AOR Modelling and Simulation: Towards a General Architecture for Agent-Based Discrete Event Simulation*

Gerd Wagner

Faculty of Technology Management, I&T, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
G.Wagner@tm.tue.nl
http://www.tm.tue.nl/it/staff/gwagner/

Abstract. Agent-oriented modelling of software systems and agent-based simulation are commonly viewed as two separate fields with different concepts and techniques. We show that the *Agent-Object-Relationship (AOR)* modelling language, proposed in [Wag03] for information systems analysis and design, can also be used for specifying simulation models that can be executed by an agent-based simulation system. The suitability of AOR modelling for simulation is also supported by the fact that the AOR meta-model and the meta-model of discrete event simulation can be combined into a model of agent-based discrete event simulation in a natural way.

1 Introduction

Agent-based simulation (ABS) is a new paradigm that has been applied to simulation problems in biology, engineering, economics and sociology. In ABS, a scenario of systems that interact with each other and with their environment is being modelled, and simulated, as a *multiagent system*. The participating agents – animals, humans, social institutions, software systems or machines – can perform actions, perceive their environment and react to changes in it. They also have a mental state comprising components such as knowledge/beliefs, goals, memories and commitments.

Compared to traditional simulation methods – like mathematical equations, discrete event-simulation, cellular automata and game theory – ABS is less abstract and closer-to-reality, since it explicitly attempts to model the specific behaviour of individual actors, in contrast to macro simulation techniques that are typically based on mathematical models averaging the behaviour effects of individuals or of entire populations (see [Dav02]).

There are many different formalisms and implemented systems that are all subsumed under the title 'agent-based simulation'. For example, one of the most prominent ABS systems is SWARM [MBLA96], an object-oriented programming library that provides special support for event management, but does not support any cognitive agent concept. Like SWARM, many other ABS systems do not have a theoretical foundation in the form of a *metamodel*. They therefore do not allow the specification

* This paper improves and extends [WT03].

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 174–188, 2004.

of a simulation model in a high-level declarative language but require specifying simulation models at the level of imperative programming languages.

We take a different approach and aim at establishing an ABS framework based on a high-level declarative specification language with a UML-based visual syntax and an underlying simulation metamodel as well as an abstract simulator architecture and execution model. The starting point for this research effort is an extension and refinement of the classical *discrete event simulation* paradigm by enriching it with the basic concepts of the *Agent-Object-Relationship (AOR)* metamodel [Wag03].

2 Modelling Agents and Multiagent Systems

2.1 AOR Modelling

The AOR modelling language (AORML) is based on an ontological distinction between active and passive entities, that is, between agents and (non-agentive) objects of the real world. The agent metaphor subsumes *artificial* (software and robotic), *natural* (human and animal) as well as *social/institutional* agents (groups, organizations etc.).

In AORML, an entity is an agent, an event, an action, a claim, a commitment or an ordinary object. Only agents can communicate, perceive, act, make commitments and satisfy claims. Objects are passive entities with no such capabilities. Besides human and artificial agents, AORML also includes the concept of *institutional agents,* which are composed of a number of other agents that act on their behalf. Organizations and organizational units are important examples of institutional agents.

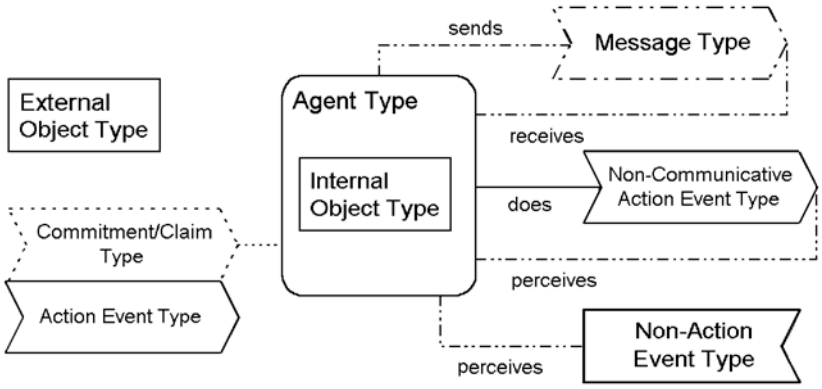

Fig. 1. The core state structure modelling elements of external AOR diagrams.

There are two basic types of AOR models: *external* and *internal* models. An external AOR model adopts the perspective of an external observer who is looking at the (prototypical) agents and their interactions in the problem domain under consideration. In an internal AOR model, we adopt the internal (first-person) view of a particular agent to be modelled. While a (business) domain model corresponds to an external model, a design model (for a specific information system) corresponds to an internal model, which can be derived from the external one.

Fig. 1 shows the most important elements of external AOR state structure modelling. There is a distinction between *action events* and *non-action events*, and between a *communicative* action event (or *message*) and a *non-communicative* action event. Fig. 1 also shows that a *commitment/claim* is coupled with the action event that fulfils that commitment (or satisfies that claim).

The most important behaviour modelling element of AORML are ***reaction rules***, which are used to express *interaction patterns*. A reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the agent rectangle whose reaction pattern it represents. Each reaction rule has exactly one incoming arrow with a solid arrowhead: it specifies the triggering event type. In addition, there may be ordinary incoming arrows representing state conditions (referring to corresponding instances of other entity types). There are two kinds of outgoing arrows: one for specifying mental effects (changing beliefs and/or commitments) and one for specifying the performance of (physical and communicative) actions. An outgoing arrow with a double arrowhead denotes a mental effect. An outgoing connector to an action event type denotes the performance of actions of that type.

Fig. 2 shows an example of an *interaction pattern diagram* specifying a reaction rule R1 for an elevator in response to the non-action event type `Arrival` at some floor. Notice that all state attributes of the elevator are optional, expressed by the UML multiplicity expression [0..1]. Both the precondition and the postcondition are expressed in UML's Object Constraint Language (OCL). The precondition for halting is `Arrival.Floor = TargetFloor`, the two sequentially triggered action events are `halt` and `closeDoor` and the postcondition representing the mental state effect is `HaltFloor = Arrival.Floor`.

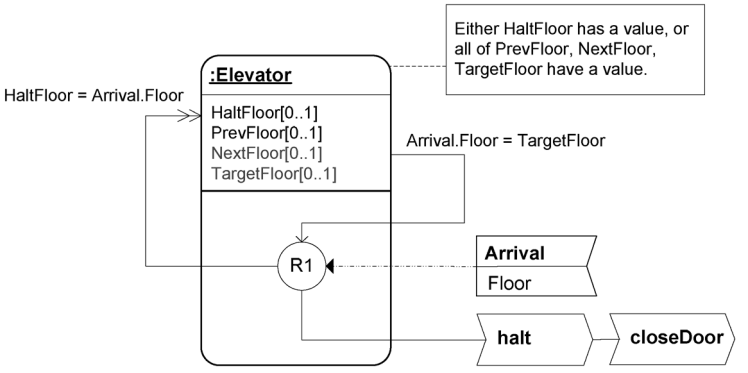

Fig. 2. An interaction pattern: when arriving at a floor, the elevator halts and opens its door.

In symbolic form, a reaction rule is defined as a quadruple

$$\varepsilon, C \rightarrow \alpha, P$$

where ε denotes an event term (the triggering event), C denotes a logical formula (the mental state condition), α denotes an action term (the triggered action) and P denotes a logical formula (specifying the mental effect or postcondition).

2.2 Modelling and Simulating Communication and Perception

For modelling and simulating *communication* between agents, we do not consider non-verbal communication and abstract away from the physical layer, where the speaker realizes a communication act (or, synonymously, sends a message) by performing some physical action (such as making an utterance) and the listener has to perceive this action event, implying that, due to the physical signal transmission, there can be noise in the listener's percepts referring to the message send event. In general, for each (external) *event E* and each agent, the simulation system has to compute a corresponding *potential percept PP* (according to physical laws), from which the *actual percept AP* has to be derived according to the perceptive capability of the agent. The mental availability of the actual percept, then, is the (internal) perception event corresponding to the external event. There are two options for simplifying the $E \rightarrow PP \rightarrow AP$ chain: we can either assume that

1. $E = PP = AP$, so we don't have to distinguish between an external event and the corresponding perception event; or
2. $PP = AP$, that is, all agents have perfect perception capabilities.

For communication events, it makes sense to assume that $E = PP = AP$, i.e. the message received is equal to the corresponding message sent. Yet, there may be a delay between these two events, depending on the type of the message transport channel and the current physical state of the speaker and listener.

For the perception of a non-communicative action event such an assumption may be not justified and would mean a severe simplification. However, the less severe simplification expressed by the assumption that $PP = AP$ may be justified for many purposes.

A *send message* action type is represented internally (in a behaviour rule) in the form of

sM(Receivers, M(C1,C2,...))

where M denotes the message type, and the Ci denote the message content parameters. When such an action is performed (as a consequence of firing the behaviour rule), we obtain external sM event expressions (in the future events list) of the form

sM(EvtID1, Sender, Receivers, M(C1,C2,...)) @ OccTime

corresponding to the external AOR model of a communicative action event (such as in Fig. 6). Such an sM event may be processed by creating *memory facts* of the form

M(EvtID1, OccTime, Sender, Receivers, C1, C2, ...)

and to *receive message* events of the form

rM(EvtID2, Receiver, EvtRef) @ OccTime+D

where the delay D is a non-zero multiple of the cycle duration and EvtRef refers to the EvtID of the corresponding sM event (i.e. EvtRef = EvtID1).

2.3 Examples

We briefly discuss three simple examples.

2.3.1 Example 1: An Elevator

A simple example scenario is an elevator with the action repertoire: moveUp, moveDown, halt, openDoor and closeDoor. The elevator agent has to react to pickup and transport requests from elevator users; when moving, it has to react to events signalling the arrival at a certain floor; and it has to react to timeout signal to close the door. This scenario is described in the AOR diagram in Figure 3.

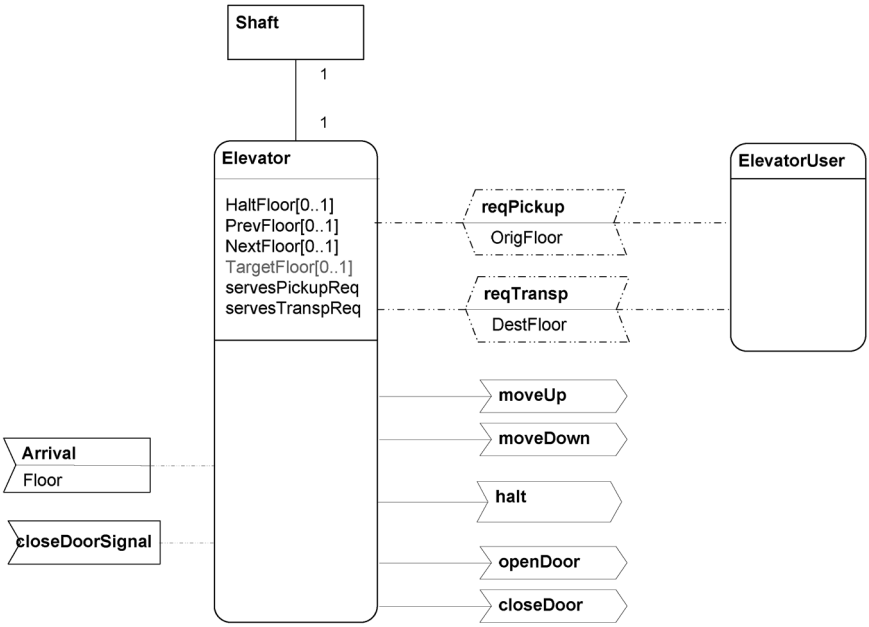

Fig. 3. An elevator scenario as an external AOR model.

An AORS scenario model describes both the systems/agents to be simulated (here the elevator) and the systems/agents outside the simulation borderline they interact with (here the elevator user). In a next step, such a scenario model is transformed into a simulation model in which the external systems are dropped and all action event types in interaction frames involving them (here reqPickup and reqTransp) are turned into corresponding *exogenous event* (here `PickupReq`) or *successor event* types (here `TranspReq`) as shown in Figure 4. ***Exogenous events*** drive the simulation and are generated at random; their periodicity is specified with the help of a tagged value – in the case of PickupReq: `{periodic=exp(…)}`. They are either non-action events that are not caused by other events or external action events in the sense that their actor is not included in the simulation.

In the simulation model of Figure 4, there are also two examples of reaction rules that represent causation rules. In AOR simulation, as explained in section 3, such

rules are used to represent a model of causality that is enacted by the environment simulator. The first causation rule, R1, creates an `Arrival` event, perceived by the elevator with a delay of 7 seconds, upon any `moveUp` or `moveDown` event. The second causation rule, between the `openDoor` action event and the `closeDoorSignal` event, is expressed in a visually simplified notation (an association connection line with stereotype «causes») specifying that, upon opening the door, a specific signal for closing the door is created with a delay of 5 seconds.

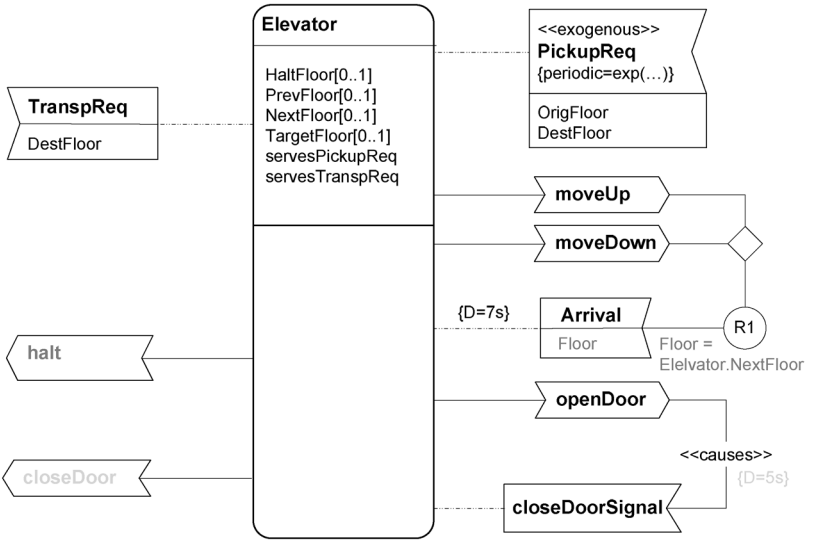

Fig. 4. A simulation model for the elevator system.

2.3.2 Example 2: Communicating Elevators

We consider an example scenario where two elevators operate in the same shaft and must take care to avoid collisions. For simplicity, we restrict our consideration to the case with three floors. Elevator A is serving floor 1 and 2, and elevator B is serving floor 2 and 3, and hence the critical zone, requiring coordination, is floor 2. This scenario is depicted in Fig. 5.

The external AOR diagram in Fig. 6 models this scenario, which requires a model of coordination between the two elevators.

Fig. 5. In addition to the modelling elements used to describe the single elevator scenario, there are two message types (`reqPerm`, `grantPerm`) and a commitment/claim type for `grantPerm` for the coordination between the two elevators.

2.3.3 Example 3: The MIT Beer Game

The MIT Beer Game is a management simulation that was developed by the System Dynamics Group of the Sloan School of Management in the early 1960s.

The game is played by teams of four persons, each person playing one of the four roles of Retailer, Wholesaler, Distributor and Factory. The four roles are arranged in a simple beer supply chain (Fig. 7).

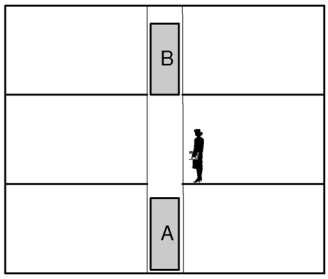

Fig. 5. Two elevators operating in one shaft.

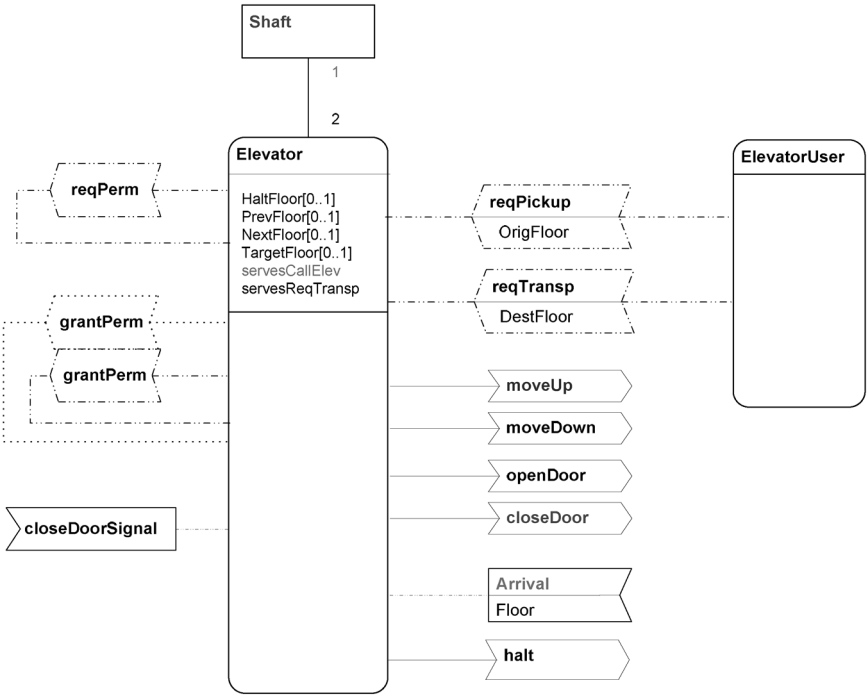

Fig. 6. An external AOR diagram modelling the communicating elevator scenario from.

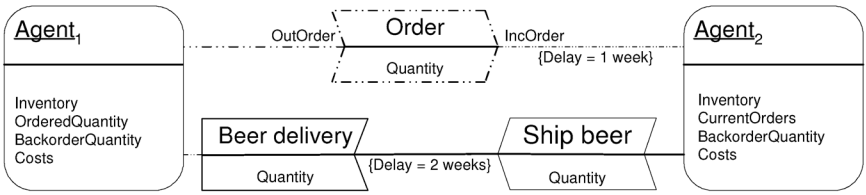

Fig. 7. The interaction frame between any two nodes of the MIT Beer Game supply chain.

The factory has an unlimited supply of raw materials and each of the roles has an unlimited storage capacity. The supply lead times (the time between shipment by the supplier and arrival at its destination) and order delay times (the time between the sending of an order and the arrival of the order at its destination) are fixed.

Supplies take two full turns to arrive; the orders for new beer arrive at their destination in the next turn.

Each turn (usually a simulated week), the retailer receives a customer order and tries to ship the requested amount from its inventory. It then orders an amount of beer from its supplier, the wholesaler, which tries to ship the requested amount from its inventory, and so on. Orders that cannot be met are placed in backorder and must be met as soon as possible.

At the end of each week, each role has to calculate that week's costs. Remaining inventory is charged $0.50 per item as holding costs and backorders are charged $1.00 per item as shortage costs. The objective of the game is to be the team with the lowest overall costs or to be the player with the lowest cost within a team, after playing a fixed number of weeks.

In [LTW04], we present a complete AORS model of the Beer Game.

3 Agent-Based Discrete Event Simulation

In Discrete Event Simulation (DES) systems are modelled in terms of *system states* and *discrete events*, i.e. as *discrete dynamic systems*. Since a system is composed of entities, its state consists of the combination (Cartesian product) of all states of its entities. All state changes are brought about by events.

DES is a very generally applicable and powerful approach, since many systems, in particular technical and social systems, can be viewed as discrete dynamic systems. There are two methods in DES for the advance of simulated time: using *event-based time progression*, the time is advanced according to the occurrence time of the next event; using *incremental time progression*, the simulated time is advanced in regular time steps.

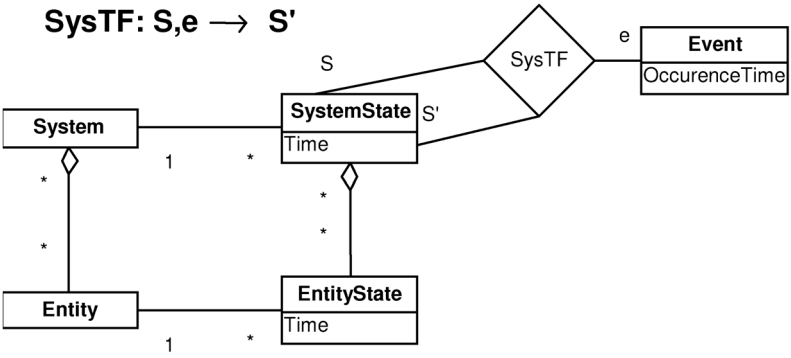

Fig. 8. The basic model of discrete event simulation as a UML class diagram. As a system consists of a number of entities, its state consists of all the states of these entities. A system transition function *SysTF* takes a system state *S* and a state-changing event *e* and determines the resulting successor state *S'*.

In many ABS approaches, the basic DES model is refined in some way by making certain additional conceptual distinctions, including the fundamental distinction between interacting agents and passive objects. These simulation approaches may be classified as *Agent-Based Discrete Event Simulation (ABDES)*.

In our version of ABDES, extending and refining the basic DES model into a model of *Agent-Object-Relationship Simulation (AORS)*, we start with time-driven DES (since we need small regular time steps for simulating the perception-reaction cycle of agents) and adopt a number of essential ontological distinctions from AORML:

- The enduring entities of a system (also called *endurants* in foundational ontologies) are distinguished into *agents* and *objects*.
- Agents maintain *beliefs* (referring to the state of their environment) and process *percepts* (referring to events).
- Events can be either *action events* or *non-action events*.
- Action events can be either *communicative* (messages) or *non-communicative*.

In addition to these conceptual distinctions from AORML, we need to introduce the distinction between *exogenous events* (non-action events that are not caused by other events or external action events in the sense that their actor is not included in the simulation) generated periodically at random and *successor events*, which are either caused events or action events. The basic DES model is shown in Fig. 8 (an extension thereof is depicted later in Fig. 10).

3.1 An Abstract Architecture and Execution Model for ABDES Systems

In ABDES, it is natural to partition the simulation system into

1. the ***environment simulator*** responsible for managing the state of all external (or physical) objects and for the external/physical state of each agent;
2. a number of ***agent simulators*** responsible for managing the internal (or mental) state of agents.

The state of an ABDES system consists of:

- the simulated time *t*
- the environment state representing
  - the non-agentive environment (as a collection of objects) and
  - the external states of all agents (e.g., their physical state, their geographic position etc.)
- the internal agent states (e.g., representing perceptions, beliefs, memory, goals etc.).
- a (possibly empty) list of future events

For each simulation run, a start and an end calendar date-time are specified. The simulated time t represents a cycle counter, which can be transformed into a calendar date-time with the help of a cycle duration constant (for which we use 100 ms by default). The simulation run stops at the end date-time. By default, both the perception-action and the action-perception delay are 100 ms, as illustrated in Fig. 9.

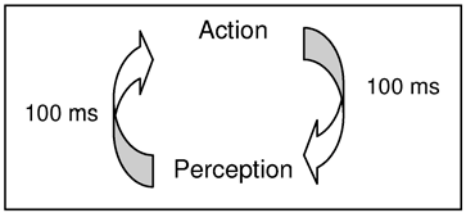

Fig. 9. The perception-action and action-perception delays.

At initialization time, for each exogenous event type, future events are created at random (according to the given distribution function) and put in the future events list.

At runtime, a simulation cycle consists of the following steps:

1. At the beginning of a new simulation cycle, say at simulated time t, the environment simulator determines the current events (all the events of the future events list whose occurrence time is now).
2. The environment simulator determines, on the basis of the current environment state and the current events:
   a) a new environment state
   b) a set of new caused events (with suitable occurrence time) created by the causality rules of the environment simulator, including
      - new caused *environment events*
      - next step *receive message events* (with occurrence time *t+1*) derived from the corresponding current send message events
      - next step *perception events* (with occurrence time *t+1*) derived from the corresponding current *environment events* and *non-communicative action events*
   c) The environment simulator then hands over to each agent simulator the current *receive message events* and *perception events* for the simulated agent.
3. Each agent simulator computes, on the basis of the current internal agent state and the delivered *perception events* and *receive message events*,
   a) a new internal agent state,
   b) the set of new *action events* (with occurrence time *t+1*) created by the behaviour rules of the agent simulator (representing the immediate reactions of the simulated agent in response to the delivered *perceptions* and *message receptions*)
4. The future events list is updated by removing all the processed events and adding the new caused events from step 2b and the new action events from step 3b.
5. If the simulation run continues, the environment simulator sets the simulated time *t*
   a) to *t+1*, if it is in incremental time progression mode, or
   b) to the occurrence time of the next event from the future events list, if it is in event-based time progression mode,

   and starts over with step 1 of the simulation cycle.

The simulation run ends when the future events list is empty or when the end date-time is reached.

This abstract architecture and execution model for ABDES systems can be instantiated by different concrete architectures and systems. In section 4, we present a Prolog program that implements an AORS system and instantiates this architecture.

The AORML distinction between external and internal models provides the means needed for modelling both the environment and the agents involved in a simulation scenario. An external model describes the perspective of the environment simulator, whereas the internal models derived from the external one describe the perspectives of the involved agents. This suggests the following methodology for developing an AOR simulation model:

1. In the domain analysis of the simulation problem, develop an external AOR model of the scenario from the perspective of an external observer. This model is the basis both for designing the environment simulation and for deriving the specification of the involved agent simulators.
2. For each involved agent, transform the external AOR model of the simulation scenario into an internal AOR model for specifying the corresponding agent simulator.

3.2 Advantages of ABDES and AORS

ABDES and AORS support

- structure-preserving modelling and closer-to-reality simulation:
  - Passive entities with certain properties are modelled as objects with corresponding attributes.
  - Interactive entities (actors) are modelled as agents, which have beliefs and perceptions, and interact with each other and with their environment.
- functionally distributed simulation where any of the participating simulators (the environment simulator and all involved agent simulators) may be deployed to different threads or processes, possibly running on different machines (realizing vertical distribution).
- *interactive* simulation where any of the involved agent simulators may be replaced by its real counterpart.
- modelling and simulating *pro-active* behaviour, in addition to the basic reactive behaviour.

4 A Prolog Prototype of an AOR Simulation System

Implemented as a Prolog program, the AOR simulation cycle yields the following procedure:

```
1:  cycle( _, _, _, []) :- !.
2:  cycle( Now, EnvSt, IntAgtSts, EvtList) :-
3:      extractCrtEvts( Now, EvtList, CrtEnvEvts, CrtPercEvts),
4:      envSimulator( Now, CrtEnvEvts, EnvSt, NewEnvSt,
            TranslCausEvts),
5:      agtsSimulator( Now, CrtPercEvts, IntAgtSts,
            NewIntAgtSts, TranslActEvts),
6:      computeNewEvtList( EvtList, CrtEnvEvts, TranslCausEvts,
            TranslActEvts, NewEvtList),
7:      NextMoment is Now+1,
8:      cycle( NextMoment, NewEnvSt, NewIntAgtSts, NewEvtList).
```

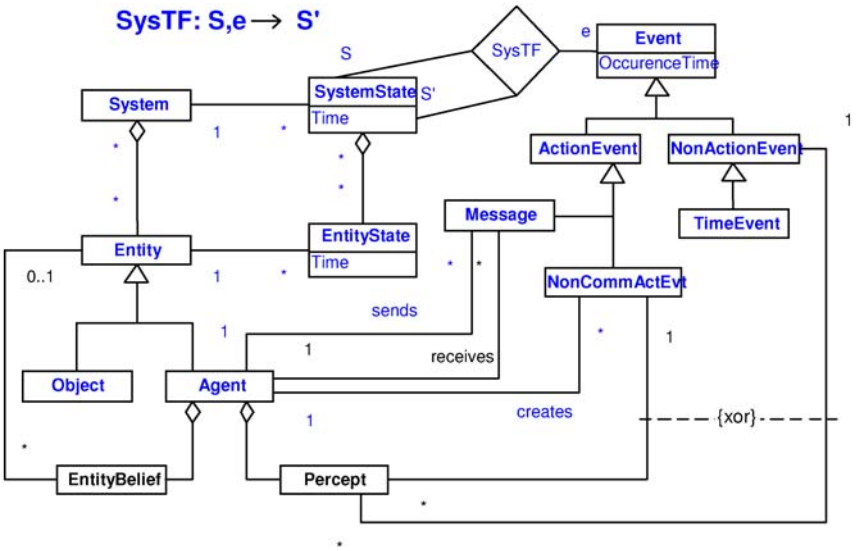

Fig. 10. A UML class diagram describing the basic ontology of AORS: Agents maintain beliefs about entities, send/receive messages, and process percepts, which refer either to a non-communicative action event or to a non-action event. Notice that communication (sending/receiving messages) is separated from perception (perceiving non-communicative action events and non-action events).

Line 1 represents the exit condition (when the future events list is empty). In line 3, the current environment events (steps 1a and 1b of the simulation cycle) and also the current perception events are extracted from the future events list. Lines 4 and 5 simulate the system in the current cycle by first calling the environment simulator and then calling all agents simulators. In line 6, the future events list is updated (step 4). The last two lines update the time and start a new cycle (step 5). `NewEnvSt` and `NewIntAgtSts` stand for the new environment state and the new internal states of agents.

We represent physical causality as a transition function, which takes an environment state and an event and provides a new environment state and a set of caused events. This function is specified as a set of reaction rules for the environment simulator in the form of

```
rrEnv( RuleName, Now, Evt, Cond, CausEvt, Eff)
```

with obvious parameter meanings. Agent behaviour, as a function from a mental state and a perception event to a new mental state and a set of action events, is also specified by a set of reaction rules:

```
rr( AgentName, RuleName, OwnTime, Evt, Cond, ActEvt, Eff)
```

For processing these rules we use two meta-predicates:

1. **`prove`**`( X, P)` where `X` is a list of atomic propositions (representing an environment state or an internal agent state) and `P` is a proposition.
2. **`update`**`( X, P, X')` where `X'` is the new state resulting from updating `X` by assimilating `P` (in our simple example this means asserting/retracting atoms).

When *E* is a current event, and there is an environment simulator rule, whose event term matches *E* such that **prove**(EnvSt, Cond) holds, then the specified CausEvt is added to the caused events list of step 2c) and the environment state is updated by performing

update(EnvSt, Eff, NewEnvSt)

In a similar way, the reaction rules of each agent are applied, updating its internal state by

update(IntAgtSt, Eff, NewIntAgtSt)

We now present the environment simulator:

```
1:   envSimulator( Now, CrtEvts, EnvSt, NewEnvSt, TranslCausEvts) :-
2:       findall( [CausEvt, Eff],
                  (
                  member( Evt/_, CrtEvts),
                  rrEnv( RuleName, Now, Evt, Cond, CausEvt, Eff),
                  prove( EnvSt, Cond)
                  ),
                  ListOfResults),
3:       extractEffects( ListOfResults, Effects),
4:       computeNewEnvState( EnvSt, Effects, NewEnvSt),
5:       extractEvents( ListOfResults, CausEvts),
6:       translateCausEvts( Now, CausEvts, TranslCausEvts).
```

In line 2 all events (and their accompanying effects) that are caused by an event from the CrtEvts list are collected in ListOfResults. Based on the effects of the current environment events (extracted on line 3) the new environment state is determined (line 4). After extracting also the caused events from ListOfResults (in line 5), their absolute time stamp is computed with respect to the current moment (line 6).

A similar procedure is performed for each agent:

```
1:   agtSimulator( AgtName, Now, CrtPercEvts, IntAgtSt,
                   NewIntAgtSt, ActEvts) :-
2:       findall( [ActEvt, Eff],
                  (
                  member( Evt, CrtPercEvents),
                  rr( AgtName, RuleName, OwnTime, Evt, Cond, ActEvt, Eff),
                  prove(IntAgtSt, Cond),
                  ),
                  ListOfResults),
3:       extractEvents( ListOfResults, ActEvts),
4:       extractEffects( ListOfResults, Effects),
5:       computeNewState( IntAgtSt, Effects, NewIntAgtSt).
```

5 Running AORS Models

In AORS, a simulation model is expressed by means of

1. a model of the environment (obtained from the external AOR model of the scenario), consisting of
   - a state structure model specifying all entity types, including exogenous event types
   - a causality model, which is specified by means of reaction rules

2. a model for each involved agent (obtained from internalizing the external AOR model of the scenario into a suitable projection to the mental state of the agent under consideration), consisting of
   - a mental state structure model
   - a behaviour model, which is specified by means of reaction rules
3. a specification of the initial states for the environment and for all agents

The environment and agent models can be defined visually by means of AOR diagrams. Also the initial states can be defined by means of instance diagrams (similar to UML object diagrams). The encoding of a simulation model by means of a high-level UML-based modelling language provides a platform-independent representation and allows the generation of platform-specific code automatically.

In the case of our Prolog simulation platform, we have to generate Prolog predicates from the AOR models. We also have to generate the reaction rules for specifying causality and agent behaviour in the format of the simulator.

Please consult the web page

`http://AORS.rezearch.info`

for obtaining further information about AORS and for downloading our Prolog AORS system.

6 Related Work

Agent-Based Simulation is being used in various research areas, today. In particular, it is being used in

- ***Biology***, e.g. for investigating eco-systems or in population ethology (especially with respect to ants and other insects), see, e.g., [Klü01];
- ***Engineering*****:** for analysing and designing complex (socio-) technical systems, such as Automatically Guided Vehicle Transport Systems [RW02];
- ***Economics*****:** e.g. in the simulation of auctions and markets, see *Trading Agent Competition* [TAC02];
- ***Social Sciences*****:** e.g. in [CD01] the phenomena of social monitoring and norm-based social influence and in [Hal02] the cooperation in teams is studied.

Some well-known platforms for Agent-Based Simulation are *Swarm* [Swarm00, Swarm96], SDML [MGWE98], *Sesam* [Klü01], *MadKit* [MadKit00] and *CORMAS* [Cormas01, Cormas00]. A particularly interesting class of simulation systems is formed by international technology competitions such as *RoboCup* [Robo98] and *Trading Agent Competition (TAC)* [TAC02]. Both RoboCup and TAC can be classified as interactive agent-based realtime simulation systems.

Although there is a large body of work on agent-based simulation, our AORS approach appears to be the first general UML-based declarative approach to agent-based discrete event simulation.

7 Conclusions

We have presented a general approach to modelling and simulating scenarios of interacting systems as multiagent systems, based on the *Agent-Object-Relationship (AOR)*

modelling language. Our Prolog implementation of an AOR simulation system is still in an early prototype stage. In the future, we will transfer it to the Java platform.

References

[Boo99] G. Booth: CourseWare Programmer's Guide, Yale Institute for Biospheric Studies, 1999.

[Cormas00] C. Le Page, F. Bousquet, I. Bakam, A. Bah, C. Baron: CORMAS: A multiagent simulation toolkit to model natural and social dynamics at multiple scales. In Proceedings of Workshop "The ecology of scales", Wageningen (The Netherlands), 2000.

[Cormas01] CIRAD: CORMAS, Common-pool Resources and Multi-Agents Systems, User's Guide, 2001.

[CD01] Rosaria Conte and Frank Dignum. From Social Monitoring to Normative Influence. *Journal of Artificial Societies and Social Simulation* 4:2 (2001).

[Dav02] Paul Davidsson. Agent Based Social Simulation: A Computer Science View. Jo*urnal of Artificial Societies and Social Simulation*, 5:1 (2002).

[Dav02] A. Davies: EcoSim: An Interactive Simulation, Duquesne Universität, Pittsburgh, 2002.

[Den71] D.C. Dennett: Intentional Systems. *The Journal of Philosophy*, 68 (1971).

[EW02] B. Edmonds, S. Wallis: Towards an Ideal Social Simulation Language, Manchester Metropolitan University, 2002.

[FG98] J. Ferber, O. Gutknecht: A meta-model for the analysis and design of organizations in multi-agent systems. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS´98), IEEE Computer Society Press, pp. 128–135, 1998.

[Hal02] D. Hales: Evolving Specialisation, Altruism and Group-Level Optimisation Using Tags. Presented to the MABS'02 workshop at the AAMAS 2002 Conference. Springer-Verlag, LNCS, 2002.

[HLA02]: Defense Modelling and Simulation Office: High Level Architecture, 2002.

[Jac94] I. Jacobson. *The Object Advantage*. Addison-Wesley, Workingham (England), 1994.

[Klü01] F. Klügl: *Multiagentensimulation*, Addison-Wesley Verlag, 2001.

[LTW04] J.v.Luin, F. Tulba, G. Wagner: Remodelling The Beer Game As An
Agent-Object-Relationship Simulation, submitted to ABS 5.

[MadKit00] J. Ferber, O. Gutknecht, F. Michel: MadKit Development Guide, 2002.

[MGWE98] Moss, Scott , Helen Gaylard, Steve Wallis and Bruce Edmonds, SDML: A Multi-Agent Language for Organizational Modelling, *Computational and MathematicalOrganization Theory* 4:1 (1998), 43–70.

[Robo98] I. Noda, H. Matsubara, K. Hiraki, I. Frank: Soccer Server: a tool for research on multiagent systems, *Applied Artificial Intelligence*, 12:2-3 (1998).

[MBLA96] N. Minar, R. Burkhart, C. Langton, M. Askenazi: The Swarm Simulation System: A Toolkit For Building Multi-Agent Simulations, 1996.

[Swarm00] P. Johnson, A. Lancaster: Swarm User Guide, 2000.

[TAC02] SICS: Trading Agent Competition 2002. See http://www.sics.se/tac/.

[Wag03] G. Wagner. The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behaviour, *Information Systems* 28:5 (2003), 475–504.

[WT03] G. Wagner and F. Tulba: Agent-Oriented Modelling and Agent-Based Simulation. In Jeusfeld, M.A. and Pastor, O. (Eds.), *Conceptual Modelling for Novel Application Domains*, volume 2814 of Lecture Notes in Computer Science, Springer-Verlag, pp. 205-216, 2003.