Monitoring Social Expectations in Second Life

(Extended Abstract)

Stephen Cranefield
Department of Information Science
University of Otago
Dunedin 9054, New Zealand
scranefield@infoscience.otago.ac.nz

Guannan Li
Department of Information Science
University of Otago
Dunedin 9054, New Zealand
telepeter520@gmail.com

ABSTRACT

Online virtual worlds such as Second Life provide a rich medium for unstructured human interaction in a shared simulated 3D environment. However, many human interactions take place in a structured social context where participants are subject to expectations governing their behaviour, and current virtual worlds do not provide any support for this type of interaction. There is therefore an opportunity to adapt the tools developed in the multi-agent systems community for structured social interactions between software agents (inspired by human society) and adapt these for use with the computer-mediated human communication provided by virtual worlds. This paper describes the integration of one such tool with Second Life. A model checker for monitoring social expectations defined in temporal logic has been integrated with Second Life, allowing users to be notified when their expectations of others have been fulfilled or violated.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; H.5.1 [**Multimedia Information Systems**]: Artificial, augmented, and virtual realities—*Second Life*; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Temporal Logic; D.2.4 [**Software/Program Verification**]: Model Checking

General Terms

Experimentation, Verification

Keywords

Virtual worlds, Second Life, social expectations

1. INTRODUCTION

Second Life[1] is a distributed client-server application providing a simulated three dimensional environment in which users can move around and interact with other users and simulated objects. Users are represented in the virtual world by animated avatars that they control via the Second Life Viewer client software. Human interaction in Second Life is essentially unconstrained—the users can do whatever they like, subject to the artificial physics of the simulated world and a few constraints that the worlds support, such as the ability of land owners to control who can access their land. However, many human interactions take place in a structured social context where there are constraints imposed on participants' behaviour by the social or organisational context, and virtual worlds currently provide no support to help users track the social state of their interactions.

In this research we have investigated the use of a tool for monitoring 'social expectations' in multi-agent systems [3] in conjunction with Second Life. The mechanism involves a script running in Second Life that is configured to detect and record particular events of interest for a given scenario, and to model these as a sequence of state descriptions that are sent to an external monitor along with a property to be monitored. The property is a statement asserting that a given social expectation, modelled as a conditional temporal logic rule, has become active, fulfilled or violated. The monitor sends notifications back to the script when the property is satisfied so that the user can be informed.

2. DETECTING EVENTS IN SECOND LIFE

The Second Life Viewer provides a graphical view of the user's avatar and other objects and avatars within the view. Avatars can be controlled to perform a range of basic animations such as standing, walking and flying, or predefined "gestures" that are combinations of animation, text chat and sounds. Communication with other avatars (and hence their users) is via text chat, private instant messages, or audio streaming. The user experience is therefore a rich multimedia one in which human perception and intelligence is needed to interpret the full stream of incoming data. However, the Linden Scripting Language[2] (LSL) can be used to attach scripts to objects (e.g. to animate doors), and there are a number of sensor functions available to detect objects and events in the environment.

In this work we have focused on the use of a script that runs within an avatar's "head-up display" and uses an LSL sensor function to detect other avatars and determine their current animation. The script is configured with scenario-specific information via a 'notecard' (a type of avatar inventory item commonly used to store configuration data for scripts). This contains the property to be monitored, a list assigning avatars to named groups (if required), and a filter list specifying which animations for particular avatars or group members should be recorded for further processing.

When the script starts up, it sends the property to be monitored to the monitor. It then sends a series of state descriptions to the monitor as sensor events occur and changes in animations are detected. State descriptions are sets of proposition symbols of the

[1] `http://secondlife.com/`

[2] `http://wiki.secondlife.com/wiki/LSL_Portal`

Cite as: Monitoring Social Expectations in Second Life, (Extended Abstract), Stephen Cranefield, Guannan Li, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1303–1304

form *avatar_animation* or *group_animation* (where the latter means that some member of the group is doing the animation).

This process can easily be extended to handle other types of Second Life events that can be described using propositional (rather than predicate) logic.

3. COMMUNICATION ARCHITECTURE

Figure 1 gives an overview of the communication architecture. To push property and state information to the monitor we use HTTP. However, instead of directly embedding the monitor in an HTTP server, to avoid local firewall restrictions we have chosen to use Twitter[3] as a message channel. An LSL XML-RPC channel key, the property to be monitored and a series of state descriptions are sent to a predefined Twitter account as direct messages using the HTTP API.

The monitor is wrapped by a Java client that polls Twitter to retrieve direct messages for the predetermined account. Each time a state description message is received, the monitor (which is implemented in C) is invoked using the Java Native Interface (JNI). The output is parsed and, if the property is determined to be true in any state, that information is sent directly back to the Second Life script via XML-RPC.

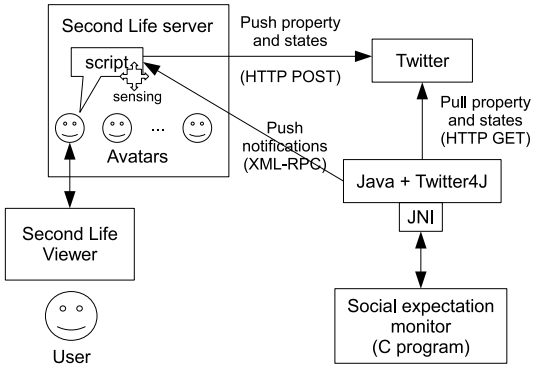

Figure 1: The communications architecture

4. THE MONITOR

The monitor used in this work is designed to track rules of *social expectation*. These are temporal logic rules that are triggered by conditions on the past and present, resulting in expectations on present and future events. The language does not include deontic concepts such as obligation and permission, but it allows the expression of social rules that impose complex temporal constraints on future behaviour, in contrast to the simple deadlines supported by most normative languages.

Expectations become active when their condition evaluates to true in the current state. These expectations are then considered to be fulfilled or violated in a state if they evaluate to true in that state without considering any future states that might be available in the model (for example, when checking a complete audit trail). If an active expectation is not fulfilled or violated in a given state, then it remains active in the following state, but in a "progressed" form. Formula progression involves partially evaluating the formula in terms of the current state and re-expressing it from the viewpoint of the next state [1]; a simple example is that an expectation $\bigcirc\phi$ (meaning that ϕ must be true in the state that follows) progresses to the expectation ϕ in the next state.

[3] `http://twitter.com/`

The monitor is an extension [3] of a model checker for hybrid temporal logics [4]. Model checking is the computational process of evaluating whether a formal model of a process satisfies a given property, usually expressed in temporal logic. For monitoring social expectations in an open system, we cannot assume that we can obtain the specifications or code of all participating agents to form our model. Instead our model is the sequence of system states recorded by a particular observer, in other words, we are addressing the problem of *model checking a path* [5]. The task of the model checker is therefore not to check that the overall system *necessarily* satisfies a given property, but just that the observed behaviour of the system has, to date, satisfied it. Formal details of the logic and model checker have been reported previously [3].

5. EXAMPLE SCENARIO

Consider a scenario in which there are two groups (or roles) specified in the script's group configuration list: `leader` (a singleton group) and `follower`. We want to monitor for violations of the rule that once the leader is standing, then from the next state a follower must not be sitting until the leader is sitting again. This is expressed using the following property (using a slight simplification of the monitor's input syntax):

```
ExistsViol(
  leader_standing,
  <next>U(!follower_sitting, leader_sitting))
```

The filter list can be configured so that only the propositions occurring in this rule are regarded as relevant for describing the state.

Suppose the scenario begins with the leader sitting and then standing, followed by the follower sitting (after standing initially), and finally the leader sitting again. This causes four states to be generated and sent to the monitor, which checks the property at each state in this sequence and detects a violation in the third state. This information is sent to the script so it can inform the user.

6. CONCLUSION

This paper has reported on a prototype application of a model checking tool for social expectation monitoring applied to monitoring social interactions in Second Life. The techniques used for monitoring events in Second Life and allowing communication between a Second Life script and the monitor have been described, and these have been successfully tested on some simple scenarios. Further details, discussion and a comparison with related work can be found in the full version of this paper [2].

7. REFERENCES

[1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[2] S. Cranefield and G. Li. Monitoring of social expectations in Second Life. Discussion Paper 2009/02, Department of Information Science, University of Otago, http://eprints.otago.ac.nz/795/, 2009.

[3] S. Cranefield and M. Winikoff. Verifying social expectations by model checking truncated paths. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IV*, *Lecture Notes in Computer Science*, 5428:204–219. Springer, 2009.

[4] L. Dragone. Hybrid logics model checker. http://luigidragone.com/hlmc/, 2005.

[5] N. Markey and P. Schnoebelen. Model checking a path. In *CONCUR 2003 – Concurrency Theory*, *Lecture Notes in Computer Science*, 2761:251–265. Springer, 2003.