DEVELOPING VISUAL-INTERACTIVE SIMULATIONS WITH JAVA AND VRML

André M.C. Campos
Blaise Pascal University / LIMOS
Computer Science & Modeling Institute
Campus des Cézeaux B.P. 125
AUBIERE Cedex 63173 FRANCE
E-mail : andre@isima.fr

KEYWORDS
Visual Interactive Simulation, Java, VRML, Web-based Simulation, Multi-Agent Paradigm, Discrete Event Simulation.

ABSTRACT
The aim of this work is to provide a tool for the development of Web-based simulations with visual and interactive features. The tool is defined by an object-oriented class library written in Java also providing a structured interface for VRML code. This library uses a discrete event simulation approach in order to assign and schedule events in a Multi-agent based environment. An application example is also presented showing the underlines of the use of the library.

INTRODUCTION
With the current explosion of Web technologies in the last years, peoples have turned their attention for new software paradigms involving the net. This is notably true in the simulation domain, where its community began to use the Web as a tool enabling distributed and remote visualization and interaction with a simulation system (Fishwick et al. 98). In this way, rather than just change messages between themselves by the net, the simulation community came able to work together in a visual and interactive simulation environment (VIS) even if they are not locally present and even if they do not have the simulation itself installed in their machine.

This Web explosion is due mainly to the facility for get and work with information. For instance if we focus on the simulation development process, we see that for an available simulation running on the net :

- remote end-users can test and provide its validation feedback for the developer at run time;
- there is no more need of dependent-machine recompilation and installation when exporting simulation code;
- there are no physical distances to break or interrupt the development, modeling and validation process.

This approach contrasts largely with using a traditional simulation language like Simula or QNAP2 which are defined to work locally. Today there are increasingly Web based simulation environments available (McNab and Howell 96; Zeigler 97; Buss and Stork 97; Healy and Kilgore 98). Most of then are Java versions of existing simulation languages, so all implement different flavors of simulation.

In order to facilitate the development of simulations in the Web using a discrete-event approach and dealing with individual-based simulation data, we have implemented a set of Java classes named JMavis – Java Multi-Agent VIS. The library uses the Web-based languages Java language and the Virtual Reality Modeling Language – VRML – allowing simulations to be incorporated into Web pages and to be executed remotely.

LIBRARY CHOICES
The discrete event and individual-based approach chose for the library aims to facilitate dynamic changes in the simulation model since the latter are applied locally into each different simulation entity (Hill 96). Spatial effects and concurrent process are taken in this approach as well. So, in a simulation system using JMavis, each simulation entity is conceived as being operationally autonomous, havig its own state variables and performing tasks in parallel with other entities. This characteristics leads us to the Multi-Agent Simulation idea where it is possible to represent an environmental phenomena as the consequence of the interactions of parallel agent sets. Each agent has its own operational autonomy, encapsulates and performs its individual tasks locally, but influencing the global behavior of the simulation (Wooldrige and Jennings 95).

In addition for the ability to allow applications running remotely, the Java language also incorporates some languages features necessary for simulations using the Multi-agent paradigm. Firstly, it is a full object-oriented language and most aspects of agent-oriented simulations systems can easily be embedded within object-oriented discrete-event simulation approaches (Uhrmacher 97). Secondly, it
supports the specification of multiple threads allowing object operations to be carried out in parallel, i.e. concurrently (Gasser and Briot 92).

Likewise, the VRML language gives us an useful tool to construct Web-based environments. In reality, it is a description files format portable on the Web capable of representing static and animated dynamic 3D objects. Thus, it is used to provide visual-interactive features for simulation entities in a persistent way (i.e. by a file). Indeed, its utilization imply the use of a VRML browser or a plug-in in a HTML browser which provides already all visual and interactive features necessary for the simulation visualization skipping the developer of the hard task to implement graphical interactive and rendering routines.

The library provides then a structured way to integrate simulation entities, specified in the Java language, with their visual aspect, described by VRML files. As consequence, the end-user has three-dimensional feedback of simulation results as well as a virtual environment where he can interact changing the simulation model dynamically.

ARCHITECTURE

The classes of the library is defined in accordance with an abstract framework which conceives an interactive simulation environment as a system with three main group of classes and objects: a simulator group, to control and schedule parallel events in a discrete event simulation approach; a user-interface group, to provide user input/outputs; and a world group, to represent the environment to simulate, i.e. the system structure possessing all the entities with their respective spatial localization. The simulator and world composes together a set of classes and objects, with their respective relationships, that we call the Agent-based Simulation Model. A second model, encapsulating the user-interface group, is incorporated to provide visual-interactive features to the simulation. This model, named Visual-Interactive Model, manages bidimensional views by bitmap image presentations as well as tridimensional views by VRML specifications. The Figure 1 presents these groups of classes as well as the models which incorporate them.

The Visual-Interactive Model works like a supervisor, getting the current simulation state to give user outputs and changing the current simulation state by user inputs. The user interface components interacts with the simulator components setting its configuration, as the step time in a clock-based discrete event simulation. It also interacts with the world components changing environment attributes, as an entity position or state.

CLASS PACKAGES

Each one of the models described above is implemented as an individual Java Package. They are then a collection of related classes and interfaces with theirs respective access protection. The Visual-Interactive Model is implemented in the Vis package and the Agent-based Simulation Model in the Sim package. A new simulation application must either use the classes of these packages to create its object instances or create instances of classes that inherit from the latter classes.

Focusing on the Sim package, it is composed by the classes Simulator, World, Entity, PassiveEntity, ActiveEntity, Event, EventList, and the Behavior interface. We will describe individually all of these ones as well as the communication rules between themselves.

The simulation kernel is represented by the Simulator class, being a Java Thread type where its $\text{run}$ method uses the time to activate simulation active entities, and to sort and schedule environmental events. In order to give a visual continuous feedback of the simulation results, i.e. an animation, the $\text{run}$ method is implemented using a clock-driven approach to schedule events (Hill 96). Other approaches may be implemented by just subclassing the Simulator class and redefining its $\text{run}$ method. Indeed this method is a infinity loop executing in background which activates (sending messages) to all simulation active entities according to the clock time.
The World class implements the Singleton pattern (Gamma et al. 95) having in this way an unique instance of itself easily accessible from any entity. It provides the physical localization of each entity running under the simulation managing conflicts of parallel spatial competition. Figure 2 shows the relationship between the classes Simulator, World and ActiveEntity (representing the active entities running on the simulation).

![Class diagram of main classes of the sim package (UML notation)](image)

Figure 2 – Class diagram of main classes of the sim package (UML notation (Booch et al. 96)).

Every abstract or physical entity under the simulation is an instance of an Entity subclass. The later class is broken into two different types of entities: the active ones and the passive ones, represented respectively by the classes ActiveEntity and PassiveEntity. The passives entities have the sense of resource, i.e. they just exist to model entities that are modified by others, and they represent stimulus or constraints for active entities. The latter entities, in contrast with the passive one, create the simulation dynamics by performing tasks and generating events in the environment. Active entities are conceived in the library as reactive agents (Ferber 95) running in the simulation. They interact with the World instance consulting the environment besides itself.

ActiveEntity instances are defined as threads running in parallel. Each one has two possible states: either it is sleeping either it is active. The transition from the sleeping state for the active one is boosted by the Simulator object which wake up the entity when a new event has arrived for it. When it is active, it uses an instance of a Behavior subclass in order to create its actions. In fact, Behavior is an Java Interface, as so it just declares the run method that active entities call to create events. The implementation of this method is done by Behavior subclasses which the client application must supply. The latter overload the run method implementing theirs own algorithms. In this way active entities can change dynamically theirs behaviors just by changing the instances which the run method is called. This approach is showed in Figure 3 and it is an implementation of the Strategy pattern described in (Gamma et al. 95). As far as the entity has create its events it came back to the sleeping state. Events are no more than simple descriptive codes which are interpreted by the active entities. They are stocked into the event list, implemented by the EventList class.

![Class diagram representing simulation entities as well as the behavior design for actives entities. (UML notation)](image)

Figure 3 – Class diagram representing simulation entities as well as the behavior design for actives entities. (UML notation).

The entities in the client application are defined as instances of classes that inherit from Active Entity or Passive Entity according to their role in the simulation. Each subclass represents a particular kind of entity (with its own features), and each instance is an individual element of this type of entity. So it is possible to:

- Define news type of entities by creating new classes;
- Define sub-species by inheritance and modification of the default behavior of their individuals.
- Define an individual differentiation among the entities by allowing a specific instantiation.

The simulation itself, modeled by the sim package, works independently of the user-interface. Indeed this package does not provide any visual-interactive service for the simulation. The vis package is then coded to despite this user-interface fail providing classes to give input/output features over the classes in the sim package.

The vis package is designed to give both 2D and 3D visual and interactive feedback. Bidimensional approach is done only with using the AWT toolkit included in the standard JDK distribution, and tridimensional feedback is given using the External Authoring Interface (EAI) API. The latter is used for communication between a VRML scene and an external element as a simulation applet developed with the JMavis library. So simulation entities implemented into the simulation applet can be represented outside the applet itself, specifically into a VRML world.

There are few classes composing the vis package. The main ones are the classes VisualSimulator, View and Viewer and its subclasses ImageViewer and VrmlViewer. The VisualSimulator class is an applet subclass providing some basic applet tasks as init, start, stop and destroy, as well as providing some methods for simulation control as pause, play,
goStepByStep and restart. It encapsulates end-user access for the Simulator class described in the sim package.

The class Viewer is an abstract class that controls user input/output over the elements of the simulation. Its subclasses VrmlViewer and ImageViewer implement the methods to provide 3D and 2D visual feedback respectively. Figure 4 gives an outlook for these classes.

Each entity running under the simulation has a “visual” object associated with it. The latter is modeled by the View class. A View instance may represent either an image bitmap (by the ImageView subclass) or a Vrml description file (by the VrmlView subclass) depending on the current Viewer object used by the VisualSimulator object. This kind of separation between data and data view is suitable for visual-interactive systems because it increases flexibility and reuse of system components. For instance the same visual description file (bitmap image or Vrml file) can be shared by different entities, all ones giving the same look-and-feel. In the other hand an entity could be associated with more than one visual description file giving in this way different aspects (2D and 3D) for the same entity.

APPLICATION EXAMPLE

A simulation application was made in order to validate the library. This simulation creates a underwater environment where preys (fishes) and predators (sharks) live together. Therefore there are no compromise to any real and concrete environmental case. Its goals only consists in providing an example of library use. It is a Vrml implementation of the simulation presented in (Campos and Hill 98).

Actually, the library use is facilitated by object-oriented features intrinsic of the Java language. Modeling a new simulation system using the JMavis library involves mainly extending the ActiveEntity and PassiveEntity classes to include its simulation entity proprieties and behavior. The excerpt of code in Figure 5 exemplifies the extending class procedure.

For each instance of the Fish class, explained above, an associated VrmlView instance is created. The same is true for shark entities. Figure 6 shows the output given by the application.

![Figure 6 - Screendump of an application example using the JMavis library. Some applet GUI controls entity attributes until Vrml Browser controls interactive visualization.](image-url)
CONCLUSION

We have described the gains of using the Web technology in the development of visual-interactive simulation environments, specifically when using Java and VRML languages. The design of the JMavis library was described as well presenting its abstract architecture and its class design. The library was developed into two different packages: one providing classes to facilitate the development of a simulation application, i.e. the kernel structure for an individual-based simulation, and the other one providing visual-interactive service over the last one.

The application example implemented to validate the library have showed us some drawbacks regarding the animation of results into the Vrml scene. There is a visual inconsistency in animation feedback because the clock-driven approach implemented in the Java package works independently from the event time generated by the Vrml browser. As consequence, entities may look like doing a different thing what they are really doing. In the application example the shark can gives the impression that it will turn to right while it goes ahead.

Despite these drawbacks, easily found in virtual reality environments, JMavis is hoped to be a useful tool for development of Web-based simulations. Future works points to resolve the visual inconsistencies providing more realistic and natural animations, as well as to grow up the library to support cooperative and distributed simulations.

ACKNOWLEDGMENTS

The author would like to acknowledge the CAPES (Brazilian Research Council) for his doctoral fellowship and Y. Wanwanscappel and A. Momboise for their contribution in the example implementation.

REFERENCES


