Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

MODEL-DRIVEN ENGINEERING OF SECOND-LIFE-STYLE SIMULATIONS

Gerd Wagner

Institute of Informatics
Brandenburg University of Technology
Cottbus, GERMANY

ABSTRACT

We present a *model-driven engineering* approach for developing Second-Life-style simulation scenarios that can be executed with SL/OpenSim. Our approach is based on the *Agent-Object-Relationship (AOR)* simulation language, which is a modeling language for expressing platform-independent simulation models that can be mapped to Java, JavaScript, PHP and SL/OpenSim code.

1 GENERAL INTRODUCTION

Model-driven engineering (MDE) is a software engineering approach that focuses on developing models rather than code, which can be generated automatically from computationally complete models. The Object Management Group (OMG) has proposed the *Model-Driven Architecture (MDA)*, see <www.omg.org/mda/>, as an MDE approach based on the fundamental idea that the chain of modeling goes from a conceptual *domain model* (called 'computation-independent model' in MDA) via a platform-independent *design model* to one or more platform-specific *implementation models* (one for each target technology platform), which can be directly mapped to code.

Since computer simulations are a particular class of software programs, simulation engineering can be considered a special case of software engineering. Like software engineering, simulation engineering can benefit a lot from developing conceptual domain models as the basis of platform-independent simulation models that are finally turned into executable simulation programs using some technology platform.

The standard view in the simulation literature (see, e.g., Himmelspach 2009) is that a 'simulation model' can be expressed either in a general purpose programming language or in a specialized simulation language. This means, that the term 'model' is used rather loosely both for low-level computer programs and for higher-level executable specifications. There is often no distinction between a conceptual/logical system model (expressed either as a non-executable conceptual model or as an executable specification in a high-level simulation language) and its implementation in some target technology platform. Clearly, as in software engineering, such a distinction would be important for several reasons: as opposed to a low-level computer program, a high-level simulation model would be more comprehensible and easier to communicate, share, reuse, maintain and evolve, while it could still be transformed into any platform-specific implementation code.

The *Entity-Relationship* and *Agent-Object-Relationship* simulation languages *ERSL* and *AORSL* are modeling languages for expressing platform-independent simulation models that can be mapped to Java, JavaScript, PHP and SL/OpenSim code. ERSL is a language for making basic discrete event simulation models. AORSL, which is a superset of ERSL, is a language for making agent-based discrete event simulation models. Both languages allow rule-based state-change modeling and include constructs for modeling the visualization/animation and the user interface of a computer simulation. A Java-based simulation management system (AOR-JavaSim) has been developed as an open-source project. A JavaScript-based Web service for AOR simulations is going to be offered by Simulario UG, a recent spin-off of Branden-

burg University of Technology (BTU), while a PHP-based simulation platform for multi-user online simulations is currently being developed in a project at BTU

This article presents first ideas of an approach to map AOR simulation models to a fourth target technology platform: the *Second-Life*-(SL-)based *SL/OpenSim* <http://SL/OpenSim.org>. We refer to this platform by *SL/OpenSim*. There are different use cases for such a mapping with respect to the role played by the SL avatar. The first use case is where the avatar just plays the role of an observer of a non-interactive simulation scenario that is rendered in front of his eyes. The second use case allows the avatar to interact with a simulation scenario that does not include any agents controlled by other users, while the third use case allows other agents of the simulation scenario to be controlled by other users who are represented by their respective avatar.

2 RELATED WORK

As a consequence of the rising interest in ontologies for the Semantic Web beginning around the year 2000, also simulation researchers working in different areas started to investigate the use of ontologies for simulation. In (Fishwick and Miller 2004), the authors report about two different efforts involving ontologies: a) the RUBE project aims at providing an XML-based simulation modeling framework supporting both 2D and 3D models, and b) the DeMO project aims at establishing an ontology for discrete event simulation. Both efforts can be viewed as attempts to establish a model-driven simulation engineering approach. As explained in (Silver et al. 2009), the main concern of DeMO is to support *ontology-driven simulation*. The starting point for the DeMO methodology is the conceptual model of a system obtained as the first step in the process of making a simulation model. This model has to be provided in the form of an OWL ontology. It is then mapped to an instantiation of the DeMO ontology, which is, in turn, mapped to an executable simulation model. Thus, the DeMO ontology constitutes a high-level simulation language supporting the paradigm of platform-independent modeling.

In (Benjamin, Patki and Mayer 2006), it is recommended to use domain ontologies in the simulation modeling process for making simulation models unambiguous and consistent. It is argued that in distributed simulation, ontologies may play the role of a vendor/platform-independent modeling language that facilitates the translation of models into the different simulation platforms involved in a distributed simulation.

So, while there are a number of proposals towards a model-driven approach to simulation engineering, we are not aware of any work on a platform-independent discrete event simulation modeling language that would allow to generate code for all kinds of target platforms from models. Neither are we aware of any work on generating code for simulations in SL and the SL/OpenSim. In (Crooks et al. 2009) it is suggested to use the SL/SL/OpenSim platform as an environment for exploring 'agent-based' simulation models. The paper reports on three models that have been ported to this environment: the well-known cellular-automata-style models 'Game of Life' and 'Schelling Segregation Model', as well as a simple pedestrian evacuation model. However, no general approach for making simulation models for the SL/SL/OpenSim platform is presented.

3 INTRODUCTION TO THE ER/AOR SIMULATION LANGUAGE

The *ER/AOR Simulation* framework, which is available from <www.AOR-Simulation.org>, was proposed in (Wagner 2004). It supports both basic discrete event simulation models without agents, also called *Entity-Relationship (ER)* simulations, and complex agent-based simulation models with agents having (possibly distorted) perceptions and (possibly false) beliefs, called *Agent-Object-Relationship (AOR)* simulations.

A simulation scenario is expressed with the help of the XML-based *AOR Simulation Language (AORSL)*. The scenario is then translated to Java source code, compiled to Java byte code and finally executed, as indicated in Figure 1.

Figure 1: From AORSL to Java byte code

Distinctive features of the *ER/AOR Simulation* framework are: (1) its high-level rule-based simulation language *AORSL*, (2) an abstract simulator architecture and execution model.

A ***simulation scenario*** essentially consists of a *simulation model*, an *initial state* definition and a *user interface* (UI) definition, including an initial state UI, a statistics UI and an animation UI. An ***ER simulation model*** consists of: (1) a set of *entity type* definitions, including different categories of *event* and *object* types; and (2) a set of *environment rules*, which define causality laws governing the state changes of the environment and the causation of follow-up events. An ***AOR simulation model*** consists, in addition, of a set of *agent* types, *message* types and *action event* types included in the entity type definitions.

An ***entity type*** is defined by means of a set of properties and a set of functions. There are two kinds of properties: attributes and reference properties. *Attributes* are properties whose range is a data type; *reference properties* are properties whose range is another entity type.

The upper level ontological categories of AOR Simulation are ***objects*** (including *agents*, *physical objects* and *physical agents*), ***messages*** and ***events***, as depicted in Figure 2. According to this upper-level ontology of AOR Simulation, agents are special objects; for simplicity it is common, though, to say just 'object' instead of using the unambiguous but clumsy term 'non-agentive object'. Notice that only objects, but neither events nor messages, have a state that may change over time.

Both the behavior of the environment (its causality laws) and the behavior of agents are modeled with the help of *rules*, which support *high-level declarative simulation modeling*.

3.1.1 Entity-Relationship Simulation

In basic discrete event simulation, which we also call *Entity-Relationship (ER)* simulation, we deal with two basic categories of entities: *objects* and *events*. A simulation model defines a number of object types and event types, each of them with one or more properties and zero or more functions (to be used for various kinds of computations). There are two different kinds of event types: those that define *exogenous* events (typically with some random periodicity) and those that define *caused* events that follow from the occurrence of other events.

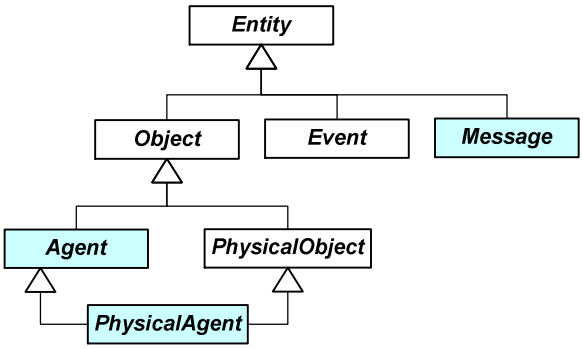

Figure 2: The upper-level ontological categories of ER/AOR simulation (agent-related categories in blue)

The state of the environment (i.e. the system state) is given by the combination of the states of all objects. Environment rules define how the state of objects is changed by (and which caused events result from) the occurrence of an event.

An ***environment rule*** is a 6-tuple

<WHEN, FOR, DO, IF, THEN, ELSE>

where: (1) the mandatory `WHEN` element denotes the type of event that triggers the rule; (2) an optional block of `FOR` elements allows to declare rule variables, such that each variable is bound either to a specific object or to a set of objects; (3) the optional `IF` element is a logical formula (allowing for variables) expressing a state condition; and (4) the optional `DO`, `THEN` and `ELSE` elements are containers for an optional `UPDATE-ENV` element specifying an update of the environment state followed by an optional `SCHEDULE-EVT` element specifying a list of resulting future events.

In each simulation step, all those rules are fired whose triggering event types are matched by one of the current events. The firing of rules may lead to updates of the states of certain objects and it may create new future events to be added to the future events list. After this, the simulation time is incremented to the occurrence time of the next future event (if no continuous changes have been defined for the given model), and the evaluation and application of rules starts over.

3.1.2 Agent-Object-Relationship Simulation

In the form of agent-based discrete event simulation, which we call *Agent-Object-Relationship (AOR)* simulation, we deal with three basic categories of entities: *objects*, *agents* and *events*. When we introduce agents, we have to make further distinctions between different types of events, as depicted in Figure 3. In particular, we need to consider *perception* events and *action* events in order to account for the perception-action cycle defining the foundation of agent behavior.

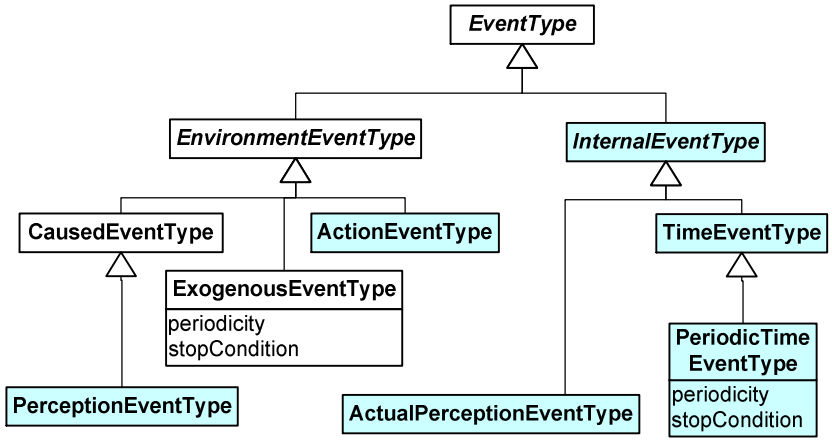

Figure 3: The different categories of event types (agent-related categories in blue)

An ***agent type*** is defined by means of: (1) a set of (objective) properties; (2) a set of (subjective) self-belief properties as well as an optional set of (subjective) belief entity types; and (3) a set of *reaction rules*, which define the agent's reactive behavior in response to perception events (and internal time events).

4 INTRODUCTION TO SECOND LIFE AND THE OPEN SIMULATOR

Second Life is a running *virtual world* launched in 2003, and a leading virtual world technology, developed by *Linden Lab*, and made freely available in the form of the open source project *SL/OpenSim*, the supporters of which include IBM, Intel and Microsoft.

4.1 SL Entities

Like AORSL, SL also makes a distinction between *objects* and *agents*. However, in SL an agent is not a special object that interacts with its environment, but rather, together with its associated *avatar*, it represents a human user. For avoiding terminological confusion, we will identify the SL term "agent" with "avatar". So, the basic entities in SL are avatars and objects, both of which are positioned on a region ("sim") that consists of land parcels, as depicted in Figure 4.

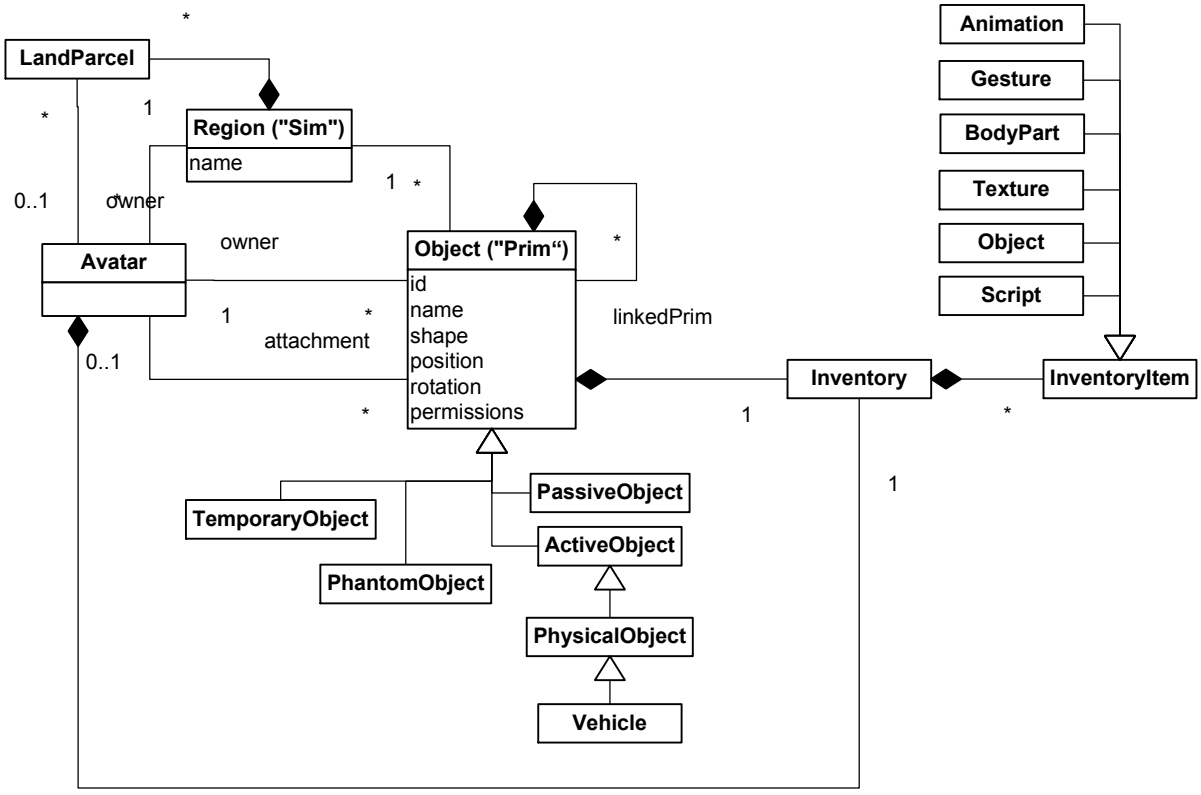

Figure 4: The main entities of SL

There are different kinds of SL objects: *active* objects, as opposed to *passive* objects, have behaviors defined by scripts written in the *Linden Scripting Language (LSL)*. *Physical* objects are active objects subject to the laws of physical kinematics and dynamics (rendered with the help of a physics engine).

Both avatars and objects have an 'inventory' containing items such as scripts, objects, textures, etc.

4.2 Communication

Avatars and active objects can communicate with each other via a mechanism for broadcasting simple string messages, called "chat". The distance that a broadcast message can be heard depends on the type of chat used (Whisper, Say, Shout, RegionSay). There is no support for point-to-point communication and for typed messages. However, using specific *channel numbers* (and possibly further filtering techniques), a kind of point-to-point communication can be achieved.

For receiving chat messages, one or more `llListen` actions, setting one or more filters, have to be performed first. After that `listen` events may occur, providing the chat messages received according to the current filter setting.

4.3 Perception Events and Timer Events

Active objects can perceive their environment, e.g. via events of the following types:

`sensor` – provides information on up to 16 objects/avatars found within a specified range; events of this type have to be triggered by invoking the active perception procedure `llSensor`
`no_sensor` – provides the information that there are no objects/avatars within the specified range
`touch` – provides the information that the object has been clicked by a user/avatar
`at_target` – occurs when a target position (set before with `llTarget`) is reached
`collision` – when a collision with another object occurs
`land_collision` – when a collision with land occurs
`timer` – occurs after some time span set before with `llTimer`

Alle these events can be handled in an object's script by providing suitable code to be executed in response to the occurrence of an event of such a type (such a section of code is often called an *event handler*).

5 MAPPING AOR SIMULATION SCENARIOS TO SL/SL/OPENSIM CODE

The main goal of this article is to discuss possible mappings from an AOR simulation scenario to suitable SL/OpenSim code, such that the mapped scenario can be run with SL/OpenSim. In this approach, the AOR simulation scenario represents a platform-independent model that can be transformed into various platform-specific models, including an SL/OpenSim model.

For gathering some first experiences, we have re-implemented two AORSL scenarios as SL/OpenSim scenarios. The first scenario represents a simple car traffic model with a one-dimensional circular space model. The second scenario is about bugs moving around in a grid space. This investigation is just a first step that has resulted in the preliminary mapping shown in Table 1. Further analysis has to be done before a first complete-enough transformation can be defined.

Table 1: Preliminary mapping

AORSL concept	*Corresponding SL concept*
Object	Passive object
Physical object	Physical object
Agent	Active object
Physical agent	Physical object
Out-message event	Broadcast (`say`) with suitable channel and reach
In-message event	`listen` with suitable filter settings
Physical object perception event	`sensor` with suitable filter settings
Collision event	`collision`
Periodic time event	`timer`
Reminder event	`timer`
Reaction rule	Event handler

Whenever we want to indicate the vocabulary from which a term comes, we use the XML namespace prefix syntax. For instance, "sl:agent" and "aors:agent" denote two terms having the same local name ("agent") but being distinct due to the fact that they are defined in two different namespaces: the Second Life vocabulary and the AOR Simulation vocabulary.

For creating the objects and agents of an AOR simulation scenario in an OpenSim world, a special OpenSim module has to be developed. This module would read an AORSL file and create corresponding sl:objects, with appropriate LSL scripts (defining their behavior), and add them to the inventory of a "master object" representing the simulation scenario. This master object is put in-world by the module,

where it listens to specific commands from an avatar for starting, stopping or resetting the simulation. On simulation start, the master object creates ("rezzes") the required sl:objects from its inventory.

6 EVALUATION AND OPEN ISSUES

Our first experiments with using the SL/OpenSim platform for implementing and running discrete event simulation (DES) scenarios have shown that this platform allows to deploy and publish DES models such that their runs can be visualized with 3D graphics and observed by any visitor of the region on which they have been deployed. The added value of using SL/OpenSim for publishing a DES model results from:

- The advanced 3D visualization techniques provided by SL/OpenSim
- The virtual world metaphor as an attractive publication channel for publishing research results and learning contents, especially when browser-based access of OpenSim worlds will be possible in the near future (through a combination of JavaScript, WebGL and the 3D context of the `canvas` element introduced in HTML 5)

In future work, we plan to consider two more use cases. The second use case are single-user participatory simulations where a user may interact with a simulation run via user interface events. In this case, the user interface events (such as clicking the mouse or pressing a key on the keyboard) have to be mapped to the action repertoire of an SL/OpenSim avatar. The third use case are multi-user participatory simulations where many users may visit our SL/OpenSim region and interact with a running multi-agent simulation scenario consisting of a number of passive objects and artificial agents. Here, the goal is to support the authoring of multi-agent simulation scenarios where users can freely choose the agent over which they want to take control. This requires a modification of the association between a user and her avatar, which is currently a frozen one-to-one association in SL/OpenSim.

Based on our initial experiments, we have identified the following issues that need further investigation:

- Typed messages, as used in AORSL, have to be emulated as string messages in LSL
- LSL does not allow to define object types with properties and functions; however, it allows to use an inventoried object as a blueprint that can be "rezzed" multiple times
- LSL does not support any discrete space model (such as two-dimensional grids, which is the most popular space model in social sciences)
- LSL does not allow to change the velocity of an object directly, but only via applying an impulse

7 CONCLUSIONS

We have shown how AOR simulation scenarios can be rewritten as SL/OpenSim scenarios. Our approach facilitates transferring basic and agent-based discrete event simulations to the SL/OpenSim platform. The most basic use case is the one where such simulations are provided within SL/OpenSim such that they can be started and observed by any user visiting the world under consideration. This could be useful for teaching. More advanced use cases involve realizing participatory simulation scenarios with SL/OpenSim, which is a topic for future work.

ACKNOWLEDGEMENTS

The author would like to thank Paul Fishwick for inviting this paper and to Steffen Tülling for implementing the SL/OpenSim scenarios.

REFERENCES

Benjamin P., M. Patki, and R. Mayer. 2006. Using Ontologies for Simulation Modeling. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, (eds.), *Proceedings of the 2006 Winter Simulation Conference*, pp. 1151–1159.

Crooks, A., A. Hudson-Smith, and J. Dearden. 2009. Agent Street: An Environment for Exploring Agent-Based Models in Second Life. *Journal of Artificial Societies and Social Simulation* **12**(4)10 <http://jasss.soc.surrey.ac.uk/12/4/10.html>.

Fishwick, P.A., and J. A. Miller. 2004. Ontologies For Modeling And Simulation: Issues and Approaches. In R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, (eds.), Proceedings of the 2004 Winter Simulation Conference, pp. 259–264.

Himmelspach, J. 2009. Toward a Collection of Principles, Techniques and Elements of Modeling and Simulation Software. Proc. of the 2009 International Conference on Advances in System Simulation. IEEE Computer Society, 2009, pp. 56–61.

Silver, G. A., K. R. Bellipady, J. A. Miller, W. S. York, and K. J. Kochut. 2009. Supporting Interoperability Using the Discrete-Event Modeling Ontology (DeMO). *Proceedings of the 2009 Winter Simulation Conference (WSC'09)*, Austin, Texas, December 2009, pp. 1399–1410.

Wagner, G. 2004. AOR Modeling and Simulation – Towards a General Architecture for Agent-Based Discrete Event Simulation. In *Agent-Oriented Information Systems*, Lecture Notes in AI, volume 3030, pp. 174–188. Springer-Verlag.

AUTHOR BIOGRAPHIES

GERD WAGNER is Professor of Internet Technology within the Department of Informatics, Brandenburg University of Technology. His research interests include agent-oriented modeling and agent-based simulation, foundational ontologies, (business) rule technologies and the Semantic Web. In recent years, he has been focusing his research on the development of an agent-based discrete event simulation framework, called *AOR Simulation*. He can be reached at <http://www.informatik.tu-cottbus.de/~gwagner/>.