# Towards Agent-Oriented Specification of Information Systems

Stefan Conrad Can Türker Gunter Saake

Otto-von-Guericke-Universität Magdeburg, Institut für Technische Informationssysteme

Postfach 4120, D–39016 Magdeburg, Germany, E-mail: `conrad@iti.cs.uni-magdeburg.de`

— 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 concept of *agent* 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.

Our approach can be described as follows: We start from a well-known object-oriented specification framework for information systems which allows us to describe structure as well as long-term behavior of objects in information systems. We extend this object-oriented framework by introducing a certain form of knowledge as part of the internal state of objects. Here, we especially 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.

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.

**Example:** In Figure 1 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 [Jungclaus *et al.* 1996]. 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 has **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.

The standard object-oriented specification can be translated into a temporal logic for objects (cf. [Conrad 1996] and [Jungclaus *et al.* 1996]). For capturing the semantics of evolving behavior we need an extended temporal logic. In [Conrad & Saake 1995] a preliminary version of an *Extended Temporal Logic* (ETL) is described. In ETL a special predicate $\mathcal{V}$ is introduced for manipulating the changeable part of the behavior specification. This predicate is used to express the current validity of the dynamic behavior axioms: $\mathcal{V}(\widetilde{\varphi})$ holds in a state (at an instant of time) means that the specification $\varphi$ is valid w.r.t. that state (please note that $\widetilde{\varphi}$ is a syntactic representation of a first order temporal formula $\varphi$). W.r.t. the reflection of $\mathcal{V}(\widetilde{\varphi})$ on the first-order level, we may establish the following axiom for ETL:

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

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 $\mathcal{V}(\widetilde{\varphi}) \to \varphi$, 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.

For the events manipulating the special attribute `Axioms` (in the specification called mutators) we need counterparts

```
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;
  events            Open(BID:|Bank|, AccNo:nat, AccHolder:|Customer|) birth;
                     Withdrawal(W:money);
                     Deposit(D:money);
                     ...
  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;
  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 1: Specification of an agent class `Account`

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 (where $\bigcirc$ is the usual linear temporal next operator):

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

Based on this simple extension of a linear temporal logic framework for object specification, we can already express and reason about evolving temporal behavior. Although this logic has some non-first-order aspects on the syntactic level, we can provide a first-order semantics for it.

In [Türker, Conrad, & Saake 1996] we discuss the example in more detail. Especially, different ways of specifying evolving behavior are discussed. Furthermore, we discuss the concept of *agent* in more detail and show how it can be used for extending the capabilities of object-oriented specification frameworks.

**Acknowledgements:** 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).

# References

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

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

[Jungclaus *et al.* 1996] Jungclaus, R.; Saake, G.; Hartmann, T.; and Sernadas, C. 1996. Troll – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems* 14(2):175–211.

[Türker, Conrad, & Saake 1996] Türker, C.; Conrad, S.; and Saake, G. 1996. Dynamically Changing Behavior: An Agent-Oriented View to Modeling Intelligent Information Systems. In Raś, Z. W., and Michalewicz, M., eds., *Foundations of Intelligent Systems (ISMIS'96)*, 572–581. LNAI 1079, Springer-Verlag, Berlin.