ION Framework - A Simulation Environment for Worlds with Virtual Agents

Marco Vala, Guilherme Raimundo, Pedro Sequeira, Pedro Cuba, Rui Prada, Carlos Martinho, and Ana Paiva

INESC-ID and IST - Technical University of Lisbon
marco.vala@inesc-id.pt

Abstract. Agents cannot be decoupled from their environment. An agent perceives and acts in a world and the model of the world influences how the agent makes decisions. Most systems with virtual embodied agents simulate the environment within a specific realization engine such as the graphics engine. As a consequence, these agents are bound to a particular kind of environment which compromises their reusability across different applications. We propose the ION Framework, a framework for simulating virtual environments which separates the simulation environment from the realization engine. In doing so, it facilitates the integration and reuse of the several components of the system. The ION Framework was used to create several 3D virtual worlds populated with autonomous embodied agents that were tested with hundreds of users.

Key words: Virtual Environment Simulation, Agent-Based Modelling, Embodied Agents, Guidelines

1 Introduction

A virtual agent is by definition an entity that senses, reasons and acts within an environment. Usually, these agents have some components related to their behaviour, referred as the *agent mind*, and components related to their embodiment in a particular environment, referred as the *agent body*.

The environment is often simulated by the *realization engine*[1] that keeps a part of the simulation model while it provides a graphical representation to the elements in the simulation, including the agents. The *realization engine* plays the role of virtual space inhabited by the agents.

However, these *realization engines* do not provide the appropriate abstraction level required by the agents. For this reason, the minds are generally dependent on the environment and on a particular form of embodiment which, in turn, leads to components which are difficult to reuse across applications.

We argue that we need a set of guidelines to make components less interdependent thus fostering reusability across different applications. Our approach

[1] By realization engine we refer to a set of components that usually includes the graphics engine and the physics engine.

separates the simulation environment from the *realization engine*. We named this simulation environment ION Framework[2].

The remainder of the paper is organised as follows. First we look at related work. Then, we describe the ION Framework and depict the guidelines of our approach using a concrete example. Then we present some other case studies to further demonstrate our results. Finally we draw some conclusions and outline future work.

2 Related Work

Several tools exist to aid in the development of autonomous embodied agents. Some of these tools support the development of the mind like Soar [11], JAM [7] or JESS [5]. Researchers frequently adopt game engines like Unreal Tournament [4], Source [16] or Ogre [14] for the embodiment.

Recent efforts from the virtual agents community lead to the SAIBA initiative [13] which specifies a framework for creating Embodied Conversational Agents (ECA). This framework lead to the definition of the FML (Function Markup Language) and BML (Behaviour Markup Language), which are an effort to provide a common language between the several components that form an ECA. There are currently systems[3] that use these markup languages like SmartBody [15] or ACE (ARticulated Communicator Engine) [10].

However, for most of these tools, the simulation environment depends on the application being developed. The components are integrated in a common environment but become dependent on each other. A good example is the use of game engines. The components around, like the agent's mind, usually become too dependent on how the game engine handles the embodiment and the environment, which reduces reusability across other applications.

On the other hand, we can borrow some ideas from tools to create agent-based simulations. Some provide conceptual frameworks and templates for the design and implementation of agent-based models, like Swarm [12] or JADE [8]. Other provide a complete simulation environment which is suitable for rapid development of prototype models, like NetLogo [17] or Breve [9].

3 ION Framework

Broadly speaking, a simulation using the ION Framework consists of a set of *Elements* whose state changes in a discrete manner over time when the simulation is updated.

Furthermore, to regulate the interactions between the several elements, the framework enforces a set guidelines which are applied to all the intervenients[4]

[2] The ION Framework is open-source under GNU LGPL license.

[3] For a complete list of tools and components within the SAIBA initiative go to http://wiki.mindmakers.org/projects:bml:main (updated in April 2009).

[4] By intervenient we mean any entity that intervenes in the simulation (either internal or external to the simulation).

in the simulation. These guidelines are: 1) coherent access to information, 2) mediation of conflicts, 3) active and passive gathering of information, and 4) dynamic configuration changes.

In order to introduce the framework and the guidelines behind it, we will present examples of FearNot! [1] which is an application aimed at reducing the bullying phenomena in schools. Several virtual characters interact with each other in a 3D environment throughout several episodes (Figure 1). Each episode portraits a situation involving a conflict between a victim and a bully. The user behaves like a friend of the victim and gives advices about the way the victim should solve its problems.

Fig. 1. FearNot!

3.1 Coherent Access to Information

Picture an example of two agents from FearNot!, John (the victim) and Luke (the bully), that decide to move depending on the position of each other. At each update cycle John looks at Luke and if he is too near John backs away. On the other hand, Luke tries not to be very far from John. So he looks at John and if he is farther than a given distance he moves closer.

Consider that John is the first to decide. John looks at Luke and since he has not moved yet they are still at a comfortable distance. Given this, John decides not to move. Next, Luke looks at John and he decides to move closer. Only Luke moves.

Now consider that Luke is the first to decide. As previously, he decides to move closer. But, when John looks at Luke, Luke is now too close because he moved closer. Thus, John decides to move away. The final outcome is that both agents have moved.

Notice that although the initial simulation state was the same for the two cases the outcome was different. The outcome depends on whom decided first and we believe this is undesirable in some situations.

The ION Framework provides *coherent access to information* and guarantees that all the intervenients get the same information if they do the same query to the simulation at a given instant. The agents always get the information as it was at the end of the last update cycle regardless of subsequent changes that will be carried out in the next update.

Therefore, all modifications to the simulation state in the ION Framework are not immediately carried out. We can schedule *Requests* on *Elements* which are handled at a later time during a specific phase of the update cycle denominated *Process Requests Phase*.

In the previous example, John and Luke were modelled with the ION Framework. Thus the outcome is always the first situation (only Luke moves) regardless of which agent decides first. This is due to the fact that even if one agent decides to move it will in practice only schedule a *Request* to do so. Therefore, both agents look at the same state of the simulation even if one of them has decided to change it before the other.

3.2 Mediation of Conflicts

In the ION Framework, changes to the simulation state are performed synchronously. As we have seen it ensures coherent access to information. But it also implies that intervenients are seen as acting simultaneously at each update cycle. Therefore, conflicts may arise between actions that try to modify the same portion of the simulation state at the same time.

An example from FearNot! happens when Luke and Paul (the bully assistant) decide to push John at the same time. We can have several outcomes: 1) John is pushed by Luke, 2) John is pushed by Paul, 3) John is pushed by both of them with their combined strength, etc.

The ION Framework ensures the *mediation of conflicts*. The previous setting corresponds to scheduling two push *Requests* on the *Element* that represents John in the simulation. When the *Process Requests Phase* takes place, a conflict arises and a decision has to be made on how the two push *Requests* are executed. The outcome will depend on the *Request Handler* that John has at that moment for executing push *Requests*.

Requests Handlers determine how *Requests* in general are executed. Every state change in the framework is performed by a *Request Handler*. This mechanism systematizes the mediation of conflicts by transmitting possible conflicting *Requests* to the same *Request Handler* which then acts as mediator.

3.3 Active and Passive Information Gathering

In the examples discussed so far the agents pro-actively gather information. John, Luke and Paul check each other's position when they need.

However, there are several circumstances in which it is preferable to be notified of a change in the simulation state. For example, in FearNot! there is an agent that manages the stories being created: the Story Facilitator. In order to decide when to advance to the next act, the Story Facilitator needs to be aware of certain events, such as whenever Luke goes away after bullying John. Instead of constantly polling for such information, it can be notified when that event happens.

The ION Framework offers both *active and passive gathering of information* which is the possibility of getting information by querying the simulation in a proactive way, or by subscribing a particular bit of information which will be delivered later. We use the observer pattern [6] to provide an event-driven paradigm. Hence, every time a change occurs in a particular *Element*, a corresponding *Event* is raised.

Similarly to what happens with *Requests*, *Events* are not processed immediately. Likewise, their handling is performed by *Event Handlers* at a specific phase of the update cycle denominated *Process Events Phase*. At that time all intervenients registered to get a particular *Event* are notified if that *Event* happened. Figure 2 depicts the simulation update cycle with both its phases.

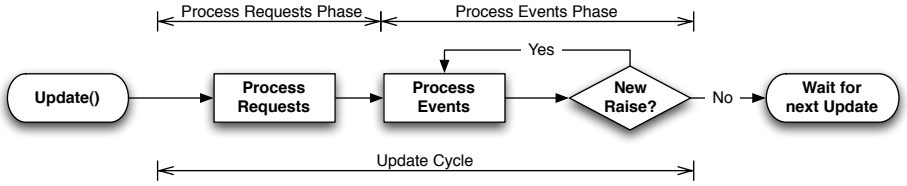

Fig. 2. Update Cycle

It should now be evident that there is a fundamental difference between *Requests* and *Events*. While *Requests* are the desired changes to the simulation state, *Events* are the information of which changes effectively took place.

3.4 Dynamic Configuration Changes

The ION Framework allows *dynamic configuration changes* and it is possible to completely change the simulation behaviour in runtime. We can add or remove *Elements* to the simulation, but also change how these *Elements* inherently behave by modifying their *Request Handlers*.

4 Case Studies

The ION Framework was used to model the virtual worlds in FearNot! [1] and Orient [2] which were tested with hundreds of users. It is also being used in the ongoing EU-funded project LIREC.

One of the most interesting aspects in FearNot! (introduced in the section 3) comes from the fact that characters are *Elements* of the ION Framework. These *Elements* provide an abstraction layer through *Requests* and *Events* which act as an interface between the mind components created in FAtiMA [3] and the embodiment provided by Ogre 3D [14]. Thus, each component can be replaced without having to change the rest of the environment.

A good example of how different applications can reuse components using the ION Framework is Orient. Orient is an application to enhance intercultural sensitivity towards people from other cultures. It has a very similar architecture and it used most of the components developed for FearNot!. A particular example was the interaction with users. In FearNot! the user was modelled as any other *Element* of the simulation and it had a single form of interaction (text input). Even though in Orient the user interacts with different devices (Wii, DanceMat, Mobile Phones) it had no impact on the other components of the simulation that still receive the same *Events* from the user *Element* regardless of the input device.

In LIREC we are exploring different forms of embodiment through the notion of competences. Competences represent specific abilities that the agent has (e.g. speech or facial expression). We are modeling these competences as ION *Elements* with a set of *Requests* and *Events* which are independent from a particular implementation. This way it is possible to change their implementation and the mind will still be able to use them as before. This possibility is particularly interesting in migration scenarios where the mind roams across different platforms which have different implementations for the same competence.

5 Conclusions

This paper introduced the ION Framework, a framework for simulating virtual environments. It uses a set of guidelines to regulate the interactions between the several elements, namely: 1) coherent access to information, 2) mediation of conflicts, 3) active and passive gathering of information, and 4) dynamic configuration changes.

Several examples depict how the ION Framework was used to address common problems in applications with multiple autonomous embodied agents.

In the future, the generic concepts we offer can be extended with patterns of use, providing an higher-level pool of components which may suit more particular applications or domains. We believe that it can establish a common ground that helps the community to share their research efforts.

6 Acknowledgments

This work was partially supported by European Community (EC) and is currently funded by the LIREC project FP7-215554. Guilherme Raimundo and Pedro Sequeira were supported by the Portuguese Foundation for Science and Technology (FCT), grant references SFRH/BD/25725/2005 and SFRH/BD/38681/2007.

The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC and FCT, and the EC and FCT are not responsible for any use that might be made of data appearing therein.

References

1. R. Aylett, S. Louchart, J. Dias, A. Paiva, M. Vala, S. Woods, and L. E. Hall. Unscripted narrative for affectively driven characters. *IEEE Computer Graphics and Applications*, 26(3):42–52, 2006.
2. R. Aylett, A. Paiva, N. Vannini, S. Enz, and E. Andre. But that was in another country: agents and intercultural empathy. In *AAMAS*, 2009.
3. J. Dias and A. Paiva. Feeling and reasoning: A computational model for emotional characters. In *EPIA*, pages 127–140, 2005.
4. Epic-Games. Unreal tournament website. (online) http://www.unrealtournament.com/, last seen in April 2009, 2008.
5. E. Friedman-Hill. *Jess in Action. Java Rule-based Systems*. Manning Publications, 2003.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
7. M. J. Huber and J. Leto. Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, USA, 1999. ACM Press.
8. JADE. Java agent development framework. (online) http://jade.tilab.com/, last seen in April 2009, 2009.
9. J. Klein. breve: a 3d environment for the simulation of decentralized systems and artificial life. In *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages 329–334, Cambridge, MA, USA, 2003. MIT Press.
10. S. Kopp. Articulated communicator engine (ace). (online) http://www.techfak.uni-bielefeld.de/ skopp/max.html, last seen in April 2009, 2000.
11. J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 1(33):1–64, 1987.
12. N. Minar, R. Burkhart, C. Langton, and Askenazi. The swarm simulation system, a toolkit for building multi-agent simulations. working paper 96-06-042, 1996.
13. SAIBA-Initiative. Situation agent intention behavior animation. (online) http://wiki.mindmakers.org/projects:saiba:main, last seen in April 2009, 2009.
14. S. Steve. Object-oriented graphics rendering engine (ogre). (online) http://www.ogre3d.org/, last seen in April 2009, 2009.
15. M. Thiébaux, S. Marsella, A. N. Marshall, and M. Kallmann. Smartbody: behavior realization for embodied conversational agents. In *AAMAS*, pages 151–158, 2008.
16. Valve. Source game engine. (online) http://source.valvesoftware.com/, last seen in April 2009, 2004.
17. U. Wilensky. Netlogo, 1999.