Interfacing a Cognitive Agent Platform with Second Life

Surangika Ranathunga, Stephen Cranefield, and Martin Purvis

Department of Information Science, University of Otago,
PO Box 56, Dunedin 9054, New Zealand
{surangika,scranefield,mpurvis}@infoscience.otago.ac.nz

Abstract. Second Life is a multi-purpose online virtual world that provides a rich platform for remote human interaction. It is increasingly being used as a simulation platform to model complex human interactions in diverse areas, as well as to simulate multi-agent systems. It would therefore be beneficial to provide techniques allowing high-level agent development tools, especially cognitive agent platforms such as belief-desire-intention (BDI) programming frameworks, to be interfaced to Second Life. This is not a trivial task as it involves mapping potentially unreliable sensor readings from complex Second Life simulations to a domain-specific abstract logical model of observed properties and/or events. This paper investigates this problem in the context of agent interactions in a multi-agent system simulated in Second Life. We present a framework which facilitates the connection of any multi-agent platform with Second Life, and demonstrate it in conjunction with an extension of the Jason BDI interpreter.

1 Introduction

Second Life [1] is a multi-purpose online virtual world that is increasingly being used as a simulation platform to model complex human interactions in diverse areas such as eduction, business, medical and entertainment. This is mainly because of the rich platform it provides for remote human interactions, including the possibility of enabling software-controlled agents to interact with human-controlled agents. Second Life is more sophisticated than conventional 2D simulation tools, and is more convenient than cumbersome robots, thus it has started to gain attention as a simulation platform for testing multi-agent systems and other AI concepts. It would therefore be beneficial to provide techniques allowing high-level agent development tools, especially cognitive agent platforms such as belief-desire-intention (BDI) programming frameworks, to be interfaced with Second Life.

When interfacing agent platforms with Second Life, there are two important aspects to be addressed: how the sensor readings from the Second Life environment are mapped to a domain-specific abstract logical model of observed properties and/or events and how the agent actions are performed on the Second Life virtual environment. The first aspect can be quite complex when considering the high volumes of potentially unreliable sensor readings an agent receives. As for the latter, it is important to identify ways of correctly interfacing the agents with their representation module inside Second Life (the Second Life avatar), because Second Life may have synchronization issues with respect to carrying out the actions specified by the agent model.

With the use of the *LIBOMV* client library [2], we have developed a framework that facilitates the connection of any multi-agent framework with Second Life and addresses the above challenges. The main focus of this paper is to highlight the importance and difficulty of creating an abstract logical model of the sensory inputs of an agent deployed in Second Life, and to present the solution we developed in our connection framework to address this problem.

Creating a high-level abstract logical model of agent sensory data involves two main steps: extracting sensory readings from Second Life accurately, and formulating a high-level domain-specific abstract logical model to be passed to an agent's cognitive module. The latter has not gained much attention in the related research with respect to deploying intelligent agents inside Second Life.

In our framework, an agent deployed in Second Life can sense the Second Life environment around it with the use of its LIBOMV client, and the framework records these sensor readings. There are some difficulties in obtaining accurate sensor readings from Second Life simulations. Therefore we have introduced a novel technique in our framework, which extracts sensor readings from Second Life more accurately than the commonly used data extraction methods.

The extracted sensory data result in a high volume of low-level information (avatar and object position information and avatar animation information), making it difficult to use this data in an agent's reasoning process. In order to convert this low-level information into a form that can be used by the multi-agent system, we employ a complex event processing mechanism and identify the high-level domain-specific complex events embedded in the retrieved low-level data. The output of the framework is a snapshot of the Second Life environment that contains all the low-level and high-level events and other contextual information that took place in a given instant of time, encoded as propositions. This provides an agent a complete view of the environment around it, thus eliminating the possibility of having to base its reasoning on a partial set of data.

We also note that our framework facilitates the co-existence of agents belonging to multiple agent platforms in the same Second Life simulation. In this paper, we demonstrate this framework in conjunction with an extension of the Jason BDI interpreter that allows agents to specify their expectations of future outcomes in the system and to respond to fulfilments and violations of these expectations [3]. An agent's expectations we consider here are constraints on the future that are based on published norms, agreed contracts, commitments created through interaction with other agents, or personally inferred regularities of agent behaviour. An agent may base its practical reasoning on the assumption that one or more of its expectations will hold, while ensuring that it will receive notification events when these rules are fulfilled and/or violated.

With the extended functionality of the Jason platform, we demonstrate how a Jason agent deployed in Second Life using our framework can take part in complex simulations and respond to the received percepts from Second Life and the identified fulfilments and violations of its expectations. The fulfilments and violations of an agent's expectations are detected by an expectation monitor [4] that is integrated with the framework through an interface, and the agent's expectations are defined as temporal logic formulae to be monitored by the expectation monitor. The framework forwards the processed sensory readings from Second Life to both the Jason environment and the expectation monitor. Therefore, in parallel to a Jason agent being able to respond to the observed changes in the environment, the expectation monitor matches these changes with the monitored formulae and identifies the fulfilment or violation of these defined expectations. The notifications of the identified fulfilments or violations are also passed to the Jason agent.

The rest of the paper is organized as follows. Section 2 describes the potential of Second Life as a simulation environment and the related implementation problems. Section 3 describes the developed framework and in Section 4, we demonstrate this developed system by means of an example. Section 5 discusses some related work. Section 6 concludes the paper.

2 Second Life as a Simulation Environment

Second Life provides a sophisticated and well developed virtual environment for creating simulations for different domains and to test AI theories, including agent-based modelling. With the average monthly repeated user logins at around 800000[1], and with the virtual presence of many organizations, Second Life contains many interaction possibilities, which inherently lead to the provision of new scenarios to be used in simulations. Second Life is not restricted to a specific gaming or training scenario. Developers can create a multitude of scenarios as they wish, using the basic building blocks that are provided. For example, in Second Life, these scenarios could be in the areas of education, business, entertainment, health or games. The significance of using Second Life scenarios lies in the fact that they can be carried out between software-controlled agents, and also between software-controlled agents and human-controlled agents.

Second Life has been identified as a good simulation platform for testing AI theories [5] and specifically multi-agent systems [6]. A detailed analysis on the benefits of using Second Life over traditional 2D simulations and physical robots has also been done [5], with the main advantage reported being the ability to create sophisticated test beds in comparison to 2D simulations, and more cost effective test beds when compared to physical robots.

Despite this, still we do not see Second Life being used for complex simulations of AI theories or multi-agent systems modelling. The lack of use of Second Life as a simulation environment for AI research can be, to a certain extent,

[1] http://blogs.secondlife.com/community/features/blog/2011/01/26/the-second-life-economy-in-q4-2010

attributed to the previous lack of a convenient programming interface. Traditional programming in Second Life is done using in-world scripts created using the proprietary *Linden Scripting Language (LSL)*. These scripts are associated with objects, and in order to use them to control an agent inside Second Life, the objects should be attached to the agent. This approach has many limitations when used for AI simulations, for reasons such as the limited control over the agent wearing the scripted object. We discuss this in more detail in Section 2.1.

With the development of the third party library LibOpenMetaverse (LIBOMV), Second Life can now be accessed through a more sophisticated programming interface. LIBOMV is a ".Net based client/server library used for accessing and creating 3D virtual worlds" [2], and is compatible with the Second Life communication protocol. Using the LIBOMV client-side API, "bots" can be defined to control avatars in Second Life. With appropriate programming techniques, the LIBOMV library can be used to create avatars that have behavioural abilities similar to those controlled by humans. This includes moving abilities such as walking, running or flying, performing animations such as crying, or laughing, communication abilities using instant messaging or public chat channels, and the ability to sense the environment around it.

2.1 Challenges in Monitoring Agent Interactions in Second Life

For Second Life simulations that contain a lot of agents and objects moving at speed, there is a challenge in retrieving accurate position information at a high frequency to make sure that important events are not missed out.

Although an in-world sensor created using an LSL script can retrieve accurate position information of avatars and objects, it has limitations when extracting position and animation information of fast moving objects and avatars. A sensor can detect only 16 avatars and/or objects in one sensor function call, and the maximum sensor range is 96 metres. One approach to overcoming this problem is to employ multiple sensors; however multiple scripts operating for long durations at high frequency introduce "lag" to the Second Life servers, i.e. they slow the rate of simulation. For the same reason and because of the imposed memory limitations on scripts, an LSL script cannot undertake complex data processing, and since there is no provision to store the recorded data in-world at runtime, recorded data must be communicated outside the Second Life servers using HTTP requests which are throttled to a maximum of only 25 requests per 20 seconds. Moreover, there is a possibility that avatar animations with a shorter duration (e.g. crying or blowing a kiss) may go undetected, because a sensor can record only animations that are played during the sensor operation.

With a LIBOMV client deployed in Second Life, all the aforementioned limitations can be avoided. Avatar and object movements and avatar animations inside a Second Life environment generate corresponding update events in the Second Life server, and the server passes this information to the LIBOMV client using the Second Life communication protocol. The processing of this information is done outside the Second Life servers, thus causing no server lag.

However, this approach does have its own limitations which affect the accuracy of recorded information. As with other viewer clients, the Second Life server sends information to the LIBOMV client only if there is any change in the environment perceived by the LIBOMV client. This means that the client has to "assume" its perceived environment. For objects and avatars that are moving, the client has to keep on extrapolating their position values based on the previously received velocity and position values until it receives an update from the server. Extrapolated position values may not be completely in tally with the server-sent values and this situation is evident when extrapolating position values for objects and avatars that move fast. Moreover, it was noted that there is an irregularity in the recorded position data for small objects that may easily go out of the viewing range of the LIBOMV client, which directly affects the recording of accurate position information for small objects.

In order to overcome these challenges, we introduce a combined approach (described in Section 3) based on attaching an object containing an LSL script to a LIBOMV client deployed in Second Life. These communicate with each other and produce near-accurate position information about avatars and objects that move at speed.

This data extraction mechanism can only generate low-level position and animation information, which should be further processed to identify the high-level domain-specific information embedded in the low-level data. In doing this, it is important that the data collected using the LIBOMV client and the LSL script are formed into one coherent snapshot which resembles the state of the Second Life environment. When deducing the high-level domain-specific information, it is important that these coherent snapshots are used, in order to make use of all the events and other related information that took place in a given instant of time. Otherwise an agent's decision may be based on partial information.

3 System Design

Figure 1 shows how different components of the system are interfaced with each other. The LIBOMV client creates and controls an avatar inside the Second Life server. It continuously senses the environment around it, and carries out movement and communication acts as instructed and passes back the result notifications to the connected agent module whenever necessary (e.g. the result notification of the login attempt). We have used the Jason agent development platform [7], which is based on the BDI agent model, to demonstrate the integration of multi-agent platforms with Second Life using our framework. Here a Jason agent acts as the coordinator component of this system. It instantiates the LIBOMV client to create the corresponding Second Life avatar, and commands the LIBOMV client to carry out actions inside Second Life on behalf of it.

The Extended Jason Platform The Jason platform we have integrated with the framework is an extended version [3] of Jason. The Jason agent platform contains an environment interface that facilitates the easy integration of Jason

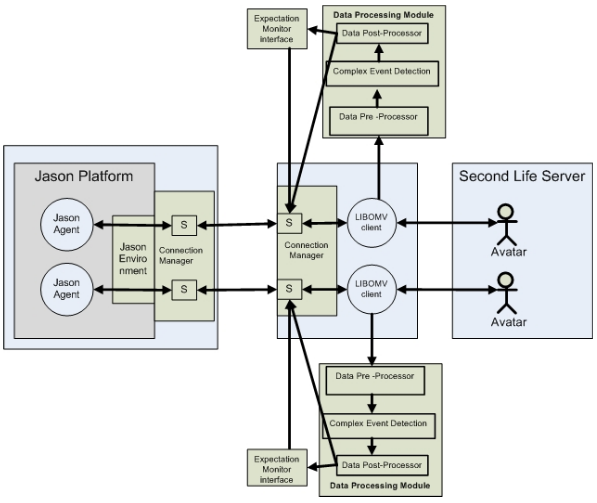

Fig. 1. Overall System Design

agents with other simulations. With this interface, it is possible to execute the agent actions in an external simulated environment (for example, passing the commands to a robot's actuators) and it is also possible to retrieve the sensory readings of the simulated environment to be presented as percepts for agents.

The extended version of the Jason architecture used in this work implements a tight integration of expectation monitoring with the Jason BDI agent model, where domain-specific individual agents can directly react to the identified fulfilments and violations of their expectations. The Jason interpreter is extended with built-in actions to initiate and terminate monitoring of expectations, and with these built in actions, any expectation monitoring tool can be "plugged in" to the Jason environment.

Interface Between the LIBOMV Client and the Jason Agent The interface between the LIBOMV client and the Jason agent is facilitated using a simple protocol we have developed (which we intend to develop further), and they communicate through sockets (denoted by 'S' in Figure 1). This decoupling makes it possible to connect any agent platform with the LIBOMV clients easily, and it could well be the case that different LIBOMV clients are connected with agents in different agent platforms. The protocol currently defines how an agent should pass commands to the LIBOMV client such as requesting the LIBOMV client to log into the Second Life server, uttering something in the public chat channels, sending instant messages to other avatars, moving to a given location in a given way (e.g. walking, running or flying) and executing an animation. It also defines how an agent platform can interpret a message sent by the LIBOMV client. These messages are formulated based on the environment information recorded by the LIBOMV client module. The Jason environment class makes use of this protocol and converts the agent actions into the corresponding protocol constructs and passes them to the LIBOMV client. Similarly, it interprets the messages sent by LIBOMV clients to generate percepts for the Jason agents.

The module that contains LIBOMV clients is capable of handling multiple concurrent LIBOMV clients and socket connections. Therefore if the corresponding multi-agent system is able to create concurrently operating agents, this can easily create a multi-agent simulation inside Second Life. Consequently, the module that contains the Jason platform is designed in such a way that it is capable of handling multiple concurrent instances of socket connections connected to the Jason agents. As shown in Figure 1, a Jason agent connects to its interface socket through the Jason Environment class, and the Jason Connection manager interface. The Jason connection manager and the LIBOMV connection manager together ensure that all these individual Jason agents are connected to the correct LIBOMV client, through the interface sockets.

Interface Between the LIBOMV Client and the Second Life Server As an attempt to overcome the limitations of data extraction using LSL and LIBOMV, we have implemented a combined approach to extract data from Second Life. In this new approach, a scripted object is attached to the LIBOMV client. Detection of the avatars and objects to be monitored is done at the LIBOMV client side. Identification information for these is then sent to the script. As the script already knows what is to be tracked, a more efficient, light-weight function can be used to record position and velocity information instead of the normal LSL sensor function. Recorded position and velocity data are sent back to the LIBOMV client, while avatar animation updates are directly captured by the LIBOMV client to make sure animations with short durations are not missed. Any messages received as instant messages or in the public chat channels are also directly captured by the LIBOMV client. With this combined approach, the LSL script guarantees the retrieval of accurate position information, while the LIBOMV client takes the burden of complex data processing off the Second Life servers, thus providing an accurate and efficient data retrieval mechanism.

3.1 Data Processing Module

The data processing module consists of three main components; the data preprocessor, the complex event detection module and the data post-processor. The responsibility of the data processing module is to map the received sensor readings from complex Second Life environments to a domain-specific abstract logical model. In essence, it creates snapshots of the system that include position and animation information of the avatars and objects in the given Second Life environment in a given instant of time, along with the identified high-level domain-specific information and other contextual information, which are encoded as propositions.

Data Pre-Processor: First, the low-level data received from Second Life are used to deduce basic high-level information about the avatars and objects, e.g. whether an avatar is moving, and if so, in which direction and the movement type (e.g. walking, running or flying), and whether an avatar is in close proximity to another avatar or an object of interest. Other contextual information such as the location of the avatar or the role it is playing can also be attached to this retrieved information as needed.

As mentioned above, the LIBOMV client receives position information of objects and avatars from the script, and the updates corresponding to avatar animations and communication messages are directly captured by the LIBOMV client. This means that a received position information update does not contain the information about the current animation of the corresponding avatar, and the received animation and message updates do not contain the information about the current position of the avatar. Moreover, these animation and communication updates do not contain the position information of other avatars and objects in the environment, or their animation information. However, for every update received by the LIBOMV client (whether it be the position updates from the script, or an animation or a communication update), it is important to combine the information received from all these different sources, in order to create a complete snapshot of the Second Life environment. Therefore the data pre-processor caches the latest received animation, position and velocity information for each avatar and object of interest.

When a new set of position information is received from the script, the cached animation information for that avatar is associated with the newly received avatar movement information. The LIBOMV client receives an update corresponding to every avatar animation change (e.g. if an avatar is currently standing, and suddenly starts running, the LIBOMV client receives an animation update 'run'). Therefore it is safe to assume that an avatar keeps on performing the animation already recorded in the cache. Similarly when an animation update is received for an avatar, it is associated with the extrapolated position values of that avatar, based on the cached position and velocity information. Since the LIBOMV client receives position information from the script every 500 milliseconds, the extrapolation error can be assumed to be very low. As for the received communication messages, the cached information corresponding to the avatar is used to generate its position and animation information at the time of the communication act. Finally, in each of these cases, animation, position and velocity information for all the other avatars and objects of interest is generated for the time instant represented by the received update. Thus, for every set of position information sent by the script and every animation and communication message update sent by the Second Life server, the data pre-processor generates a complete snapshot of the environment that contains the avatar and object positions and avatar animations. These snapshots can be easily distinguished from each other with the use of the associated timestamp.

These processed data are then sent to another sub-component of the data pre-processor which prepares data to be sent to the complex event detection module. We specifically extracted this sub-component from the main data pre-processing logic in order to make it possible to easily customize the data preparation logic according to the selected complex event detection module. For example, for the complex event detection module we have employed currently, this sub-component decomposes the generated snapshot into the constituent data structures corresponding to individual avatars and objects, and sends the information related to objects to the complex event detection module before those corresponding to avatars.

Complex Event Detection Module: An event stream processing engine called Esper [8] is used to identify the complex high-level domain-specific events embedded in the data streams generated by the data pre-processor. The Esper engine allows applications to store queries and send the low-level data streams through them in order to identify the high-level aggregated information. Esper keeps the data received in these data streams for time periods specified in these queries, thus acting as an in-memory database. Esper also has the ability to process multiple parallel data streams.

Esper provides two principal methods to process events: event patterns and event stream queries. We make use of both these methods when identifying the high-level domain-specific events. The received data streams are sent through the event stream queries first, to filter out the needed data. Then these filtered data are sent through a set of defined patterns which correspond to the high-level events that should be identified. Event identification using patterns is done in several layers to facilitate the detection of events with a duration. The output of each layer is subsequently passed on to the layer that follows, thus building up hierarchical patterns.

The output of the complex event detection module is sent to the data post-processor.

Data Post-Processor: The data post-processor is required to convert the recognized low-level and high-level information into an abstract model to be passed to the connected multi-agent system.

The detected low-level data, as well as high level events and other context information are converted to propositions and are grouped into states to be sent to the multi-agent system. Essentially, a state should represent a snapshot of the Second Life environment at a given instant of time. Therefore the times at which the basic events (e.g. receipt of avatar animation, or receipt of the position information from the script) were received by the system were selected as the instants modelled in the output state sequence. This creates separate states consisting all the low-level events that took place at the same basic event, high-level events as well as the related contextual information.

Expectation Monitor Interface: The expectation monitor interface shown in Figure 1 is an optional sub-component that processes the output of the data post-processor a step further by adding a reference to the dependent state for those events that depend on previous other high-level events. It sends these data to an expectation monitor attached to it, and in this work we use an expectation monitor that was developed in previous research [4]. The responsibility of the expectation monitor is to identify the fulfilments and violation of agent expectations that are defined using the extended version of the Jason platform as explained in Section 1.

When an expectation monitor is initially started, it receives a rule (a condition and an expectation) and a property (fulfilment or violation) through the expectation monitor interface to start monitoring. The rule's condition and resulting expectation are provided as separate arguments using a specific form of temporal logic, with the expectation expressing a constraint on the future sequence of states [4]. When the monitor starts receiving the output of the data post-processor as a sequence of states, it matches these against the rule's condition to determine if the expectation has become active. It also evaluates any active expectations (created by a condition evaluating to true), progressively simplifies the monitored expectation and finally deduces fulfilment or violation of the expectation[2]. The fulfilments and violations of agent expectations add a new level of abstraction above the state descriptions generated by the data post-processor, where the expectations are introduced by the agent dynamically and the fulfilments and violations are detected based on the already identified information in the snapshots. Therefore, in addition to the continuous stream of domain-specific high-level events and state information that our framework supplies to the agent from Second Life, an agent developed using this extended version of the Jason platform can dynamically subscribe to fulfilment and violation events for specific rules of expectation that are appropriate to its personal or social context.

4 Example - A Jason Agent Engaged in the Football Team Play Scenario "Give and Go"

In this section we demonstrate how a Jason agent can engage in a SecondFootball [9] virtual football training scenario with a human controlled player[3], and how it can reason based on received percepts and the detected fulfilments and violations of its expectations.

SecondFootball is an interesting simulation in Second Life which enables playing virtual football. It is a multi-avatar, fast-moving scenario which promises to be a hard test case to test our framework when compared with most of the publicly accessible environments in Second Life. This system provides scripted stadium and ball objects that can be deployed inside Second Life, as well as a "head-up display" object that an avatar can wear to allow the user to initiate kick and tackle actions.

In this example, we implement a simplified version of the football team play scenario "give and go". Figure 2 shows a screen shot of this training scenario.

[2] The system employs multiple expectation monitor instances in parallel in order to monitor multiple concurrently active expectations an agent may have. This is due to a limitation in the expectation monitor we have employed that it cannot monitor for concurrently active individual expectations.

[3] One of our agents is currently controlled by a human as our Jason agents are still not capable of handling complex reasoning involved with playing football.

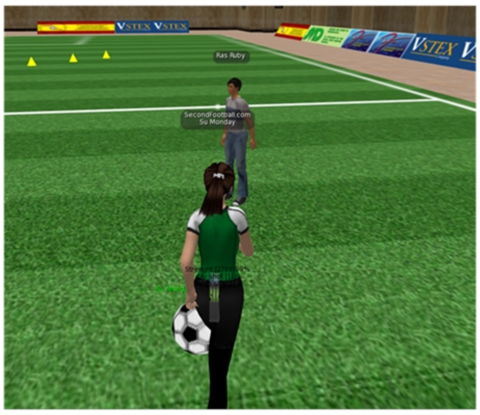

Fig. 2. Su_Monday getting ready to pass the ball to Ras_Ruby to start the training scenario

Here, the Jason agent Ras_Ruby is engaged in the team play scenario with the player Su_Monday, who is controlled by a human. When Ras_Ruby receives the ball, she adopts the expectation that Su_Monday will run until she reaches the PenaltyB area, so that she can pass the ball back to Su_Monday, to attempt to score a goal.

In order to implement this team-play scenario, the high-level complex events of the SecondFootball domain we wanted to detect were whether the ball was in the possession of a particular player, whether the ball is being advanced towards a goal, and successful passing of the ball among players by means of up-kicks and down-kicks. Though not used in the example, the framework is also capable of detecting goal scoring by up-kicks and down-kicks, dribbling the ball over the goal line, and successful or unsuccessful tackles. The developed framework had to be customised to achieve these requirements, and in the future we intend to introduce options(e.g. configuration files and run-time scripts) that can be utilized to customize the framework for a given Second Life simulation more easily.

When the system starts, the Jason agent corresponding to Ras_Ruby is initialized. When the Jason agent starts executing, it first tries to log itself in Second Life. The following Jason plan initiates the login process.

```
// The ‘+!’ prefix resembles a new goal addition
+!start  <-
   connect_to_SL("xxxx", "Manchester United, 88, 118, 2500");
   !check_connected.
```

The parameters specify the login password and the login location, respectively.

After sending this login request to the LIBOMV client, the agent has to wait till it gets the confirmation of the successful login from the LIBOMV client, as shown in the following plan:

```
+!check_connected: not connected
   <-
   .wait(2000);
   // ‘!!’ means tail-recursion optimised posting of a goal
   !!check_connected.
```

When it finally receives the successful login notification, the agent instructs the LIBOMV client to run the avatar to the area MidfieldB2 using the plan shown below.

```
+!check_connected: connected
    <-
    action("run","MidfieldB2").
```

Once in the area MidfieldB2, the agent Ras_Ruby waits for Su_Monday to kick and pass the ball to it. Once it successfully receives the ball the agent gets the “successful_kick(su_monday, ras_ruby)” percept (which is generated by the framework and states that Su_Monday successfully passed the ball to Ras_Ruby through a kick), and this generates a new belief addition event (‘+successful_kick’) which triggers the corresponding plan given below.

In this plan, we have used the internal action start_monitoring defined in the extended version of the Jason platform [3], and initiate monitoring for the fulfilment and violation of the expectation. Here, in the first parameter we define the type of expectation; whether it is a fulfilment or a violation. The second parameter assigns a name for the expectation. The third parameter is the name of the expectation monitor used. The fourth parameter is the triggering condition for the expectation, and in this example, it is a keyword with a special meaning (`#once`). For this scenario the initiating agent wants the rule to fire precisely once, as soon as possible, and this can be achieved in our current expectation monitor by using a ‘nominal’ (a proposition that is true in exactly one state) for the current state as the rule’s condition. However, the BDI execution cycle only executes a single step of a plan at each iteration, and any knowledge of the current state of the world retrieved by the plan may be out of date by the time the monitor is invoked. The `#once` keyword instructs the monitor to insert a nominal for the current state of the world just before the rule begins to be monitored. Here, the actual expectation formula is given by the fifth parameter, and the sixth parameter is a list of optional context information, which we do not utilize in this example.

The fulfilment of this expectation occurs when Su_Monday advances towards GoalB (`advanceToGoalB(su_monday)`), until `('U')` she reaches PenaltyB, denoted by `'penaltyB(su_monday)'`. Similarly, the violation of this expectation occurs if Su_Monday stopped somewhere before reaching penaltyB, or she moves in the opposite direction before reaching PenaltyB area[4].

[4] The conditions and expectations are defined in temporal logic and we do not wish to elaborate on them in the scope of this paper. These are written as nested Python tuples, as this is the input format for the expectation monitor written in Python.

```
//The '+' prefix resembles an event relating to belief addition
+successful_kick(su_monday,ras_ruby)  <-
   //internal actions
   .start_monitoring("fulf",
      "move_to_target",
      "expectation_monitor",
      "#once",
      "('U',
         'advanceToGoalB(su_monday)',
         'penaltyB(su_monday)')",
      []);

   .start_monitoring("viol",
       "move_to_target",
       "expectation_monitor",
       "#once",
       "('U',
         'advanceToGoalB(su_monday)',
         'penaltyB(su_monday)')",
      []).
```

If Su_Monday fulfilled Ras_Ruby's expectation, the expectation monitor detects this and reports back to the Jason agent. The following plan handles this detected fulfilment and instructs the avatar to carry out the kick action[5].

```
+fulf("move_to_target", X)  <-
   //Calculate kick direction and force, turn, then ...
   action("animation", "kick").
```

On the other hand, if Su_Monday violated the expectation, the expectation monitor reports the violation to the Jason agent, and the agent uses the first plan below to decide the agent's reaction to the detected violation, which creates a goal to choose a new tactic for execution. The second plan (responding to this new choose_and_enact_new_tactic) is then triggered, and the agent adopts the tactic of attempting to score a goal on its own by running towards the PenaltyB area with the ball[6] .

```
+viol("move_to_target",X)  <-
   !choose_and_enact_new_tactic.

+!choose_and_enact_new_tactic : .my_name(Me)  <-
   action("run", "penaltyB").
```

[5] Due to technical problems the Second Life avatar cannot currently perform the actual 'kick' animation

[6] when an avatar is in possession of the ball and the avatar starts moving, the ball moves in front of the avatar

5 Related Work

Research involved with programming with Second Life has focused either on extracting sensory readings from Second Life, or controlling avatar movement and conversational behaviours to create Intelligent Virtual Agents (IVA). Not much research has attempted to model reactive agents that generate behavioural responses to their observations on the Second Life environment, or addressed the issue of mapping low-level sensory data to high-level domain-specific information.

Most of the research that worked on extracting sensory readings from Second Life has utilized this retrieved information for statistical purposes. It can be seen that both LSL scripts and LIBOMV clients have been used for sensory data extraction from Second Life servers, but the latter had been more effective in collecting large amounts of data. LIBOMV clients have been successfully used to create crawler applications that collected large amounts of data about avatars and user-created content, to statistically analyze the number of avatars and objects present in various different Second Life regions over periods of time [10, 11]. There has also been an attempt to exploit the power of both these approaches in designing a multi-level data gathering tool which collected more than 200 million records over a period of time [12]. There have also been several attempts to collect data from Second Life using LSL scripts to examine social norms related to gender, interpersonal distance, dyantic interaction proximities and spatial responses in a virtual environment [13, 14], however both these studies had to base their analysis on a very low number of data samples due to the data collection mechanism they employed. A similar study was carried out using LIBOMV clients where the authors tried to capture spatio-temporal dynamics of user mobility [15].

Cranefield and Li presented an LSL script-based framework that sensed the Second Life environment and tried to identify the fulfilments and violations of rules defined in structured virtual communities [16]. However, this research had been conducted in a narrow scope which dealt only with animations of human-controlled avatars.

Burden provided a theoretical proposal for creating IVAs inside Second Life with the sophisticated abilities of concurrent perception, rational reasoning and deliberation, emotion, and action, and also pointed out the complexities of a practical implementation [17]. A theoretical framework has also been proposed which integrates different modules that handle these different capabilities [6], but the practical implementation of both of these is still limited to simple sensory, movement and conversational abilities.

There have been several research attempts on creating IVAs inside Second Life using LIBOMV clients, but their main focus had been on improving the conversational and animation abilities of the agents [18, 19].

Research has been carried out by Bogdanovych and colleagues who developed a number of useful libraries for connecting agents to Second Life (including their own BDI interpreter for controlling agents inside Second Life), in specially designed environments that were instrumented to connect to "electronic institution" middleware [20]. In contrast, our research focused on developing a framework that supports connecting multi-agent systems with existing Second Life environments. Moreover, they have not much focused on how to create coherent snapshots that provide a complete view of a given Second Life environment at a given instant of time to be presented to the multi-agent system, or how the extracted low-level data can be used to identify much complex high-level information, which was the main focus of our work.

6 Conclusion

In this paper we presented a framework that can be used to deploy multiple concurrent agents in complex Second Life simulations, and mainly focused on how the potentially unreliable data received by an agent deployed in a Second Life simulation should be processed to create a domain-specific high-level abstract model to be used by the agent's cognitive modules. This problem has not gained much attention from the past research on Second Life. We hope the implementation details we provided would be a valuable road map for future researchers hoping to use Second life for multi-agent simulations in various different paradigms, apart from the developed framework being a potential starting point for further research in integrating multi-agent systems with Second Life.

We note that any multi-agent platform can be connected with Second Life using our framework, and demonstrated this with an extended version of the Jason BDI interpreter. With the use of an example, we demonstrated how a Jason agent can execute actions inside Second Life and how it can respond to the observed changes in the environment. We also integrated an expectation monitor with our framework and demonstrated how Jason agents can use the sensory data to identify higher level events associated with fulfiled and violated personal expectations, based on the complex interactions that they take part in.

Although the current framework is customized for the SecondFootball simulation, in the future we plan to enhance this framework to be more generalized, and experiment with it in various simulations such as medical training scenarios. Moreover, we intend to enhance the capabilities of Jason agents, so that they will be able to actively participate in more complex scenarios.

References

1. Linden Lab. Second Life Home Page. `http://secondlife.com`
2. OpenMetaverse Organization. libopenmetaverse developer wiki. `http://lib.openmetaverse.org/wiki/Main_Page`
3. Ranathunga, S., Cranefield, S., Purvis, M.: Integrating Expectation Handling into Jason. Discussion Paper 2011/03, Department of Information Science, University of Otago (2011). `http://eprints.otago.ac.nz/1093/`
4. Cranefield, S., Winikoff, M.: Verifying social expectations by model checking truncated paths. Journal of Logic and Computation (2010). Advance access, doi: 10.1093/logcom/exq055

5. Veksler, V.D.: Second Life as a Simulation Environment: Rich, high-fidelity world, minus the hassles. In: Proceedings of the 9th International Conference of Cognitive Modeling (2009)
6. Weitnauer, E., Thomas, N., Rabe, F., Kopp, S.: Intelligent agents living in social virtual environments bringing Max into Second Life. In: H. Prendinger, J. Lester, M. Ishizuka (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 5208, pp. 552–553. Springer Berlin / Heidelberg (2008)
7. Bordini, R.H., Hubner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons Ltd, England (2007)
8. EsperTech. Esper Tutorial. `http://esper.codehaus.org/tutorials/tutorial/tutorial.html`
9. Vstex Company. SecondFootball Home Page. `http://www.secondfootball.com`
10. Varvello, M., Picconi, F., Diot, C., Biersack, E.: Is there life in Second Life? In: Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT '08, pp. 1:1–1:12. ACM, New York, NY, USA (2008)
11. Eno, J., Gauch, S., Thompson, C.: Intelligent crawling in virtual worlds. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03, WI-IAT '09, pp. 555–558. IEEE Computer Society, Washington, DC, USA (2009)
12. Kappe, F., Zaka, B., Steurer, M.: Automatically detecting points of interest and social networks from tracking positions of avatars in a virtual world. In: Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining, pp. 89–94. IEEE Computer Society, Washington, DC, USA (2009)
13. Friedman, D., Steed, A., Slater, M.: Spatial social behavior in Second Life. In: C. Pelachaud, J.C. Martin, E. Andr, G. Chollet, K. Karpouzis, D. Pel (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 4722, pp. 252–263. Springer Berlin / Heidelberg (2007)
14. Yee, N., Bailenson, J.N., D, P., Urbanek, M., Chang, F., Merget, D.: The unbearable likeness of being digital; the persistence of nonverbal social norms in online virtual environments. Cyberpsychology and Behavior **10**, 115–121 (2007)
15. La, C.A., Michiardi, P.: Characterizing user mobility in Second Life. In: Proceedings of the first workshop on Online social networks, WOSP '08, pp. 79–84. ACM, New York, NY, USA (2008)
16. Cranefield, S., Li, G.: Monitoring social expectations in Second Life. In: J. Padget, A. Artikis, W. Vasconcelos, K. Stathis, V. Silva, E. Matson, A. Polleres (eds.) Coordination, Organizations, Institutions and Norms in Agent Systems V, *Lecture Notes in Artificial Intelligence*, vol. 6069, pp. 133–146. Springer (2010)
17. Burden, D.J.H.: Deploying embodied AI into virtual worlds. Knowledge-Based Systems **22**, 540–544 (2009)
18. Ullrich, S., Bruegmann, K., Prendinger, H., Ishizuka, M.: Extending MPML3D to Second Life. In: H. Prendinger, J. Lester, M. Ishizuka (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 5208, pp. 281–288. Springer Berlin / Heidelberg (2008)
19. Jan, D., Roque, A., Leuski, A., Morie, J., Traum, D.: A virtual tour guide for virtual worlds. In: Proceedings of the 9th International Conference on Intelligent Virtual Agents, IVA '09, pp. 372–378. Springer-Verlag, Berlin, Heidelberg (2009)
20. Bogdanovych, A., Rodriguez-Aguilar, J.A., Simoff, S., Cohen, A.: Authentic interactive reenactment of cultural heritage with 3D virtual worlds and artificial intelligence. Applied Artificial Intelligence **24**(6), 617–647 (2010)