level in the usual way.

Please note that $\mathcal{V}(\widetilde{\varphi})$ can be considered as an elementary proposition in ETL. Therefore, we may assume that for each state s_i in a life cycle λ there is a truth assigning function denoting the validity of $\mathcal{V}(\widetilde{\varphi})$ for each first-order formula φ.

From the definition given before and from the usual properties of the temporal operators we can now immediately conclude:

$$\lambda \models \mathcal{V}(\widetilde{\Box\varphi}) \quad \textbf{implies} \quad \forall i \geq 0 : \lambda^i \models \varphi$$
$$\lambda \models \mathcal{V}(\widetilde{\Diamond\varphi}) \quad \textbf{implies} \quad \exists i \geq 0 : \lambda^i \models \varphi$$

This is due to $\lambda \models \mathcal{V}(\widetilde{\Box\varphi})$ implies $\lambda \models \Box\varphi$ and $\lambda \models \Box\varphi$ is defined by $\forall i \geq 0 : \lambda^i \models \varphi$ (and analogously for $\Diamond\varphi$). This special property is depicted in Fig. 3: Assume $\mathcal{V}(\widetilde{\Box\varphi})$ holds in state s_i in a life cycle λ. Then φ holds in all the states $s_i, s_{i+1}, s_{i+2}, \ldots$ — independent of whether $\mathcal{V}(\widetilde{\Box\varphi})$ is true in $s_{i+1}, s_{i+2}, \ldots$ Therefore, it should be clearly noted that there is a big difference between $\mathcal{V}(\widetilde{\Box\varphi})$ and $\mathcal{V}(\widetilde{\varphi})$. Once $\mathcal{V}(\widetilde{\Box\varphi})$ has become true, φ remains true forever. In contrast, if $\mathcal{V}(\widetilde{\varphi})$ becomes true, φ needs only to remain true as long as $\mathcal{V}(\varphi)$ does.

For the events manipulating the special attribute `Axioms` (in the specification called mutators) we need counterparts in the logic. For a general manipulation of the predicate $\mathcal{V}$ we introduce two special events $axiom^+(\widetilde{\varphi})$ and $axiom^-(\widetilde{\varphi})$ for adding an axiom to $\mathcal{V}$ and for removing an axiom from $\mathcal{V}$, respectively. From the logical point of view these two events are sufficient for representing all possible ways of manipulating the attribute `Axioms`. As introduced before we use the notation $\nabla axiom^+(\widetilde{\varphi})$ for denoting the occurrence event $axiom^+$ (analogously for $axiom^-$). For occurrences of these events the following axioms are given:

$$\nabla axiom^+(\widetilde{\varphi}) \rightarrow \bigcirc\mathcal{V}(\widetilde{\varphi})$$
$$\nabla axiom^-(\widetilde{\varphi}) \rightarrow \bigcirc\neg\mathcal{V}(\widetilde{\varphi})$$

$\nabla axiom^+(\widetilde{\varphi})$ (or $\nabla axiom^-(\widetilde{\varphi})$) leads to $\mathcal{V}(\widetilde{\varphi})$ ($\neg\mathcal{V}(\widetilde{\varphi})$, resp.) in the subsequent state. Frame rules are assumed restricting the evolution of $\mathcal{V}$ to changes which are caused by occurrences of the events $axiom^+$ and $axiom^-$:

$$\neg\mathcal{V}(\widetilde{\varphi}) \wedge \bigcirc\mathcal{V}(\widetilde{\varphi}) \rightarrow \nabla axiom^+(\widetilde{\varphi})$$
$$\mathcal{V}(\widetilde{\varphi}) \wedge \bigcirc\neg\mathcal{V}(\widetilde{\varphi}) \rightarrow \nabla axiom^-(\widetilde{\varphi})$$

Before we show how to formulate some properties specified in Fig. 2 we want to briefly discuss the understanding of negation w.r.t. the predicate $\mathcal{V}$. The question to answer is whether $\mathcal{V}(\widetilde{\neg\varphi})$

is different from $\neg\mathcal{V}(\widetilde{\varphi})$. The answer is quite simple: From $\lambda \models \mathcal{V}(\widetilde{\neg\varphi})$ it follows that $\lambda \models \neg\varphi$. In contrast we cannot derive the same from $\neg\mathcal{V}(\widetilde{\varphi})$. Therefore, $\mathcal{V}(\widetilde{\neg\varphi})$ and $\neg\mathcal{V}(\widetilde{\varphi})$ have to be distinguished. This is of course not surprising because it corresponds to our intuition about the predicate $\mathcal{V}$.

Another important issue we do not discuss in full detail is a proof system for ETL. In fact, we think of taking a proof system for first-order linear temporal logic (like OSL [SSC92]) and extending it a little bit in order to get a grasp of the predicate $\mathcal{V}$.

4.2 Expressing the Example Using ETL

In the example given in Fig. 2 several properties are specified for the special attribute `Axioms`. Here, we formulate some of them as ETL formulas where the attribute `Axioms` is represented by the special predicate $\mathcal{V}$. Due to the fact that we have to distinguish between different agents we prefix each occurrence of $\mathcal{V}$ in a formula by a variable (or an agent name) referring to the agent concerned. This corresponds to the way we have prefixed predicates denoting an event occurrence for an agent before.

In all formulas given below there is an implicit universal quantification over all variables (including $\widetilde{\varphi}$). Please recall that we assume $\widetilde{\varphi}$ to be a variable over an abstract data type `Formula`.

The way we express the initial value property for `Axioms`, i.e., that directly after the occurrence of the birth event `Open` there is no formula $\widetilde{\varphi}$ for which $\mathcal{V}(\widetilde{\varphi})$ holds is a little bit tricky:

$$\Box(\ \bigcirc a.\mathcal{V}(\widetilde{\varphi}) \rightarrow \neg a.\nabla\mathrm{Open}(B, N, H)\ )$$

The effect the so-called mutator event `AddAxioms` has on the value of `Axioms` can be described by simply reducing the occurrence of `AddAxioms` to occurrences of the special pre-defined event $axiom^{+}$:

$$\Box(\ a.\nabla\mathrm{addAxioms}(\widetilde{\Phi}) \wedge \widetilde{\varphi} \in \widetilde{\Phi}\ \rightarrow\ a.\nabla axiom^{+}(\widetilde{\varphi})\ )$$

For the mutator event `ResetAxioms` we choose a similar way of expressing its effect:

$$\Box(\ a.\nabla\mathrm{ResetAxioms} \wedge a.\mathcal{V}(\widetilde{\varphi}) \rightarrow a.\nabla axiom^{-}(\widetilde{\varphi})\ )$$

Considering the property of $axiom^{+}$ described before we can immediately conclude:

$$\Box(\ a.\nabla\mathrm{ResetAxioms} \wedge a.\mathcal{V}(\widetilde{\varphi}) \rightarrow \bigcirc\neg a.\mathcal{V}(\widetilde{\varphi})\ )$$

Finally, the effect of the mutator event `RemoveAxioms` can be described by:

$$\Box(\ a.\nabla\mathrm{RemoveAxioms}(\widetilde{\Phi}) \wedge \widetilde{\varphi} \in \widetilde{\Phi}\ \rightarrow\ a.\nabla axiom^{-}(\widetilde{\varphi})\ )$$

Obviously, it is possible to express a nearly arbitrary manipulation of the behavior specification. From a pragmatic point of view this is not a desirable property. Therefore, we think of restricting the possibilities by means of the specification language. The specification language should only allow those ways of manipulating the dynamic behavior specification which can be captured by the logic in a reasonable way. Furthermore, we have to make sure that only certain users (represented by special objects or agents) are allowed to change the dynamic part of the specification. For that, additional mechanisms are needed in the specification framework.

5 Conclusions

In this paper we have motivated the necessity of evolving specifications in the area of information systems. As a rather straightforward step to modeling information systems dynamics, we presented a first approach of an agent-oriented specification framework. For that, 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 presented 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. Furthermore, we sketched an extension of linear temporal logic (called ETL, Evolving Temporal Logic) which allows us to express dynamically changing behavior within the logic. Thereby, it becomes possible to reason about changes of behavior.

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, and 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 seems to be 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.

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 information systems consisting of components which are partially autonomous.

Acknowledgements: We thank Michael Höding, Jan Kusch and Ingo Schmitt for useful remarks.

References

[Bro92] M. L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Systems. In P. M. Gray and R. J. Lucas, editors, *Advanced Database Systems, Proc. of the 10th British National Conf. on Databases, BNCOD 10, Aberdeen, Scotland, July 1992*, pages 1–28. LNCS 618, Springer-Verlag, Berlin, 1992.

[Buc91] A. P. Buchmann. Modeling Heterogeneous Systems as an Active Object Space. In *Implementing Persistent Object Bases, Principles and Practice, Proc. of the 4th Int. Workshop on Persistent Object Systems, Martha's Vineyard, MA, USA, September 23–27, 1990*, pages 279–290. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Con96] S. Conrad. A Basic Calculus for Verifying Properties of Interacting Objects. *Data & Knowledge Engineering*, 18(2):119–146, March 1996.

[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)*, pages PP7:1–16. Participant's Proceedings, Paderborn, September 1995.

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

[FW93] R. B. Feenstra and R. J. Wieringa. LCM 3.0: A Language for describing Conceptual Models. Technical Report, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[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, editors. *Proc. of the Int. Conf. Intelligent and Cooperating Information Systems, Rotterdam, The Netherlands.* IEEE Computer Society Press, May 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.

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

[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*, 14(2):175–211, April 1996.

[Jun93] R. Jungclaus. *Modeling of Dynamic Object Systems — A Logic-Based Approach.* Advanced Studies in Computer Science, Vieweg Verlag, Wiesbaden, 1993.

[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*, pages 251–275. North-Holland, Amsterdam, 1992.

[Rei84] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modeling*, pages 191–239. Topics in Information Systems, Springer-Verlag, New York, 1984.

[Rya93] M. Ryan. Defaults in Specifications. In A. Finkelstein, editor, *Proc. of the IEEE Int. Symposium on Requirements Engineering (RE'93), San Diego, CA*, pages 142–149. IEEE Computer Society Press, 1993.

[Rya94] M. Ryan. Belief Revision and Ordered Theory Presentation. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information.* De Gruyter Publishers, 1994.

[SCT95] G. Saake, S. Conrad, and C. Türker. From Object Specification towards Agent Design. In M. Papazoglou, editor, *OOER'95: Object-Oriented and Entity-Relationship Modeling, Proc. of the 14th Int. Conf., Gold Coast, Australia*, pages 329–340. LNCS 1021, Springer-Verlag, Berlin, December 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.

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

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P. M. Stocker and W. Kent, editors, *Proc. of the 13th Int. Conf. on Very Large Data Bases (VLDB'87), Brighton, England*, pages 107–116. Morgan Kaufmann Publishers, Los Altos, CA, September 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 R. Wieringa and R. Feenstra, editors, *Information Systems — Correctness and Reusability. Selected Papers from the IS-CORE Workshop*, pages 84–99. World Scientific Publishing, 1995.

[TCS96] C. Türker, S. Conrad, and G. Saake. Dynamically Changing Behavior: An Agent-Oriented View to Modeling Intelligent Information Systems. In Z. W. Raś and M. Michalewicz, editors, *Foundations of Intelligent Systems, Proc. of the 9th Int. Symposium on Methodologies for Intelligent Systems, ISMIS'96, Zakopane, Poland*, pages 572–581. LNAI 1079, Springer-Verlag, Berlin, June 1996.

[WJ95] M. J. Wooldridge and N. R. Jennings. Agents Theories, Architectures, and Languages: A Survey. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents, Proc. of the ECAI'94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 1994*, pages 1–39. LNAI 890, Springer-Verlag, Berlin, 1995.