EnvEdit - A GRAPHICAL ENVIRONMENT EDITOR
Approaches towards a Fast and Simple Way of Building Complex Environments in Multi Agent Simulations

Christoph Schwarz[1], Jan Busch[2], Carsten Beth[1], Jürgen Sauer[2] and Axel Hahn[2]
[1] *OFFIS - Institute for Information Technology, Escherweg 2, Oldenburg, Germany*
[2] *Department of Computing Science, Carl von Ossietzky University, Ammerländer Heerstrasse 114-118 Oldenburg, Germany*

Keywords: Multi agent systems, MASON, Graphical editors, Simulation.

Abstract: Multi Agent Simulations are well known and widely-used simulations which can simulate embodied agents in neighbourhoods or environments. The designing of such environments has, so far, mainly to be done by hard coding them directly in the simulation source code.
In this paper we present EnvEdit, a graphical environment editor for MASON. EnvEdit leads to a less time consuming and less error-prone design of environments.

1 INTRODUCTION AND MOTIVATION

The simulation of complex processes is increasingly performed with the help of multi agent systems (MAS). A Java Framework for multi-agent simulations in which agents, at least in part, can represent physical entities is MASON - Multi Agent Simulation Of neighbourhoods ((Balan et al., 2003), (Luke et al., 2005)). The framework allows you to create complex simulations by providing a few base classes from which the agents inherit then. In the design of such a simulation there is often a distinction between the acting agents of the simulation and the environment. In more complex cases, the environment is also represented by agents (such as warehouses, ports, etc.) which in addition to their position also have properties (such as size, ceiling height, storage capacity, etc.) and which can collect data for statistic analyses (like goods throughput).

Creating such a complex environment is effected usually in two steps. In the first step, all agent types that represent environmental elements (such as warehouses, roads, Bridges) are written and in the second step the environment will be composed out of these agent types. Therefore the appropriate agents first are created and then placed in a so-called *SparseGrid* with the specification of coordinates. Environments which are composed from a large number of agents are only very uncomfortable to create since there is no visual feedback during the process.

In our research on intelligent and efficient means of transport for intralogistics we need to design a lot of slightly different warehouses (see (Beth et al., 2009),(Ommen et al., 2009), (Ommen et al., 2010)). In MASON this has to be done by hard coding these environments (the warehouses) in the source code. This process was time consuming and error-prone. In addition, it was difficult to create the environments and the simulation clearly seperated. Such a separation is necessary if the environments and the simulations are to be created by different persons. To get rid of these drawbacks we would like to have a graphical editor for the environments we need for our simulations.

2 MASON

MASON is a discrete-event multi agent simulation library core written in Java. It works through advancing in simulation time in discrete steps. Per step, all the agents will be activated once and can perform actions. The behaviour of the agents is determined by the Java class which represents them.

A central concept of the simulated environment in MASON is the so-called *Grid*, which provides an array of fields, which can take different values. MAS-

ON offers simple grids that store only numeric values, but these cases are not of interest for our approach. The interesting grids for EnvEdit manage program objects by letting the fields contain one or several references to program objects. These program objects are the previously introduced agents and they are placed on start up on some fields of the grid. The agents can then interact with the grid and with each other. The interaction of the agents may each be considered as a communication that takes place with the help of the grid and mutual function calls. The grid allows the agent via a neighbourhood function to determine which fields are adjacent to their own field and whether or which agents are on it. Dynamic agents can move over the grid, by go on an adjacent field. Internally, the reference to the grid object representing the agent, is moved from one field of the grid to another. Via the grid, all fields, and thus all agents are reachable and available for communication.

3 EnvEdit

With the help of the developed software system, EnvEdit, it is possible to create a variety of environmental models. It is possible to save, load, and modify the created models. The section of the model which is shown can be zoomed, moved and reset with mouse and keyboard interactions. The properties of the currently selected environment object can be inspected and changed. Environment objects can be created, selected and moved. It is also possible to move the object up and down in the third dimension (to change the ordering of the objects). The environment object types available for the design of the environment model can be modified with comprehensive editing capabilities of the object descriptions.

If an environment model should be imported into a simulation system, the EnvEdit library has to be added to the project. The functionality of converting an environment model into a grid usable for simulation is implemented in this library. If a model was converted into a grid, it can be used as usual in MASON. The generated grid objects makes it possible to access the environment objects which created them.

The class BaseGridObject serves as a base class for all grid classes, which will be used during the conversion of the environment-model in a MASON-Grid. BaseGridObject is a MASON agent and defines functions which can be used to access the system properties (like unique identifier (ID), name, position and dimensions (width and height)) of the underlying environment object. Before a model can be loaded, all used environment object types must be assigned to an existing grid-class. On the part of the modelling system, this is done by editing the object descriptions. Each object description has a class-property, a string whose value determines the name of the associated grid-class. On the part of the simulation system, a class with the appropriate name has to be created for every such environment object type. These classes are derived from BaseGridObject, the upper class for all grid objects. Any required functions for objects of that type can then be implemented in this grid-class.

4 DESIGN OF THE SYSTEM

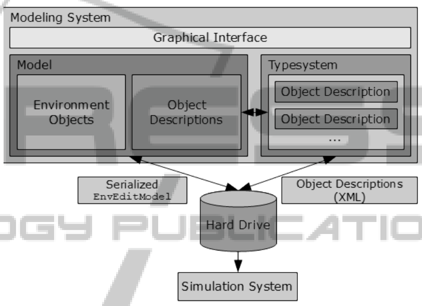

Figure 1: EnvEdit Overview.

Figure 1 shows an overview of the structure of Envedit. We will now discuss the parts of EnvEdit in detail.

Interface. The user interface of EnvEdit is a standart Java/Swing user interface with the usual mouse/keyboard support.

Type System. The type system determines how objects are associated to Java classes. Since the editor should be able to handle previously unknown types of environmental objects and since the objects simultanesouly should lead to different object types in MASON, a flexible type system was developed.

EnvEdit uses XML to save object descriptions. XML was used to ensure the modelling of environments completely without the need to produce Java code. The properties which are in the object description can later be read directly into the simulation. The property ”class” specifies which Java class has to be used in the simulation for this environment object type. The developer of simulation and the environment must therefore ittle only on the XML object descriptions. All other steps are then carried out independently.

The requirement for a simple extensibility implies that a one-to-one relationship between environment object types and program object classes is not an option. So it had to be resolved, how a description of the object properties in an XML file may lead to specific program objects. From the type of object descriptions arises that the environment objects only differ in their type-bound properties, which can be considered as "identifier" - "value" pairs. Different functions of the environment objects need not be depicted, because these as well as the specific meaning of a property's value, only play a role in the simulation system. Therefore a basic object class was designed whose instances manage a list of properties. The creation of an environment object of a particular object type creates an instance of this class with the set of properties, specified in the description of the object. The values of these properties can be accessed by certain functions which take the identifier of the desired property as a parameter. Therefore in EnvEdit all the available environment object types of the modelling are program objects of the same class.

Model. The type system defines how the environmental objects are associated to program objects during the conversion. The model, however, describes, in what way EnvEdit itself manages the environment.

An environment model is managed as a tree-like structure. Environment objects can contain other Environment objects, thereby hierarchical relationships can be mapped. Environment objects that are registered in such a list at the top are covered by the following registered environment objects. Thus, an order can be realized in the third dimension, and thereby spatial relationships can be modelled. The basic structure of the data is therefore a tree of objects. The root of the tree is the *EnvEditModel* object that manages a set of *BaseObject* objects. Each BaseObject can manage again a set of BaseObject objects, so that the creation of a tree structure is possible.

In addition to managing BaseObject objects this program object serves as a location for the entire data concerning the given model. Besides the environment objects the EnvEditModel also possess a collection of *ObjectDescription* objects, which describe the object types used in this model. Object descriptions are stored in the model, because it is the only way of ensuring that the model can be edited on any system. If a model is loaded on a system, the previously unknown object descriptions are added to the repository of the system, so that an exchange of object descriptions is possible with the help of the model.

5 CONNECTION TO THE SIMULATION SYSTEM

This section presents how the environment models and the MASON simulation are connected. Figure 2 illustrates this process schematically.

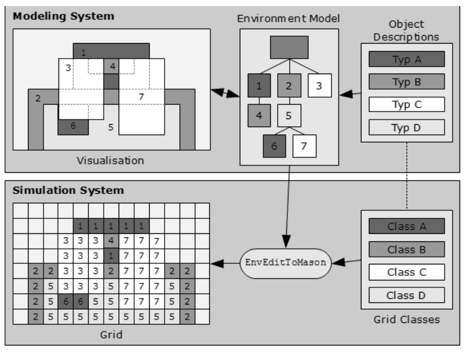

Figure 2: Transformation of a environment model in a MASON-grid.

The environment model and the necessary object descriptions for generating the environment objects are seen at the top of the figure. The environment model is here represented once by the data structure (middle) and once by the visualization which was generated by the modelling system (left). The numbers on the objects are used to associate visualization and data structure and correspond to the unique identifiers. The transformation of the model into a MASON-Grid is made by the functions of the EnvEdit library. These functions scan the tree of the environment objects and assigns each field in the grid to an grid object which corresponds to the environment object type (bottom left, the colors correspond to the used grid-class). The order of the environment objects is preserved. The environment objects which were obscured by overlying objects have no influence on the Grid. Grid objects are instances of the grid-classes that implement the major functions of the environment objects which are mapped in the environment model. The created grid objects maintain a reference to the environment objects which produce them (represented by the numbers), and can thus access all registered property values.

The environment objects created in the modelling-system must be mapped onto the grid to be used as agents by the simulation system. In the model, two environment objects of different types are distinguished by their different set of properties and by the object description which created them. But from the programming point of view they are two instances of the same class. In the modelling system this is intended and causes no problems. In the simulation system, the various objects in the simulation have different properties and different functions. Should a single grid-class be used for all the environment objects according to the procedure in the modelling-system this class must have all the features of all existing environment object types included. This single grid-class would have to be adapted to any model that includes a new set of environment object types. There is therefore little point in bringing the environment objects directly on the MASON-Grid. Instead, environment objects with different types should produce different grid objects. To facilitate this, each environment object type of the model is assigned a class in the simulation system. As described, the object description provides an entry for the class of the type in the simulation system. This property of an environment object type is used during conversion to instantiate an appropriate grid-object and for position it in the grid.

The environment model is, as described, designed as a tree whose root is the EnvEditModel-object. The different environment objects are stored as BaseObject instances and form the nodes and leaves of the tree. Important in this arrangement, except the depth of the environment objects in the tree, is the sequence of individual children of a node. Both factors, depth and order are crucial in order to understand the arrangement of the environment objects in three dimensions. Nodes which are located deeper in the tree hide all their direct predecessor. Sibling nodes and their children form subtrees. Their order in the common parent node is interpreted as follows: subtrees which are placed further back (added later) hide subtrees which are located further forward.

During the transformation of the environment model into a MASON-Grid, the just described tree of objects is traversed by a depth-first search. If a node has several sibling nodes the search begins with the most recently added.

Position and dimensions of the environment object are projected onto the grid, and the fields covered by it are traversed gradually. If there is already a grid object on a field, it will be skipped and proceed to the next. If the field is still vacant, the class property of the environment object is read, and an instance of the specific grid-object-class is created and attached to the field. In this way maximum flexibility in the development of the simulation system is provided.

For instantiation of grid-objects a mechanism of the Java programming language is used which is inter alia called *Dynamic Class Instantiation*. The required classes must be designed by the developers of the simulation system in cooperation with the developers of the environment models, since these are directly related to the object descriptions used in the modelling. Object descriptions define the data which an environment object contains, the grid-classes define the functions of the corresponding program objects in the simulation. The dynamic class instantiation is used to create a program usable class out of the string containing the class property of the environment objects. The so-used classes must have a common parent class, so they can be used by the programming language in this way. In the case of EnvEdit, this is the *BaseGridObject*.

The class BaseGridObject defines the basic functions of all grid classes. These functions can access the system properties like the x- and y-coordinates, the height and width and the type of the underlying environment object directly. All other properties can be accessed through the underlying environment object.

6 CONCLUSIONS AND FUTURE WORK

EnvEdit is a flexible tool for the design of two dimensional environment models for MASON. The flexible type system allows it to respond to as yet unknown requirements, and to map new environment objects. EnvEdit allows a clear separation of the model data from the simulation system, and facilitates the development of different environment models. The operation of the software system has been kept simple. The environment data is transferred into objects which are usable for the simulation.

In the future we want to allow the creation of so-called graphs (which describe, for example, road networks) directly in the environment model.

Another possible future work is the use of EnvEdit for the design of environments for different frameworks than MASON. The flexible type system and the internal model could make it possible to easily adopt the integration into the MASON framework for other MAS-libraries.

REFERENCES

Balan, G. C., Cioffi-Revilla, C., Luke, S., Panait, L., and Paus, S. (2003). Mason: A java multi-agent simulation library. In *Proceedings of the Agent 2003 Conference*.

Beth, C., Kamenik, J., Ommen, D., and Hahn, A. (2009). Design aspects of cognitive logistic systems. In *2nd International Conference on Dynamics in Logistics*.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81:517–527.

Ommen, D., Beth, C., Kamenik, J., and Hahn, A. (2009). A system-architecture for robotic movements of goods - approaches towards a cognitive material flow system. In *ICINCO-RA*, pages 342–347.

Ommen, D., Kamenik, J., Beth, C., Busch, J. C., Kulas, A., Cramer, E., and Hahn, A. (2010). An adaptable architecture for intelligent conveyors. In *ICINCO (1)*, pages 249–254.