

Hospital Workflow Simulation Client Implementation in Unity

by

Alexander Crichton

N6878296

15/06/2014

Unit Code: INN693

Unit Coordinator: Dr Ernest Foo

Project Supervisor: Dr Ross Brown

TABLE OF CONTENTS

1	Introduction	1
2	Project Background	2
2.1	Technical Overview	2
2.1.1	Client Simulation	2
2.1.2	Database	3
2.1.3	Socket Server	3
2.1.4	Web Server	3
2.1.5	Workflow Manager	3
2.2	Reimplementation Motivations	3
2.3	Chosen Development Solution	4
3	Implementation	5
3.1	Scope	5
3.2	OpenSim to Unity Porting Process	5
3.2.1	Understanding the Implementation	5
3.2.2	Deconstructing the Implementation	5
3.2.3	Constructing the New Implementation	8
3.3	Graphic Client	8
3.3.1	Environment	8
3.3.2	Graphic User Interface	9
3.3.3	Simulation Control	10
3.3.4	Asset Interaction	10
3.4	Simulation Model	11
3.4.1	Simulation Structure	11
3.4.2	Simulation Update	12
3.4.3	Work Item Completion	13
3.4.4	Asset Service Routine Handling	13
4	Execution	15
4.1	Relevant Interface Overview	15
4.1.1	Veis Java Socket Server Console Interface	15
4.1.2	YAWL Web Interface	15
4.1.3	World State Table View	17
4.1.4	Asset Service Routine Table View	17

4.2	Unity Application Demonstration	17
4.2.1	Client Start	18
4.2.2	Connecting to the Veis Java Socket Server	18
4.2.3	Registering as a Participant.....	18
4.2.4	Launching the Case	18
4.2.5	Completing the First Work Item	20
4.2.6	Completing the Remaining Work Items.....	26
4.2.7	Completing the Case	30
4.2.8	Launching a Subsequent Case.....	31
4.2.9	Cancelling a Case.....	32
5	KNOWN ISSUES	34
5.1	Issues Specific to this Implementation	34
5.1.1	Unity DLL Conflicts	34
5.1.2	Unity Type Threading Issues	35
5.2	Issues Not Specific to this Implementation	36
5.2.1	World State Reset Not Automated	36
5.2.2	Veis Database Missing Some Asset Methods	36
5.2.3	Simulation Client Connection with Veis Java Socket Server Unreliable	38
5.2.4	Veis Java Socket Server Unnecessarily Computationally Intensive	39
6	Future Development.....	42
6.1	Web Interface Integration	42
6.2	Networking.....	42
	References	43
	Appendix A Project Setup Instructions	44
	Web Server.....	44
	YAWL.....	44
	Veis Java Socket Server	45
	Unity Application	45
	Appendix B Veis Java Socket Server Communication Handling.....	46

1 INTRODUCTION

The project referred to in this report is a continuation of previous efforts. The long-term goal of the project is to produce an interactive and collaborative virtual environment capable of simulating hospital scenarios for training and education purposes. The simulation utilises a Business Process Management Workflow tool to handle scenarios and to distribute tasks between participants. The client for user interaction was originally implemented in OpenSim.

This report details the re-implementation of the client in Unity. Section 2 outlines the project's existing architecture, motivations for the client migration, and justification for continuing development in Unity. Section 3 explains the new Unity client implementation. Section 4 demonstrates the completed functionality of the new Unity client. Section 5 mentions any significant known issues. Section 6 discusses some possible future features.

2 PROJECT BACKGROUND

Instructions for setting up the project can be found in

Appendix A.

2.1 Technical Overview

Figure 1 illustrates the general project architecture. It is logically and conceptually simpler to imagine a small hierarchy which is controlled by the Client Simulation (see Figure 2).

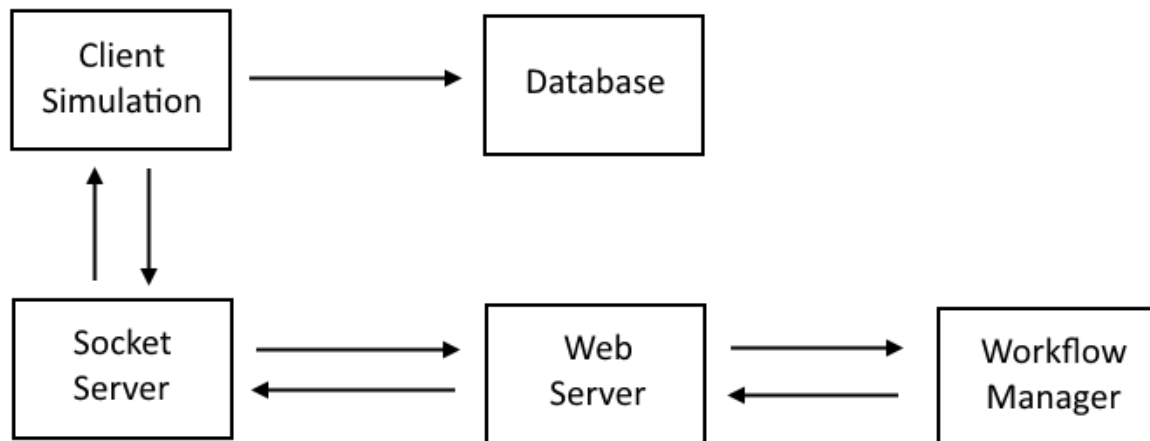


Figure 1. Top level project architecture.

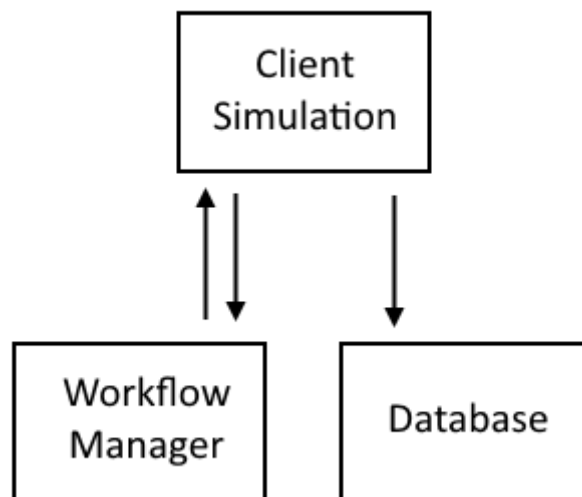


Figure 2. Project logical hierarchy controlled by the Client Simulation.

2.1.1 Client Simulation

The Client Simulation (previously implemented in OpenSim) is technically one application, but should be logically regarded as having three parts: a Graphic Client, a Web Interface, and a Simulation Model (see Figure 3). The Graphic Client is the virtual environment that the user perceives. The Web Interface is how the user affects the state of assets in the environment; it is what updates the database. The Simulation Model handles all of the workflow tracking on the client-side, as well as all of the communication with the Workflow Manager and reading the database for state changes.

2 Project Background

Note that the separation of the Graphic Client and the Simulation Model is only logical; the Graphic Client is a standalone application that contains and references the Simulation Model via DLLs.

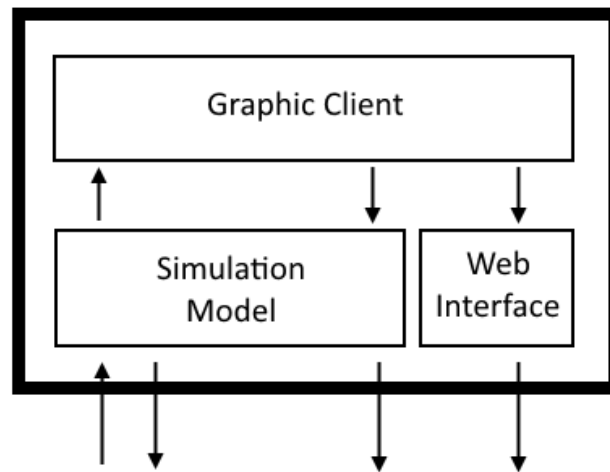


Figure 3. Logical architecture for the Client Simulation.

2.1.2 Database

The database consists of three MySQL databases: `veis_knowledge_base`, `veis_logging`, and `veis_world_states`. The Knowledge Base holds all of the constant pre-set values for the Simulation assets. The Logging database just records events that occur. The World State database holds the dynamic content such as asset states and asset actions to be performed, which the Web Interface updates and the Simulation Model reads.

2.1.3 Socket Server

The Veis Java Socket Server is a small application that handles communication between the Simulation, which is a .NET application, and the Workflow Manager, which is a Java application. A more sophisticated depiction of the communication process between the Simulation and YAWL can be found in Appendix B.

2.1.4 Web Server

This is a simple Windows, Apache, MySQL, and PHP (WAMP) server. The server is required for the Workflow Manager, but not for the database which is directly accessed by the Simulation.

2.1.5 Workflow Manager

The Workflow Manager is an application called YAWL. It is a tool for defining and executing complex workflows. Its role in the project is to delegate tasks to participants for them to complete. It posts workflow data which is handled by the Veis Java Socket Server and passed to the Simulation. YAWL does not interact with the database; it is not aware of the state of workflow items and must be notified by the Simulation when they have been completed.

2.2 Reimplementation Motivations

The previous client implemented in OpenSim was functional but littered with problems. It was comprised of an OpenSim server and a graphic client for user interaction. The OpenSim server is a large, pre-release, open source application with many features, a large portion of which are not relevant to this project. The graphic client was one of a number of third-party open source applications for connecting to virtual worlds.

2 Project Background

The previous client was unreliable, and it was in the long-term interest of the project to investigate an alternative development solution with stability, support, and efficient, dedicated tools.

2.3 Chosen Development Solution

Many possible solutions for developing graphic clients exist, but they are not all equally suited to this project. The optimal development solution needs to suit both the technical needs and the relatively small scale of the project. The previous implementation in OpenSim suggests a solution with a toolset more closely aligned with the needs of the project.

The range of video game development tools offers more relevant and effective solutions. There are many to consider: Unreal Development Kit, CryENGINE, Game Maker Studio, and Source Engine are some of the more notable ones. The most appropriate has been deemed to be Unity for its relative ease of use and comparatively high productivity potential for small development environments.

3 IMPLEMENTATION

This section details the implementation process; including the implementation scope, the process of porting the previous implementation from OpenSim to Unity, and key aspects of the implementation design of the Unity application.

3.1 Scope

The feature scope for this implementation was kept small as it was hard to predict what difficulties would arise during the migration from OpenSim to Unity. The goal was to recreate the basic functionality that was present in the previous implementation.

It was to be hosted locally; it was not necessary to be able to operate over a network. This meant it was not necessary to allow multiple simultaneous users. There was no need to cater for more than one case or more than one player avatar, though the user needed to be able to complete the case by completing its work items.

All assets pertaining to the existing case were to be implemented. The method of interacting with assets was via the existing web interface. Assets that could move needed to retain that functionality.

The virtual environment needed to update with the Simulation. The environment could be rudimentary shapes, however it would ideally resemble a hospital and be navigable by the avatar.

3.2 OpenSim to Unity Porting Process

This section describes the approaches used in porting the OpenSim implementation to its Unity implementation. This section does not mention the many issues and roadblocks during the reimplementing; any issues of significance are detailed later in section 5.

3.2.1 *Understanding the Implementation*

Initially some time was spent using the OpenSim implementation to get a general idea of how it was intended to work and how things interacted. It was useful to learn what was actually necessary to the Simulation. Many superfluous objects such as coffee machines and other patients could be eliminated as they didn't form the interactive part of the scenario.

3.2.2 *Deconstructing the Implementation*

Understanding how the Simulation was perceived by the user was the first task. Understanding how the Simulation worked underneath was the next, much longer task. Much time was spent digging around lines of code trying to understand how pieces fit together, and how they corresponded to the user-perceived application.

Especially helpful was the ability to generate dependency graphs for the various libraries and their classes. The abstracted simulation classes are contained in `Veis` and `Veis.Data`, they are referenced by `Veis.OpenSim` (see Figure 4). `Veis.OpenSim` is what gets loaded by OpenSim as an extension to provide specific functionality. `Veis` and `Veis.Data` were set up to be non-implementation-specific. This meant that they could be mostly left alone, whereas `Veis.OpenSim` would need reimplementing.

Visualising the namespaces, their classes, and their dependencies (see Figure 5 and Figure 6) was useful for determining how the implementation worked. It was clear that most of the work happened in `OpenSimYAWLSimulation`, `ControllableNPCModule`, and `SceneService`. The other

3 Implementation

classes were utility classes or provided functions outside the scope of the new implementation, such as chat handling.

OpenSimYAWLSimulation was an extension of Veis.Simulation.Simulation, most of which would need to be re-implemented in the Unity project. This class tracks all of the participants in the Simulation, as well as work items and their conditions for completion. It also handles all communication to the database and is responsible for initiating updates to the world state.

ControllableNPCModule is actually an extension of an OpenSim class, NPCModule. This class holds information about NPCs in the OpenSim world, such as ID, location, and appearance. It does not hold anything directly related to the Simulation like work items. It was included in the OpenSim implementation as a way to mirror the NPCModule, making it easier to access NPC information that OpenSim requires when performing actions on them. It was deemed not to be necessary for the new implementation; partly because a lot of it was specific to OpenSim, and partly because the new NPCs would not yet need that level of sophisticated data.

The SceneService was used to trigger functions within OpenSim such as moving an NPC. It implemented the ISceneService interface which is non-implementation-specific. This meant that it could be re-implemented to serve the same functions, but using Unity-specific methods.

After deconstructing the OpenSim implementation it was determined that only two classes would really be needed as the foundation for the new implementation: an extension of Veis.Simulation.Simulation and a class that implements ISceneService.

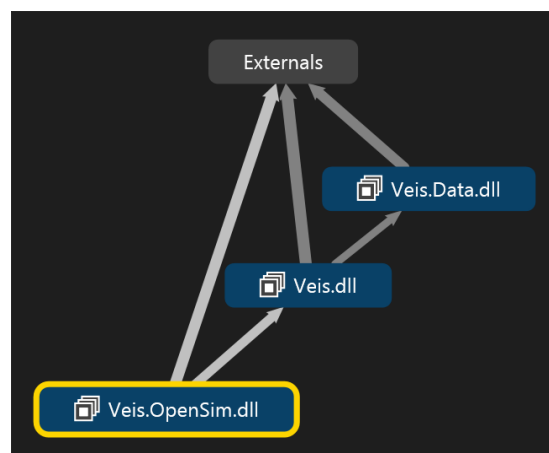


Figure 4. The abstracted simulation classes contained in Veis and Veis.Data, and referenced by Veis.OpenSim.

3 Implementation

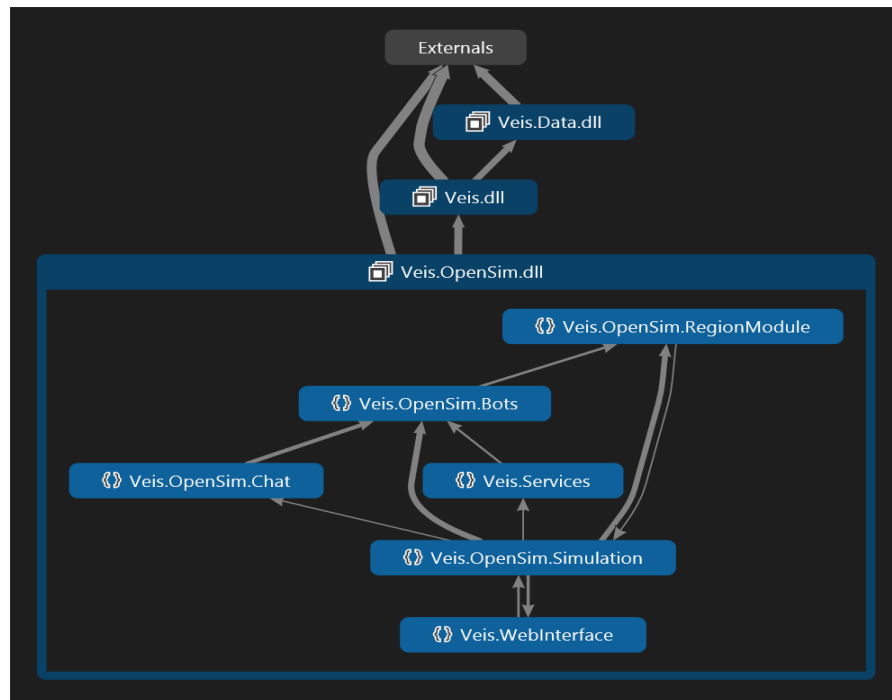


Figure 5. The namespaces within Veis.OpenSim.

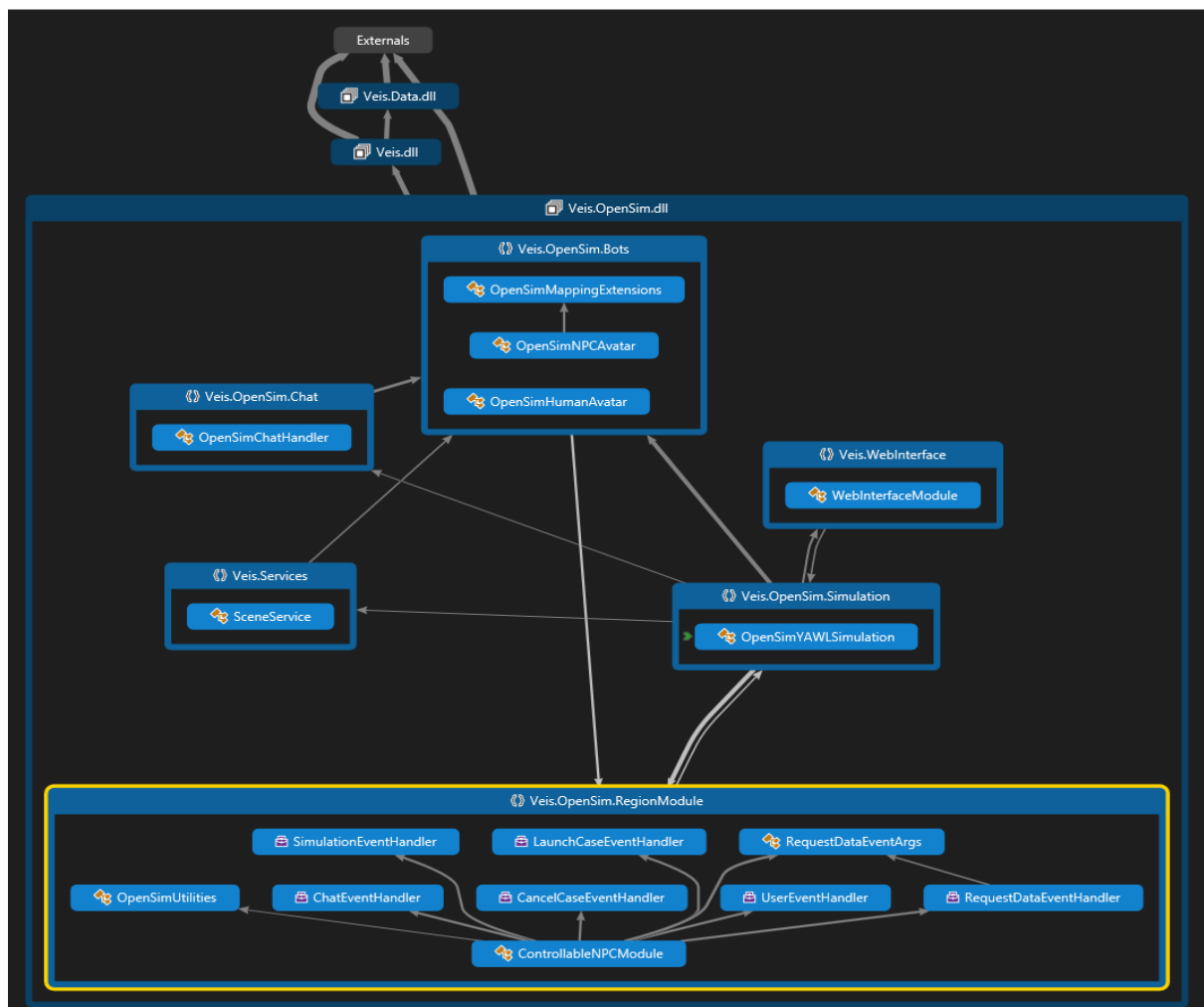


Figure 6. The classes within Veis.OpenSim and their relationships between each other.

3 Implementation

3.2.3 Constructing the New Implementation

The first approach was to copy over the necessary classes and attempt to remove unneeded dependencies. This turned out to be a very inefficient approach, because everything instantly broke, and it was eventually abandoned. The much more successful approach was to identify small pieces of the Simulation and add them together one at a time.

Initially a console application was used for investigating the communication between the Simulation and YAWL. This was done early because it was one of the more obvious and intuitive sections in the code. After that, work began on the rest of the Simulation, and development moved into a Unity project. This happened early in the process to make the most of Unity's built-in logging, and to deal with any compatibility issues that might undermine future progress.

3.3 Graphic Client

The Graphic Client is a Unity application which provides the virtual environment for user interaction. This section describes what was implemented as part of the Graphic Client.

3.3.1 Environment

The environment resembles a hospital containing the rooms required by the asset specifications: the Emergency Room, Examination Room, Intensive Care, Radiology Room, and a final unnamed room (see Figure 7). The avatar starts in the bottom-left corner in the Emergency Room; this is where the asset for the first work item is located.

Assets are distinguishable by their bright red, green, or blue colour. They are environment objects that the user can interact with. Place Markers are the less obvious transparent green objects. They hold possible locations for certain assets. They are not for user interaction.

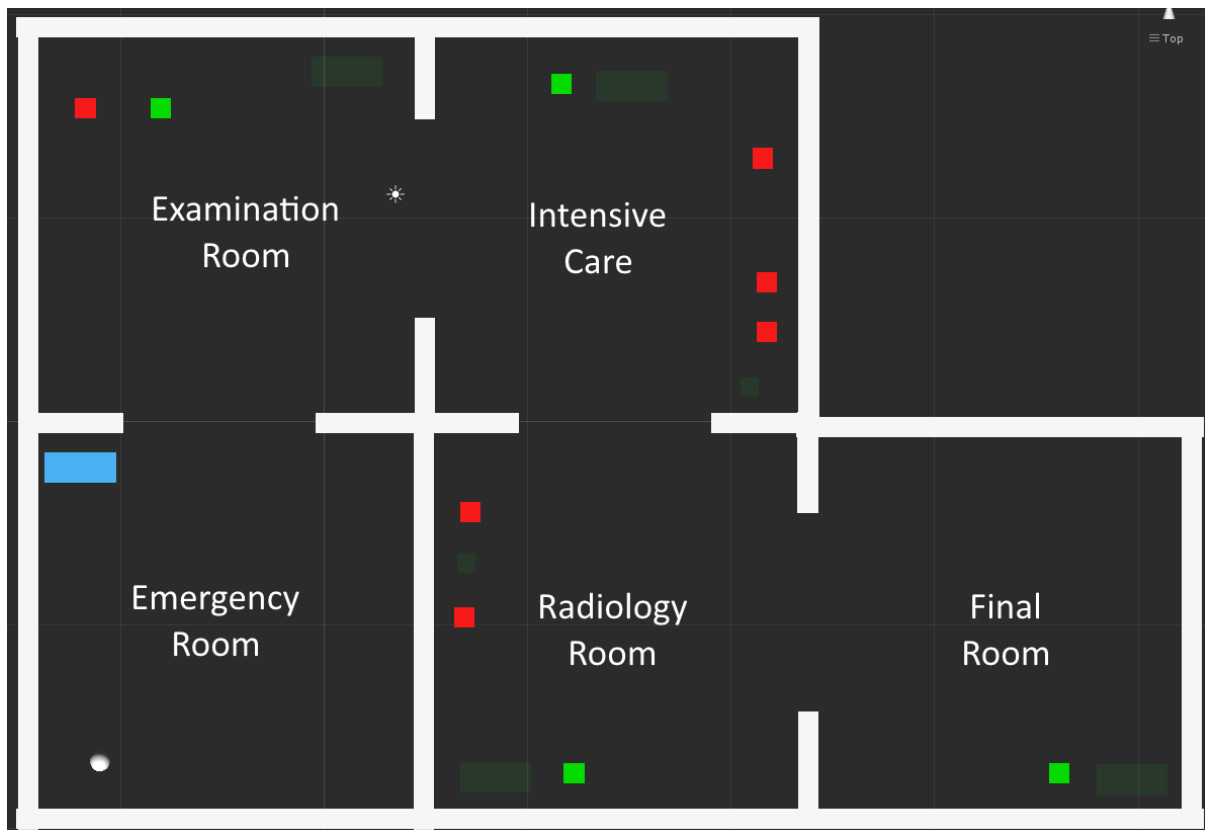


Figure 7. The Unity application environment Layout.

3 Implementation

3.3.2 Graphic User Interface

Figure 8 shows the Graphic User Interface (GUI) as seen by the user. The user sees the hospital environment and their avatar in the third person. The available simulation and case controls are located in the bottom-left corner of the interface (see Figure 9). Information about any current case and work items is located in the top-left corner of the interface. Particular assets can be easily identified by their asset name which is rendered as floating text layered on top them (see Figure 11).

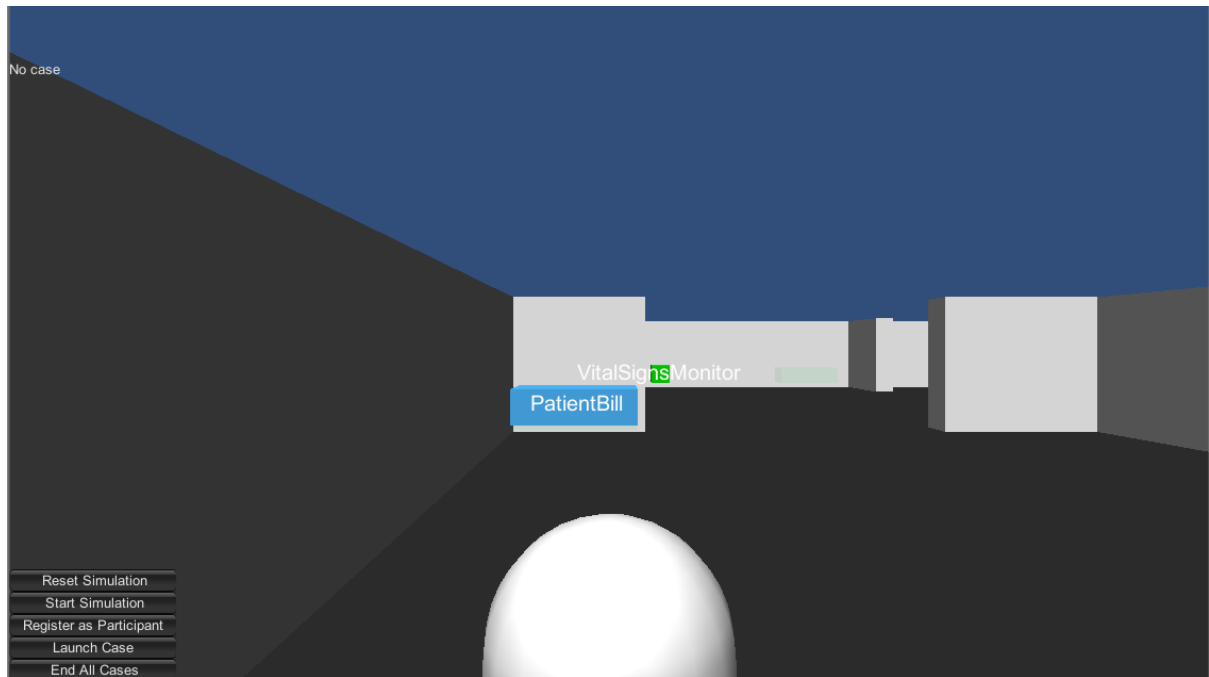


Figure 8. The Unity application Graphic User Interface.

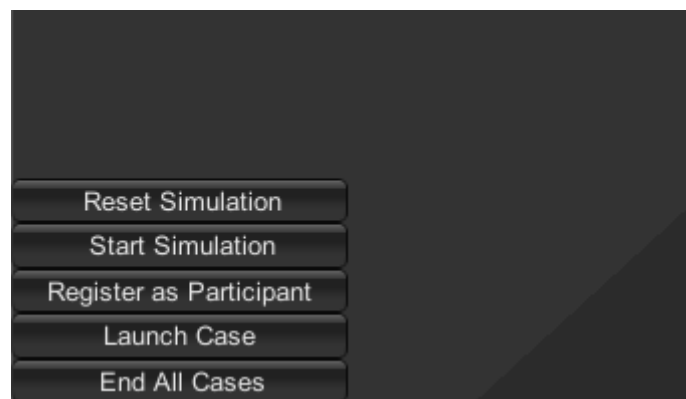


Figure 9. The Simulation controls available to the user.

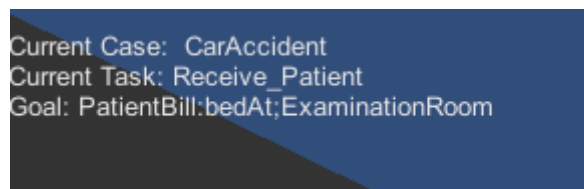


Figure 10. Case and work item information.

3 Implementation

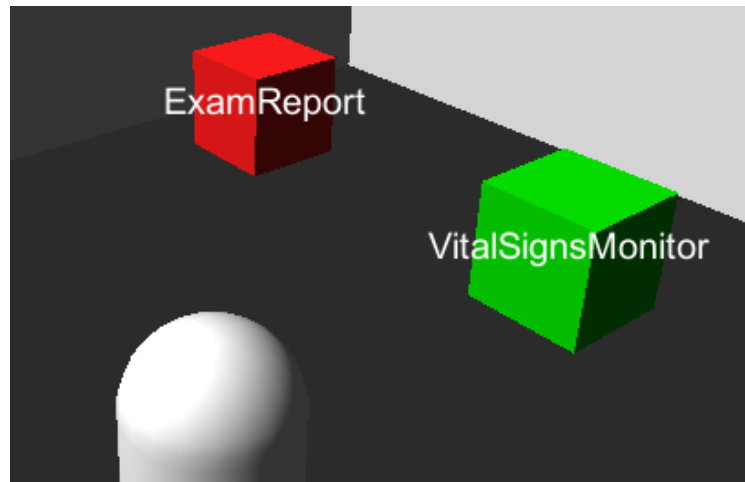


Figure 11. Floating labels for asset identification.

3.3.3 Simulation Control

The Simulation controls previously shown in Figure 9 are used to get the Simulation going.

The Reset Simulation button just sets Simulation variables to their initial values. This affects local data tracking like registered participants and current work items; it does not affect externally stored data like asset states and work item states, which are stored in the database and YAWL respectively. The Start Simulation button is currently redundant because when the Simulation resets it performs that function as well.

The Register as Participant button assigns a case participant to the user's avatar. In this implementation there is only one case participant so the user is not able to make a selection. There is currently no way to unregister from being a participant.

The Launch Case button notifies YAWL to start the case, which is also sent back and stored in the Simulation.

The End All Cases button notifies YAWL to end any running case. This only ends locally tracked cases; it does not end cases that might have been started in YAWL outside of the Simulation.

3.3.4 Asset Interaction

Assets in the environment can be interacted with by clicking on them. This will launch the Asset Interaction Web Interface (see Figure 12) which uses asset details to load its state and available methods. The Asset Interaction Web Interface is the only way the user can affect asset states. It directly updates the World State database, which is then read by the Simulation Model. It updates asset values and inserts asset service routines if an action needs to be performed on an asset, such as moving it.

The screenshot shows a web interface titled "PatientBill - Asset Method List". Inside, there is a box labeled "Current state of 'PatientBill'" containing a list of three items: "1) bedAt 'EmergencyRoom'", "2) isHearingTested 'False'", and "3) isVisuallyExamined 'False'". Below this box, the text "Please select a method:" is followed by a dropdown menu labeled "SELECT METHOD" with a downward arrow. At the bottom right of the interface is a "Submit" button.

Figure 12. The first page of the Asset Interaction Web Interface.

3.4 Simulation Model

This section describes the relevant parts of the Simulation that have been extended for this implementation. This section does not describe the entire Simulation Model that is found in the Veis libraries. Note that the Simulation is part of the Graphic Client but is explained as its own entity.

The Simulation tracks participants, cases, and work items. It queries the database, updates work items, communicates with YAWL, and performs actions on objects in the world. Note that the Simulation does not set World State database values, it only reads them and acts on any changes. Modifications to the World State database are handled by the Asset Interaction Web Interface (see section 3.3.4).

3.4.1 Simulation Structure

The Simulation contains several parts: the user avatars, the Workflow Provider, Database Services, and World State Services (see Figure 13). The Workflow Provider keeps track of running cases and work items assigned to participants. The Database Services manage access to the Veis database; they check them frequently for changes and notify the World State Services when they need to update. The World State Services initiate updates to work items and the world.

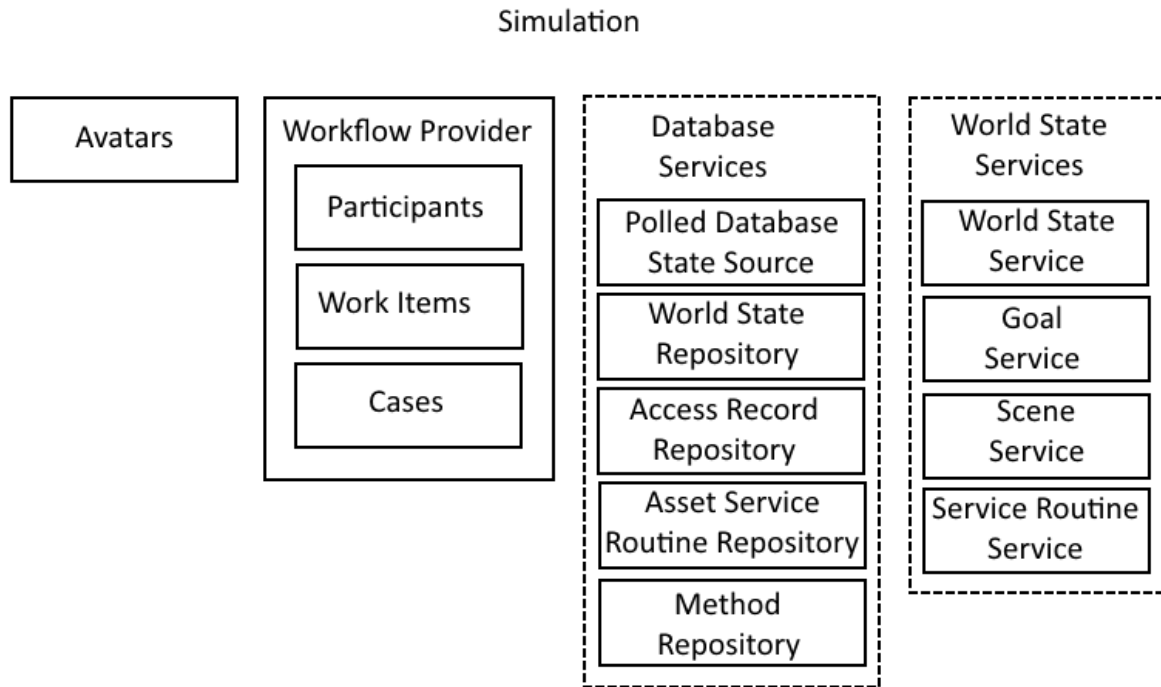


Figure 13. Simulation Structure.

3.4.2 Simulation Update

The Simulation polls the database on a timer and notifies relevant services if they need to do anything (see Figure 14 and Figure 15). The relevant services are the Goal Service, which deals with work item Goals, and the Service Routine Service, which invokes any services found in the Asset Service Routine table of the database, such as moving an asset.

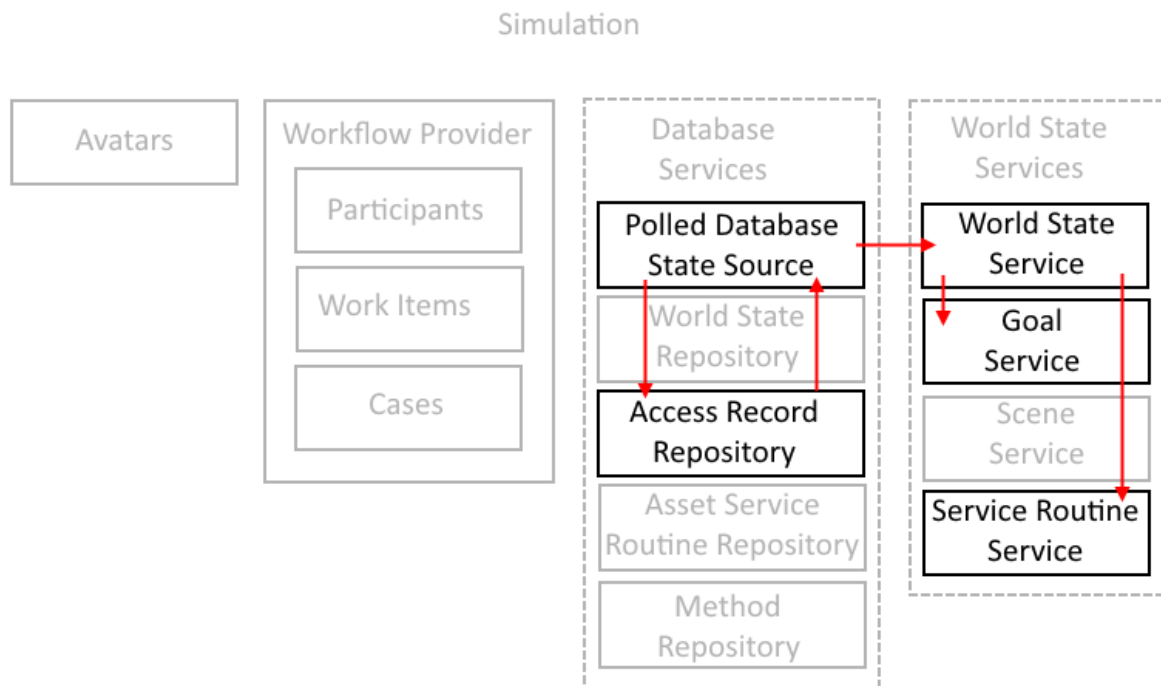


Figure 14. Simulation update process.

3 Implementation

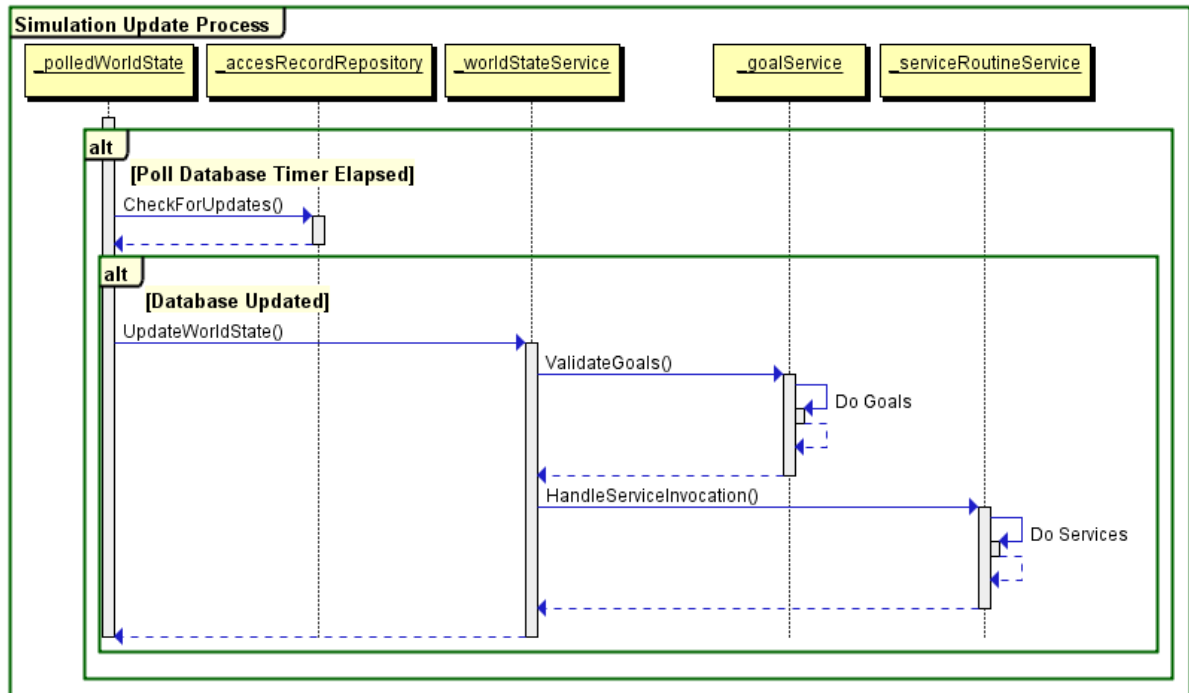


Figure 15. Simulation update process.

3.4.3 Work Item Completion

Work items are completed when their Goal is satisfied (see Figure 16). When a work item is completed the Simulation sends a message to YAWL, which will send back a new work item, or if it was the last work item, will complete the case.

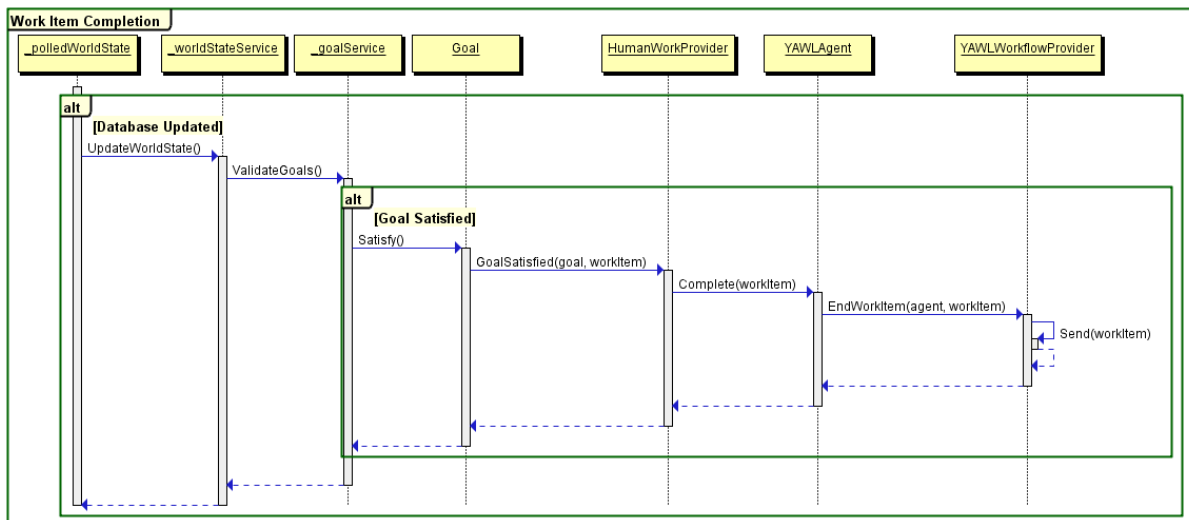


Figure 16. Work item completion process.

3.4.4 Asset Service Routine Handling

Asset Service Routines are what affect objects in the world. They are inserted into the database by the Asset Interaction Web Interface and read by the Simulation. Service Routines are passed to their respective handlers by the ServiceRoutineService. Currently the only handler is the MoveObjectHandler, which notifies the SceneService of actions that need to be performed in the world (see Figure 17). Note that the Service Routine is added to SceneService by the

3 Implementation

MoveObjectHandler, but the action is left to be handled by the main Unity thread (this is explained later in section 5.1.2).

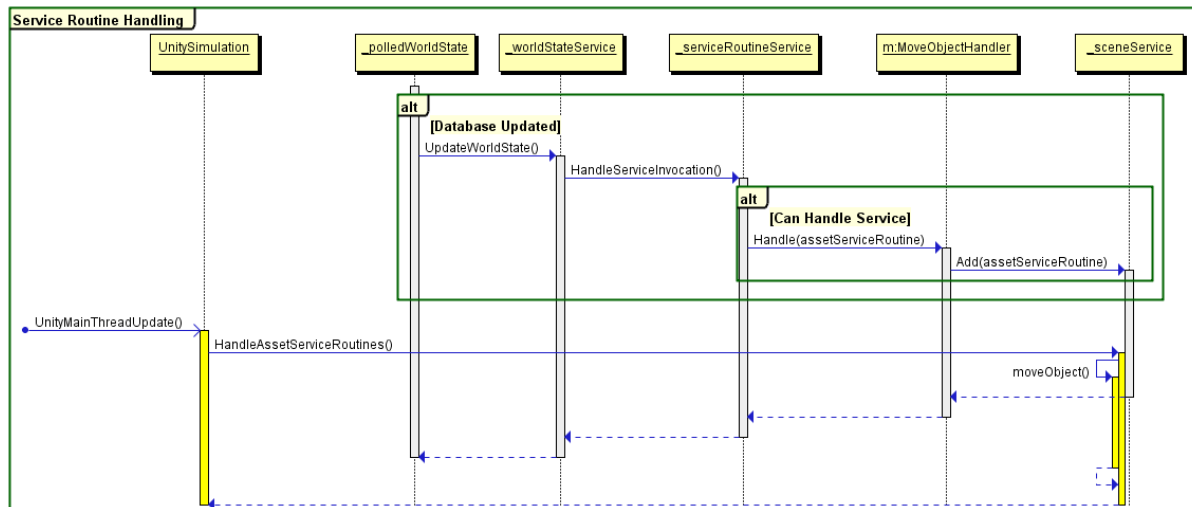


Figure 17. Asset Service Routine being sent to the SceneService and then handled on the main thread.

4 EXECUTION

This section demonstrates the completeness of the Unity application implementation. Screenshots of the application and other relevant interfaces will be used extensively to convey its state.

4.1 Relevant Interface Overview

This section explains any other relevant interfaces that will be shown.

4.1.1 *Veis Java Socket Server Console Interface*

Figure 18 shows the console interface that is part of the Veis Java Socket Server. As this application passes messages between the Simulation and YAWL it also relays some information through this console interface for human viewing. This interface will be used to confirm some state changes as they are communicated between the Simulation and YAWL.

```
Log file: ListennedLog.txt
Started socket server on port 4444 to listen to client.
Started socket server on port 1527 to listen to YAWL server.
SESSION-IX: <success/>
SESSION-IA: ef111477-0163-44fa-8d99-335ea1fa8118
SESSION-IB: b816c069-72ed-4cd5-8ddd-e21007b9aaef
SESSION-RS: d5006d2e-4241-4bce-b97b-675d396eb2b8
SESSION-WQ: 159cb2f8-0933-4d63-9183-c275b9a8d989
```

Figure 18. Veis Java Socket Server console window immediately after opening

4.1.2 *YAWL Web Interface*

Figure 19 shows the Cases tab in the YAWL Web Interface. Here, case specifications can be loaded, cases can be launched, and running cases can be viewed or cancelled. This interface will be used to confirm that cases are running or not.

Figure 20 shows the Admin Queues tab, and the Worklisted sub-tab, in the YAWL Web Interface. Here, work items that are being handled by YAWL can be monitored. This interface will be used to confirm the state of various work items.

4 Execution

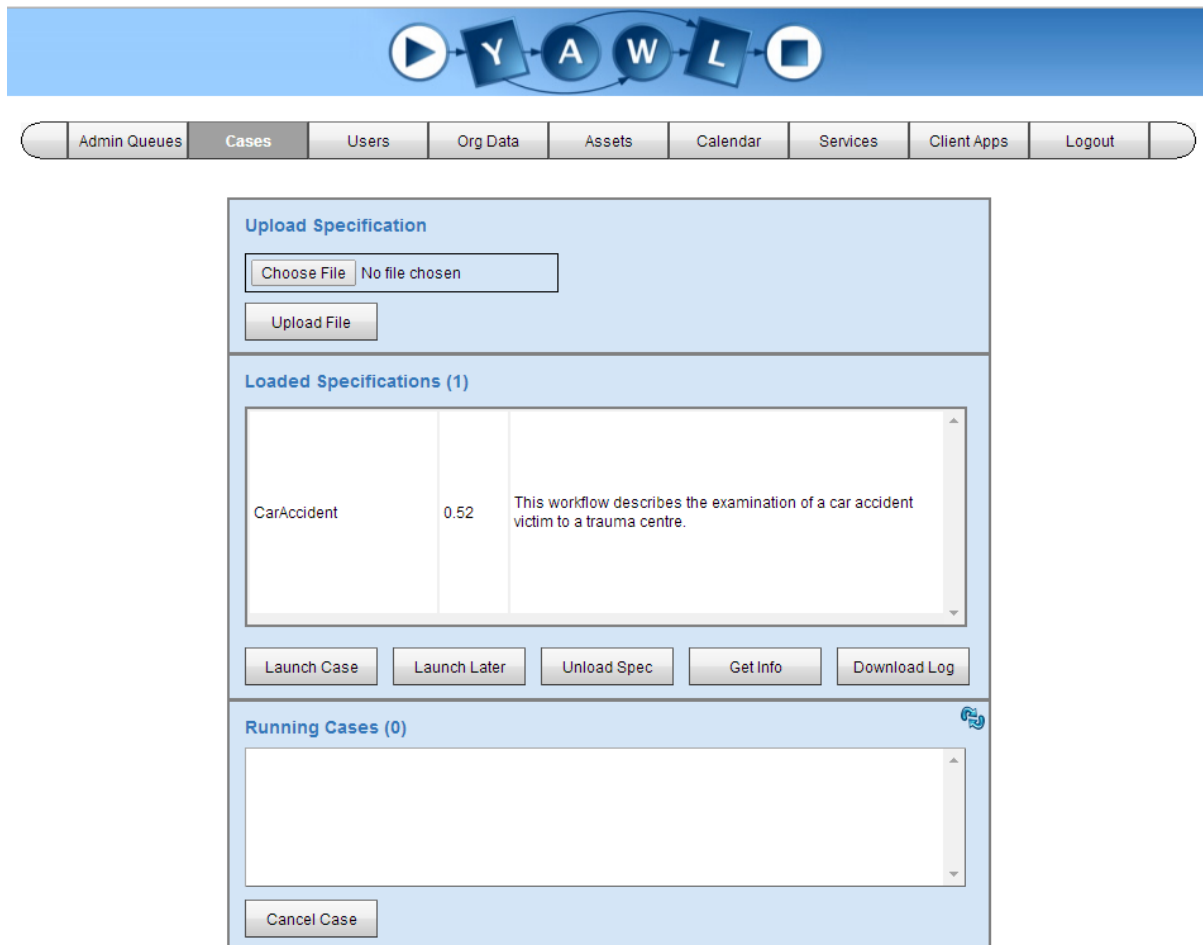


Figure 19. YAWL web interface view of case management section with no case running

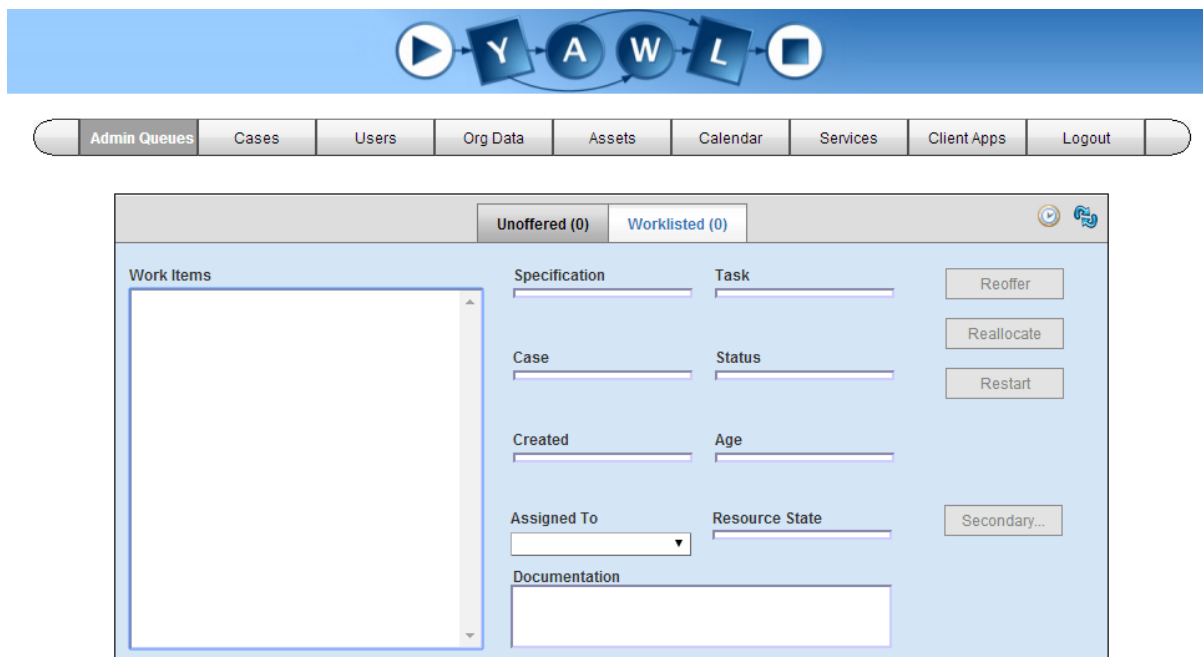


Figure 20. YAWL web interface view of current work items with no case running

4 Execution

4.1.3 World State Table View

Figure 21 Shows a view of the World State table, part of the Veis MySQL database, accessed through the PHPMYAdmin Web Interface. Simulation assets are defined in the Knowledge Base, and the World State stores various current states for those assets. This view will be shown to confirm the state of assets. Note that some of the assets listed in this table are not relevant to this simulation.

←T→		world_key	asset_name	predicate_label	value	TIMESTAMP
<input type="checkbox"/>	Edit Copy Delete	1	VitalSignsMonitor	bandAt	PatientArm	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	bedAt	EmergencyRoom	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	CoffeeCup	cupAt	Table	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	CoffeeCup	cupContains	Empty	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	RequestPathology	forPatient	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	BloodSampleVials	Has	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	isHearingTested	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	isVisuallyExamined	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	VitalSignsMonitor	mouthpieceAt	PatientHead	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	GROComputer	orderLogged	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	GROComputer	orderProcessed	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	ExamReport	reportTo	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	RequestXRay	requestApproved	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	RequestXRay	requestAt	Critical	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	RequestXRay	requestForBodyPart	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	VitalSignsMonitor	sensorAt	PatientArm	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	Truck	truckAt	Entrance	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	Truck	truckLoadStatus	Loaded	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	MachineXRay	xrayAt	XRaysMiddle	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	MachineXRay	xrayForBodyPart	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	ReportXRay	xrayReportTo	None	2014-01-26 03:29:49

Figure 21. MySQL database table view of asset world states with its initial values

4.1.4 Asset Service Routine Table View

Figure 22 shows a view of the Asset Service Routines table. This table holds actions to be performed on assets such as moving to a new position. The actions are handled directly by the Simulation and then removed; as such the table is usually empty.

←T→		key	priority	asset_key	service_routine	world_key	TIMESTAMP
<input type="checkbox"/>	Edit Copy Delete	132	1	2297ca59-bfe2-49ac-9265-e4af9b35b5fb	Move:Bed to=ExaminationRoom	1	2014-06-09 23:12:42

Figure 22. MySQL database table view of asset service routines.

4.2 Unity Application Demonstration

This section demonstrates the functional aspects of the Unity application. It makes extensive use of figures to confirm the validity of said functionalities. This demonstration assumes the following preconditions:

- An appropriate webserver is being hosted locally with MySQL and PHP
- The necessary Veis MySQL database has been set up
- The YAWL Web Service is running
- The necessary hospital case specification has been loaded
- The Veis Java Socket Server is running

4 Execution

4.2.1 Client Start

As shown in Figure 8 when the Unity application starts the user is positioned in the hospital environment. The section they start in is the Emergency Room. The only Simulation asset in this room is PatientBill. Initially, the user is not registered as a participant, there is no case running, and there are no work items.

4.2.2 Connecting to the Veis Java Socket Server

Figure 23 confirms that the Unity application successfully establishes communications with the Veis Java Socket Server. The client is now able to send and receive messages to and from YAWL.

```
Log file: ListennedLog.txt
Started socket server on port 4444 to listen to client.
Started socket server on port 1527 to listen to YAWL server.
SESSION-IX: <success/>
SESSION-IA: 9a79b937-f6a4-475a-9ea6-61b63a3cc9c7
SESSION-IB: b1150d76-203f-41dd-b280-255fd786b54d
SESSION-RS: d0ee6903-5303-4524-ad44-70909a20d0a5
SESSION-WQ: 5eef14e7-e88f-4783-b92d-10573ebd37a2
A client connected on port 4444 Current connections: 1
```

Figure 23. A Simulation client has successfully connected to the Veis Java Socket Server.

4.2.3 Registering as a Participant

Figure 24 shows the user pressing the Register as Participant button in the Unity application interface

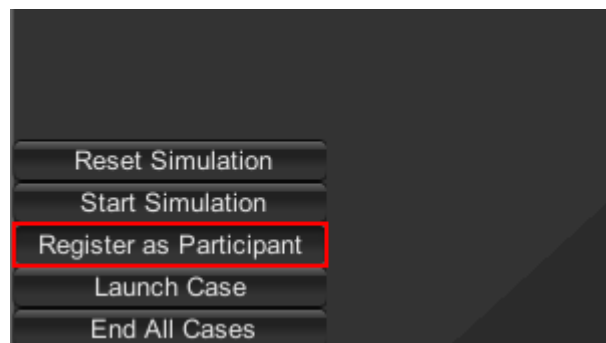


Figure 24. The user assigns a registered case participant to their avatar.

4.2.4 Launching the Case

Figure 25 shows the user pressing the Launch Case button. Figure 26 shows the application interface updates to show the case, the first work item, and the work item's condition for completion. The work item will not be completed until the World State table has been updated so that it matches those values.

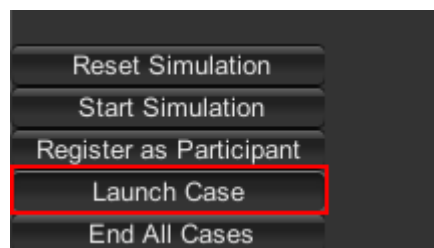


Figure 25. The user launches the case.

4 Execution

```
Current Case: CarAccident
Current Task: Receive_Patient
Goal: PatientBill:bedAt;ExaminationRoom
```

Figure 26. The interface updates to show the newly launched case, the user's first work item, and the work item's condition for completion.

Figure 27 confirms the corresponding case has been launched and is running in the YAWL Web Interface. Figure 28 shows the corresponding work item has been delegated. Figure 29 shows the messages received from YAWL and sent to the Simulation by the Veis Java Socket Server. The first section is the case being sent from YAWL; the second section is the first work item being sent from YAWL; the third section is the Socket Server sending the work item to the Simulation.

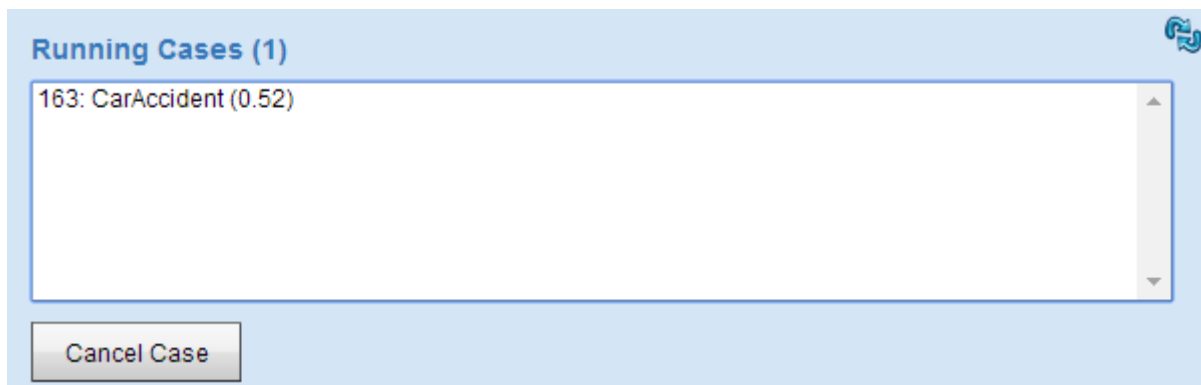


Figure 27. The YAWL Web Interface shows the case running.

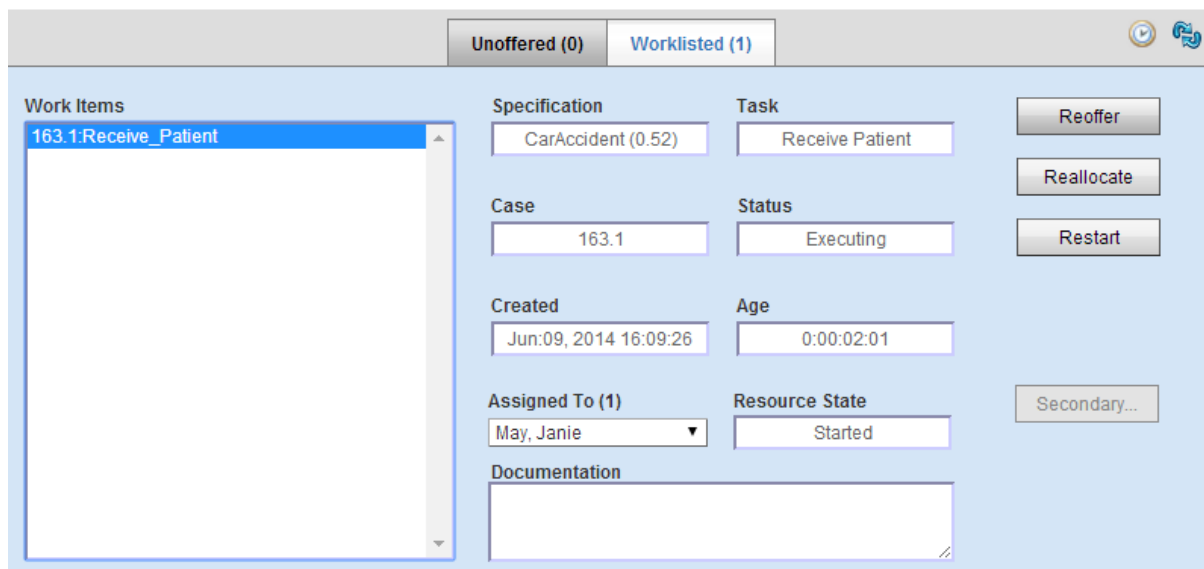


Figure 28. The YAWL Web Interface shows the work item delegated to the user.

```

A client connected on port 4444 Current connections: 1
YAWL server just say something to me...

[====YAWLListenerIX BEGIN RECEIVING=====]
specVersion=0.52&caseID=163&action=0&specID=UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d&specURI=CarAccident&preCheck=true
[====YAWLListenerIX END RECEIVING=====]

YAWL server just say something to me...

[====YAWLListenerIX BEGIN RECEIVING=====]
workItem=<workItem><taskId>Receive_Patient</taskId><caseid>163</caseid><uniqueid>00000000000000000000000000000000</uniqueid><taskname>Receive_Patient</taskname><documentation></documentation><specifier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specifier><specversion>0.52</specversion><specuri>CarAccident</specuri><status>Enabled</status><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/><enablementTime>Jun:09, 2014 16:09:26</enablementTime><enablementTimeMs>1402294166389</enablementTimeMs></workItem>&data=<?xml version="1.0" encoding="UTF-8"?>
<Trauma_Centre_Patient_Examination />
&action=1&preCheck=true
[====YAWLListenerIX END RECEIVING=====]

Accepted: <success/>
Started: <workItemRecord><id>163.1:Receive_Patient</id><specversion>0.52</specversion><specuri>CarAccident</specuri><caseid>163.1</caseid><taskId>Receive_Patient</taskId><uniqueid>00000000000000000000000000000001</uniqueid><taskname>Receive_Patient</taskname><documentation></documentation><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/><enablementTime>Jun:09, 2014 16:09:26</enablementTime><firingTime>Jun:09, 2014 16:09:27</firingTime><startTime>Jun:09, 2014 16:09:27</startTime><completionTime/><enablementTimeMs>1402294166389</enablementTimeMs><firingTimeMs>1402294167813</firingTimeMs><startTimeMs>1402294167836</startTimeMs><completionTimeMs/><timertrigger/><timerexpiry/><status>Executing</status><resourceStatus>Started</resourceStatus><startedBy>admin</startedBy><completedBy/><tag/><customform/><logPredicateStarted/><logPredicateCompletion/><specifier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specifier><data><Receive_Patient /></data><updateddata></updateddata></workItemRecord>

```

Figure 29. The Veis Java Socket Server shows the data received from YAWL and sent to the Simulation.

4.2.5 Completing the First Work Item

Figure 30 outlines what is required to complete the first work item. It reads: for the asset PatientBill, its predicate label bedAt must have the value ExaminationRoom. Figure 31 outlines the Unity GameObject that represents the asset PatientBill in the client. The user can press on the GameObject to launch the asset interaction.

```

Current Case: CarAccident
Current Task: Receive_Patient
Goal: PatientBill:bedAt:ExaminationRoom

```

Figure 30. The client interface shows the asset state required to complete the work item

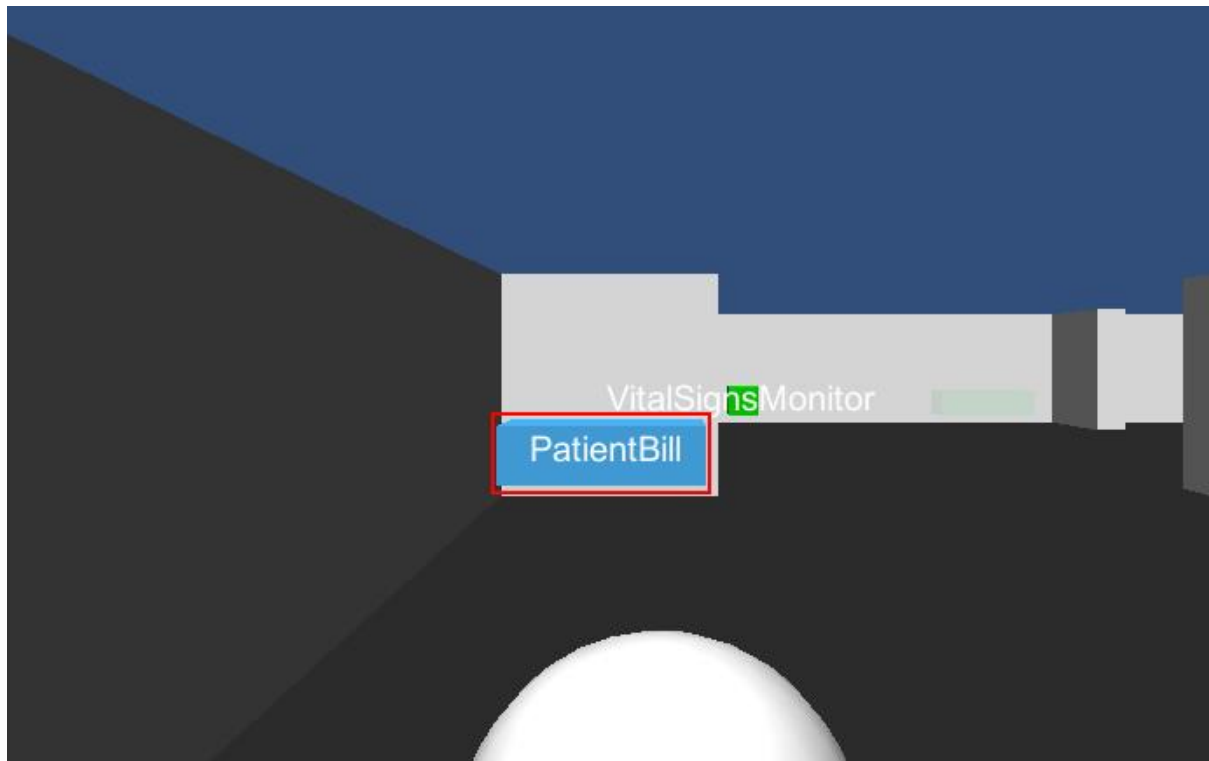


Figure 31. Outlining the asset PatientBill that is part of the first work item.

When the user presses on the PatientBill GameObject the Asset Interaction Web Interface is launched (see Figure 32). It selects all the predicate labels associated with the asset PatientBill and their current values. The user selects the asset method that corresponds with the predicate label for which they intend to change the value (see Figure 33). The user presses the Submit button and the next page loads.

The asset method is selected and the corresponding predicate label's current value is shown (see Figure 34). The user selects from the available values the one they intend to assign to the predicate label (see Figure 35). The user presses the Submit button and the next page loads.

The selected value is assigned to the selected predicate label of the selected asset. The user is notified that the database has been updated successfully (see Figure 36), and the user is finally prompted to close the Asset Interaction Web Interface (see Figure 37).

PatientBill - Asset Method List

Current state of "PatientBill"

- 1) bedAt "EmergencyRoom"
- 2) isHearingTested "False"
- 3) isVisuallyExamined "False"

Please select a method:

SELECT METHOD ▼

Submit

Figure 32. The Asset Interaction Web Interface when launched by pressing on the asset PatientBill.

Please select a method:

SELECT METHOD ▼

SELECT METHOD

Examine Visual

Move Bed

Test Hearing

Submit

Figure 33. The available asset methods for the asset PatientBill.

Method Parameter Lists

Current state of "PatientBill"

- 1) bedAt "EmergencyRoom"

Options for action "Move Bed":

Bed to: EmergencyRoom ▼

Submit

Figure 34. The current value for the selected predicate label bedAt.

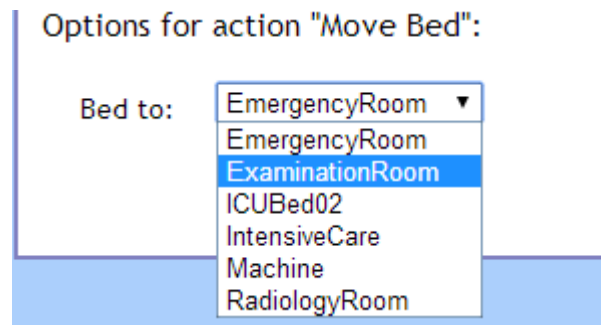


Figure 35. The available values for the predicate label bedAt.

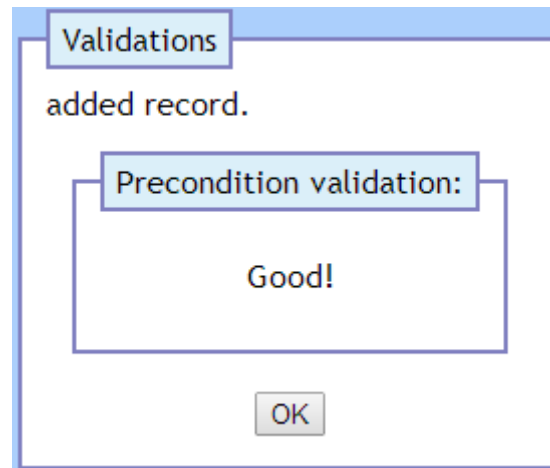


Figure 36. The Asset Interaction Web Interface has updated the corresponding asset value.

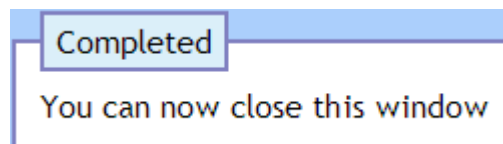


Figure 37. The Asset Interaction Web Interface has been completed for the asset.

The World State table is updated (see Figure 38). The asset PatientBill has its predicate label bedAt changed to ExaminationRoom. An Asset Service Routine is inserted into the Asset Service Routine table (see Figure 39). In this instance the action is to move the asset PatientBill to the location ExaminationRoom.

The YAWL Web Interface shows the previous work item has been removed and new work items have been set (see Figure 40). The Veis Java Socket Server shows the messages from YAWL to the Simulation (see Figure 41). The first two sections indicate the new work items being sent from YAWL; the third section indicates the work item Visual_Examination being sent to the Simulation; the fourth section indicates the previous work item being completed.

4 Execution

←T→		world_key	asset_name	predicate_label	value	TIMESTAMP
<input type="checkbox"/>	Edit Copy Delete	1	VitalSignsMonitor	bandAt	PatientArm	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	bedAt	ExaminationRoom	2014-06-09 16:47:16
<input type="checkbox"/>	Edit Copy Delete	1	CoffeeCup	cupAt	Table	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	CoffeeCup	cupContains	Empty	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	RequestPathology	forPatient	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	BloodSampleVials	Has	None	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	isHearingTested	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	PatientBill	isVisuallyExamined	False	2014-01-26 03:29:49
<input type="checkbox"/>	Edit Copy Delete	1	VitalSignsMonitor	mouthpieceAt	PatientHead	2014-01-26 03:29:49

Figure 38. The World State table has been updated for the asset PatientBill.

←T→		key	priority	asset_key	service_routine	world_key	TIMESTAMP
<input type="checkbox"/>	Edit Copy Delete	132	1	2297ca59-bfe2-49ac-9265-e4af9b35b5fb	Move:Bed to=ExaminationRoom	1	2014-06-09 23:12:42

Figure 39. The move action to be performed on the asset PatientBill.

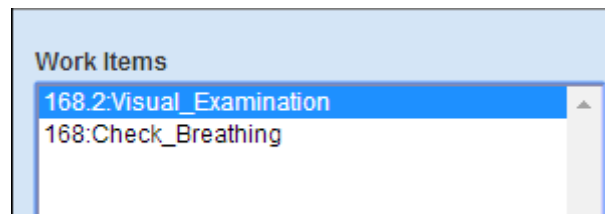


Figure 40. The work item Receive_Patient has been completed and new work has been delegated.

```

YAWL server just say something to me...

[====YAWLListenerIX BEGIN RECEIVING=====]
workItem=<workItem><taskid>Visual_Examination</taskid><caseid>168</caseid><uniqueid>00000000000000000000000000000000E</uniqueid><taskname>Visual_Examination</taskname><documentation/><specidentificier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specidentificier><specversion>0.52</specversion><specuri>CarAccident</specuri><status>Enabled</status><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/><enablementTime>Jun:09, 2014 16:47:18</enablementTime><enablementTimeMs>1402296438526</enablementTimeMs></workItem>&data=<?xml version="1.0" encoding="UTF-8"?>
<Trauma_Centre_Patient_Examination />
&action=1&preCheck=true
[====YAWLListenerIX END RECEIVING=====]

YAWL server just say something to me...

[====YAWLListenerIX BEGIN RECEIVING=====]
workItem=<workItem><taskid>Check_Breathing</taskid><caseid>168</caseid><uniqueid>00000000000000000000000000000000F</uniqueid><taskname>Check_Breathing</taskname><documentation/><specidentificier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specidentificier><specversion>0.52</specversion><specuri>CarAccident</specuri><status>Enabled</status><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/><enablementTime>Jun:09, 2014 16:47:18</enablementTime><enablementTimeMs>1402296438540</enablementTimeMs></workItem>&data=<?xml version="1.0" encoding="UTF-8"?>
<Trauma_Centre_Patient_Examination />
&action=1&preCheck=true
[====YAWLListenerIX END RECEIVING=====]

CompleteWorkItem: <success/>
Accepted: <success/>
Started: <workItemRecord><id>168.2:Visual_Examination</id><specversion>0.52</specversion><specuri>CarAccident</specuri><caseid>168.2</caseid><taskid>Visual_Examination</taskid><uniqueid>00000000000000000000000000000000G</uniqueid><taskname>Visual_Examination</taskname><documentation/><documentation></documentation><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/></codelet><deferredChoiceGroupid/><enablementTime>Jun:09, 2014 16:47:18</enablementTime><firingTime>Jun:09, 2014 16:47:18</firingTime><startTime>Jun:09, 2014 16:47:18</startTime><completionTime/><enablementTimeMs>1402296438526</enablementTimeMs><firingTimeMs>1402296438616</firingTimeMs><startTimeMs>1402296438637</startTimeMs><completionTimeMs/><timertrigger/><timerexpiry/><status>Executing</status><resourceStatus>Started</resourceStatus><startedBy>admin</startedBy><completedBy/><tag/><customform/><logPredicateStarted/><logPredicateCompletion/><specidentificier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specidentificier><data><Visual_Examination /></data><updateddata></updateddata></workItemRecord>
YAWL server just say something to me...

[====YAWLListenerIX BEGIN RECEIVING=====]
workItem=<workItem><taskid>Receive_Patient</taskid><caseid>168.1</caseid><uniqueid>00000000000000000000000000000000D</uniqueid><taskname>Receive_Patient</taskname><documentation/><specidentificier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d</specidentificier><specversion>0.52</specversion><specuri>CarAccident</specuri><status>Completed</status><allowsdynamiccreation>false</allowsdynamiccreation><requiresmanualresourcing>true</requiresmanualresourcing><codelet/><data><Receive_Patient /></data><enablementTime>Jun:09, 2014 16:41:50</enablementTime><enablementTimeMs>1402296110324</enablementTimeMs><firingTime>Jun:09, 2014 16:41:50</firingTime><firingTimeMs>1402296110395</firingTimeMs><startTime>Jun:09, 2014 16:41:50</startTime><startTimeMs>1402296110412</startTimeMs><startedBy>admin</startedBy></workItem>&data=<?xml version="1.0" encoding="UTF-8"?>
<Receive_Patient>
  <Tasks>Receive_Patient</Tasks>
  <Goal>PatientBill:bedAt;ExaminationRoom</Goal>
</Receive_Patient>
&action=1&preCheck=false
[====YAWLListenerIX END RECEIVING=====]

```

Figure 41. The Veis Java Socket Server passing new work items from YAWL to the Simulation.

The Simulation client updates to show the new work item and its predicate label value required to complete it (see Figure 42). The GameObject for the PatientBill asset moves in the Unity Scene (see Figure 43). Its new position is in the Examination Room as specified in the Asset Service Routine table. The service routine action is now removed from the table.

4 Execution

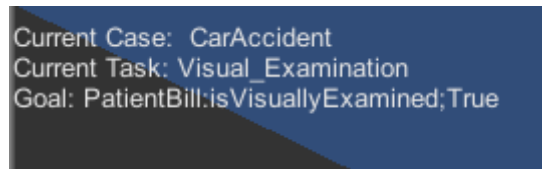


Figure 42. The Simulation client interface has updated to show the new work item *Visual_Examination* and the action required to complete it.

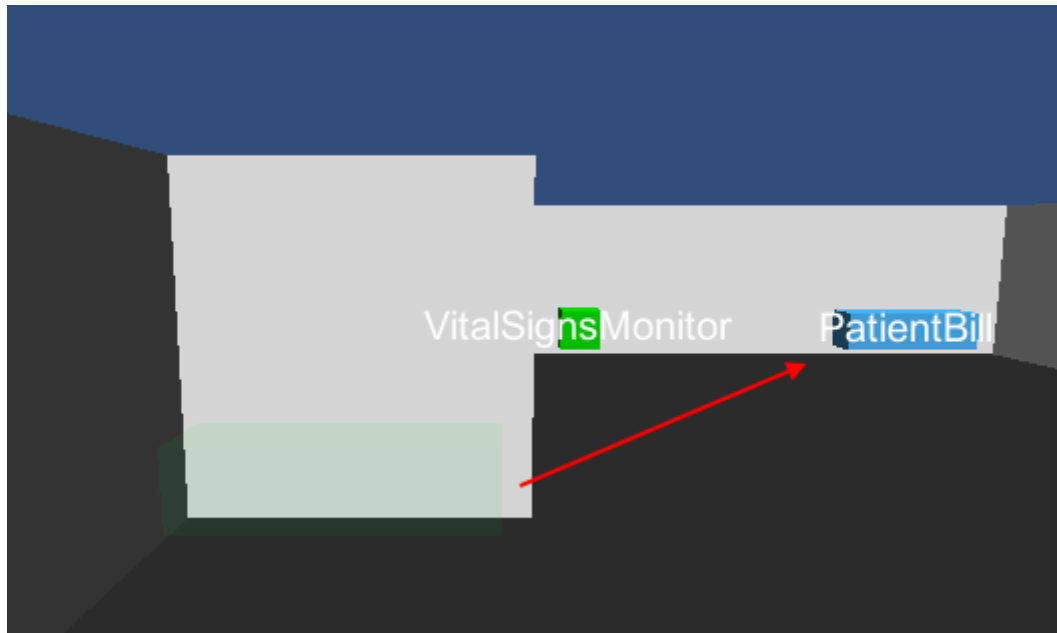


Figure 43. The *GameObject* for the asset *PatientBill* has moved to its new position.

4.2.6 Completing the Remaining Work Items

To avoid unnecessary repetition of the processes involved in completing work items, they will be summarised unless significant.

After completing the first work item the user moves into the Examination Room (see Figure 44). The assets in this room are:

- PatientBill
- VitalSignsMonitor
- ExamReport

The work items associated with the assets in this room are:

1. Check_Breathing
2. Auditory_Inspection
3. Check_Pulse
4. Check_Blood_Pressure
5. Update_Examination_Report
6. Doctor_to_Nurse_Hand_Over

Completing the last work item moves the asset *PatientBill* to the Intensive Care room (see Figure 45).

4 Execution

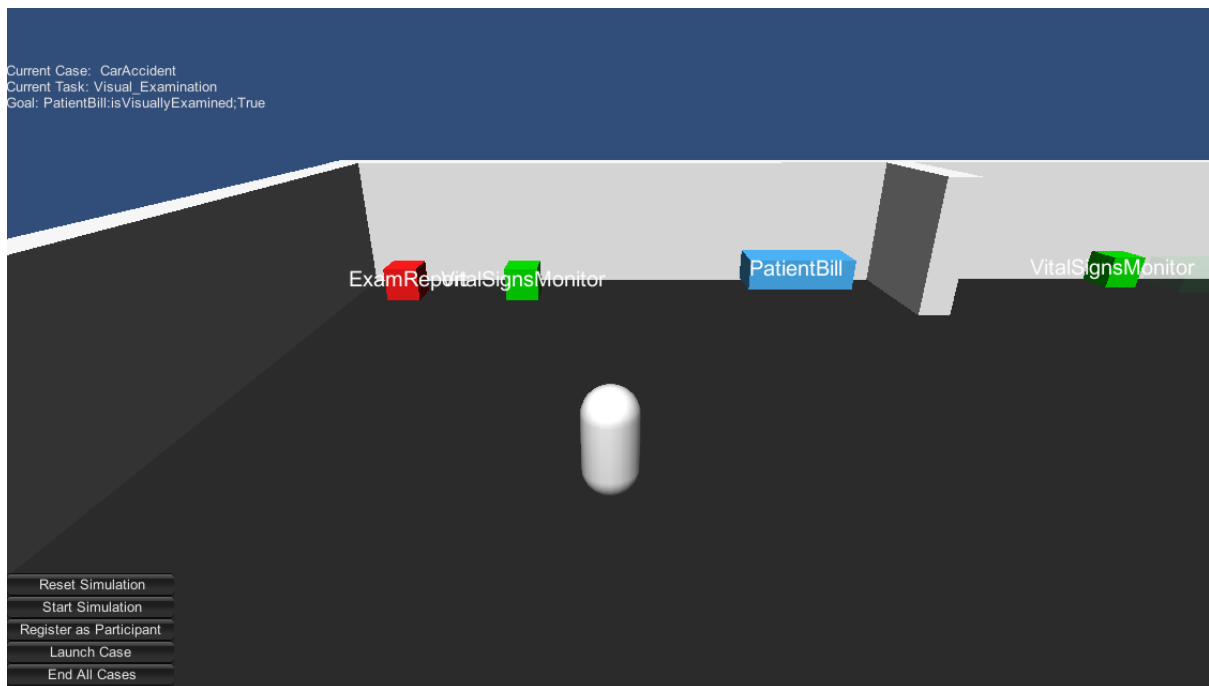


Figure 44. The user has moved their avatar to the Examination Room.

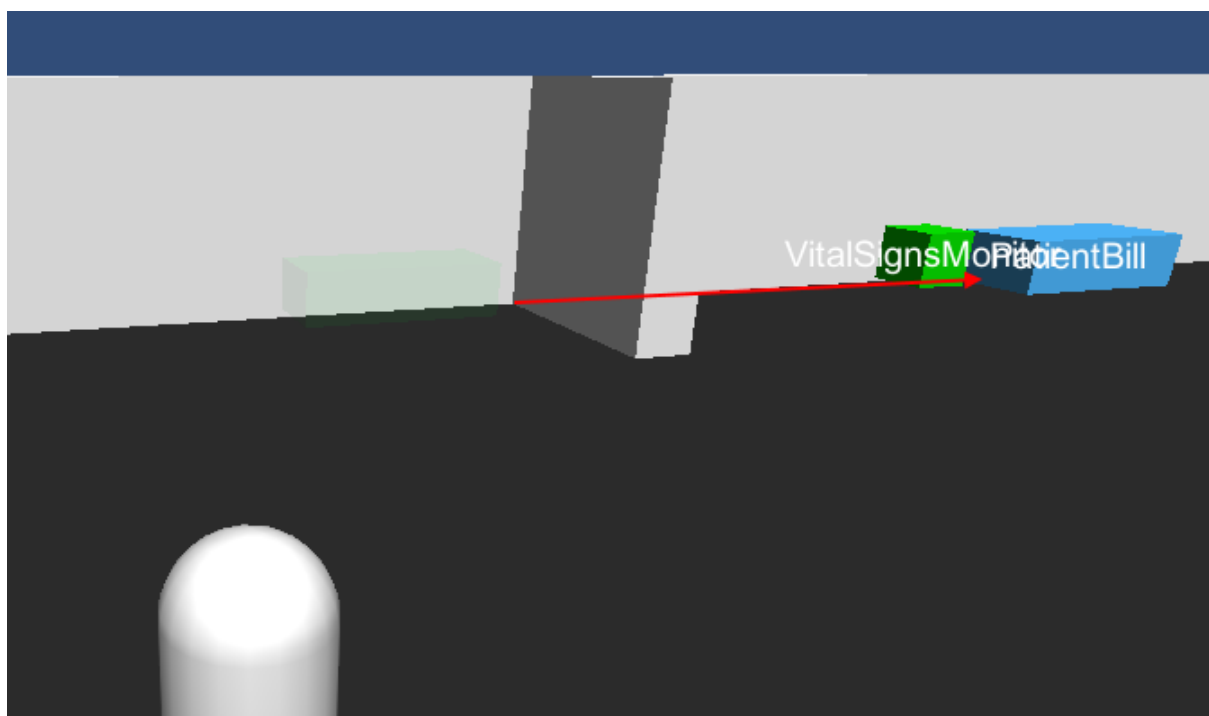


Figure 45. The asset PatientBill has moved from the Examination Room to the Intensive Care room.

The user moves into the Intensive Care room (see Figure 46). The assets in this room are:

- PatientBill
- VitalSignsMonitor
- BloodSampleVials
- RequestXRay
- RequestPathology

4 Execution

The work items associated with the assets in this room are:

1. Request_X-Ray
2. Request_Pathology
3. Take_Blood_Samples
4. Attach_Vital_Signs_Monitor
5. Admit_Patient_to_Intensive_Care
6. Collect_X-Ray_Requests
7. Collect_Patient_from_Intensive_Care

Completing the last work item moves the asset PatientBill to the Radiology Room (see Figure 47).

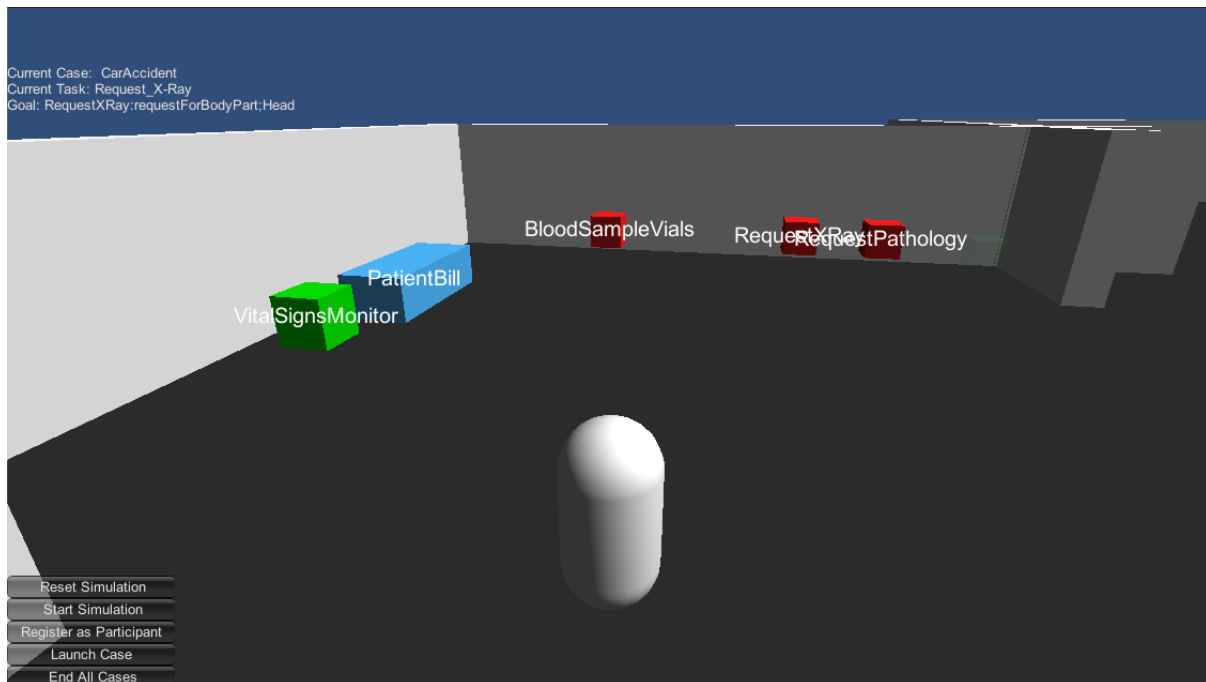


Figure 46. The user has moved their avatar to the Intensive Care room.

4 Execution

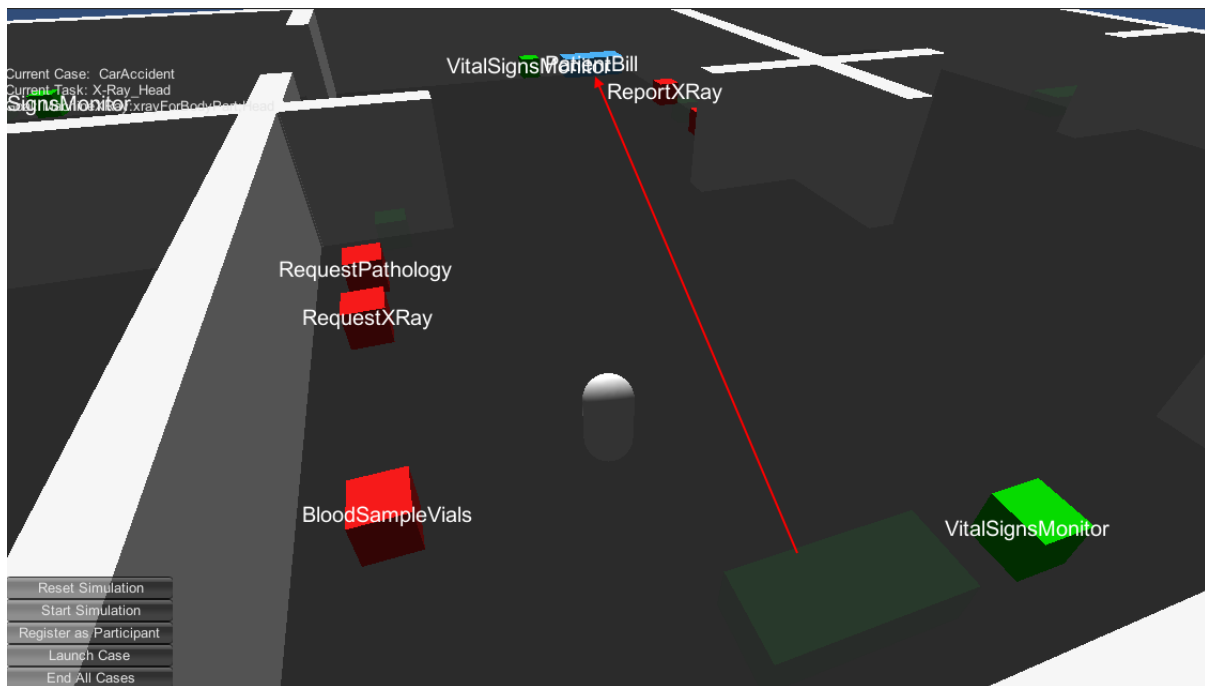


Figure 47. The asset PatientBill has moved from the Intensive Care room to the Radiology Room

The user moves into the Radiology Room (see Figure 48). The assets in this room are:

- PatientBill
- VitalSignsMonitor
- ReportXRay
- MachineXRay

The work items associated with the assets in this room are:

1. X-Ray_Head
2. X-Ray_Torso
3. Report_X-Ray_Results_to_Doctor
4. Complete_XRay_Duties

Completing the last work item moves the asset PatientBill to ICUBed02 (see Figure 49).

4 Execution



Figure 48. The user has moved their avatar to the Radiology Room.

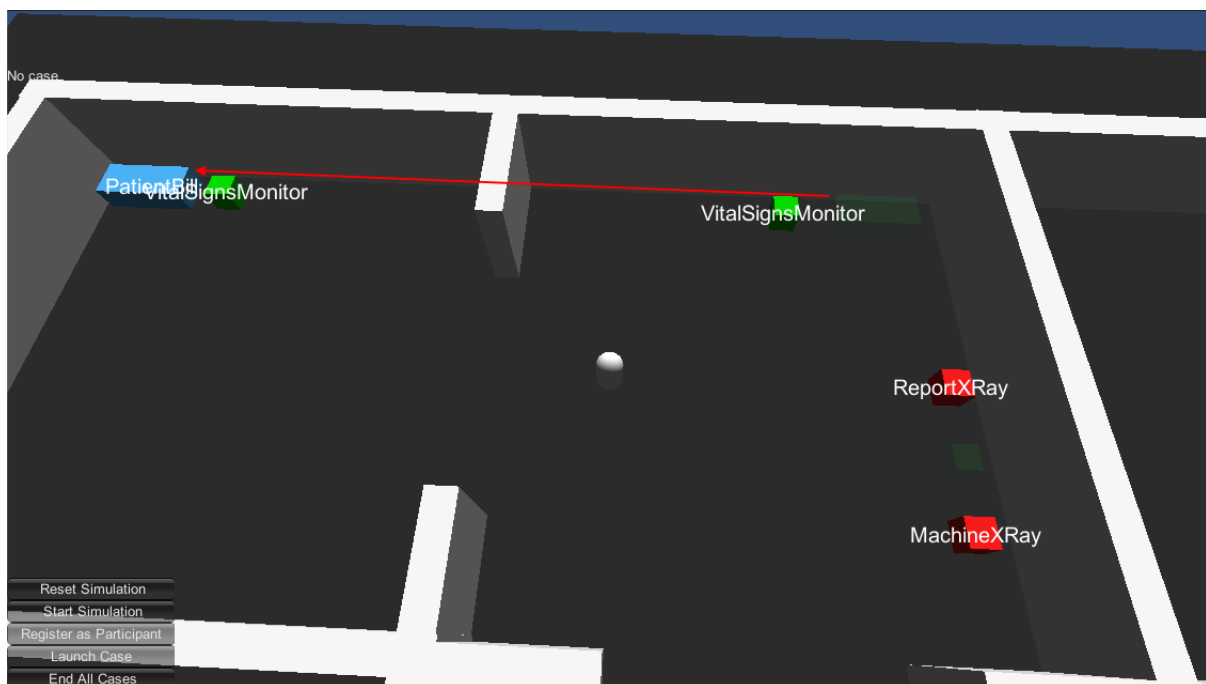


Figure 49. The asset PatientBill has moved from the Radiology Room to ICUBed02.

4.2.7 Completing the Case

Moving the asset PatientBill to ICUBed02 is the final work item for the case, so the case is now completed. The Simulation client interface updates and no longer shows a case (see Figure 50). The YAWL Web Service confirms that there are no longer any work items (see Figure 51) and there is no

4 Execution

case running (see Figure 52). The Veis Java Socket Server indicates that the case has been completed (see Figure 53).



Figure 50. The Simulation client interface updates to show that there is no longer a case running.

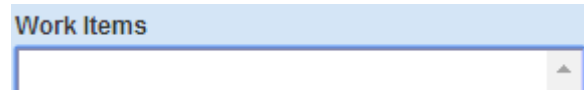


Figure 51. The YAWL Web Interface shows that there are no remaining work items.

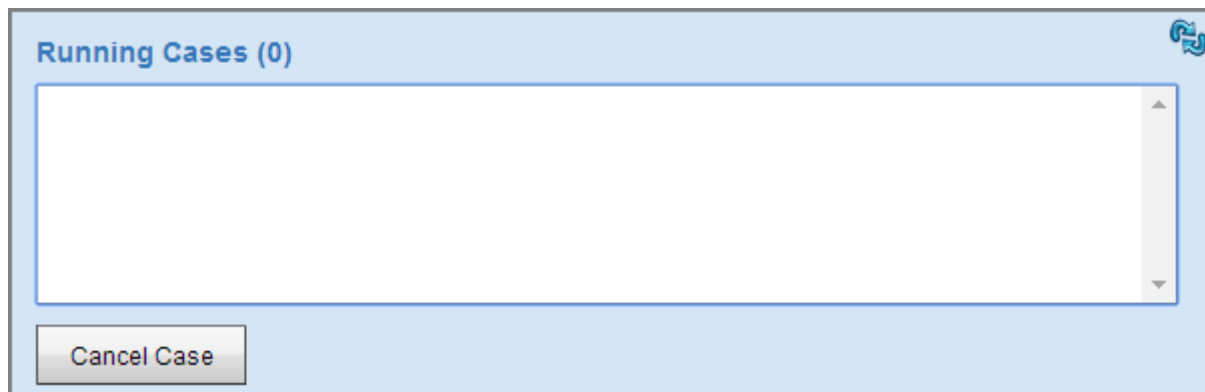


Figure 52. The YAWL Web Interface shows that there is no case running.

```
YAWL server just say something to me...
CompleteWorkItem: <success/>

[====YAWLListenerIX BEGIN RECEIVING=====]
workItem=<workItem><taskId>Complete_XRay_Duties</taskId><caseid>176.14.6</caseid>
<uniqueid>00000000000000000000000000000000x</uniqueid><taskname>Complete XRay Duties</t
askname><documentation/><specidentifier>UID_142f2e5a-7c7c-4d2e-a684-02c230e3689d
</specidentifier><specversion>0.52</specversion><specuri>CarAccident</specuri><s
tatus>Complete</status><allowsdynamiccreation>false</allowsdynamiccreation><requ
iresmanualresourcing>true</requiresmanualresourcing><codelet/><data><Complete_XR
ay_Duties /></data><enablementTime>Jun:09, 2014 18:57:38</enablementTime><enable
mentTimeMs>1402304258577</enablementTimeMs><firingTime>Jun:09, 2014 18:57:38</fi
ringTime><firingTimeMs>1402304258674</firingTimeMs><startTime>Jun:09, 2014 18:57
:38</startTime><startTimeMs>1402304258695</startTimeMs><startedBy>admin</started
By></workItem>&data=<?xml version="1.0" encoding="UTF-8"?>
<Complete_XRay_Duties>
  <Tasks>Complete_XRay_Duties</Tasks>
  <Goal>PatientBill:bedAt;ICUBed02</Goal>
</Complete_XRay_Duties>
&action=1&preCheck=false
[====YAWLListenerIX END RECEIVING=====]
CaseCompleted: 176
```

Figure 53. The Veis Java Socket Server shows the message being sent that the case has been completed.

4.2.8 Launching a Subsequent Case

A case may be started as long as there is no current case running. After finishing a case the user may choose to launch another one (see Figure 54). The user does not have to register as a participant again because their current registration isn't affected by case completion. The Simulation will handle the subsequent case without error, but the asset World State values are not reset, and assets that can move do not revert to their initial position (see Figure 55).

4 Execution

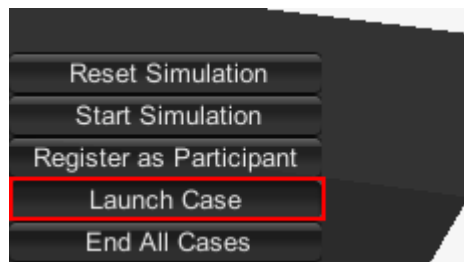


Figure 54. The user can press the Launch Case button to launch another case.

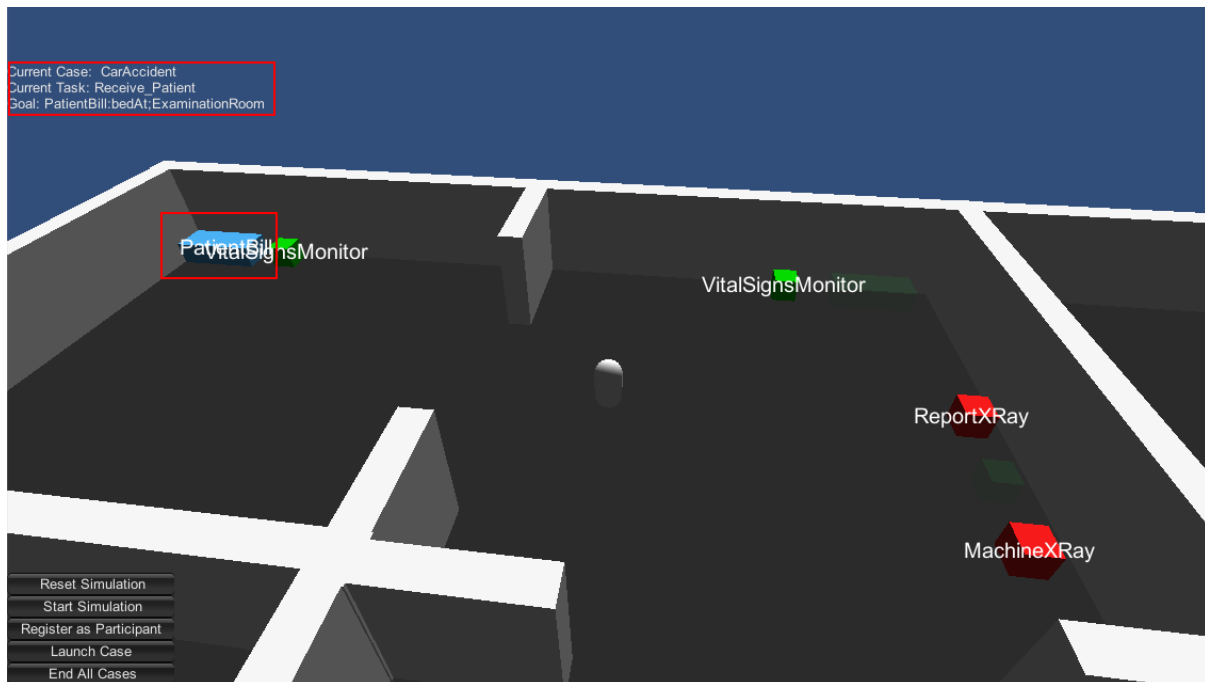


Figure 55. Launching a subsequent case works but dynamic assets will not move to their start position.

4.2.9 Cancelling a Case

A running case may be cancelled regardless of what work items have been completed (see Figure 56). The Simulation client will show that there is no case running (see Figure 57). The YAWL Web Interface confirms that the case has been cancelled (see Figure 58).

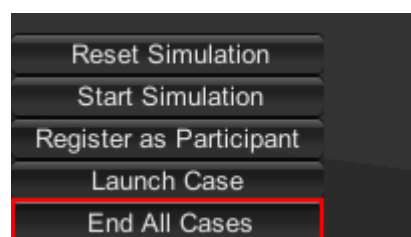


Figure 56. The user can press the End All Cases button to cancel a running case.



Figure 57. The Simulation client showing that no case is running after being cancelled.

4 Execution

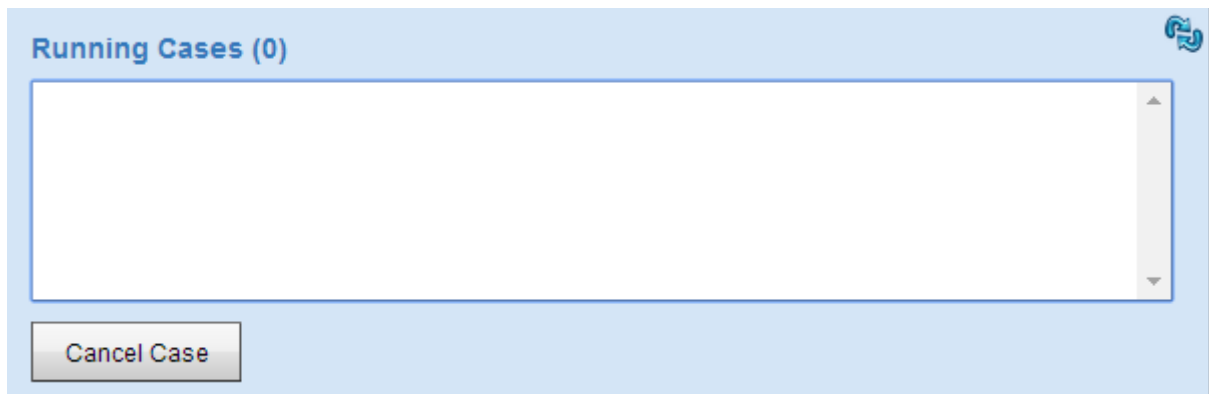


Figure 58. The YAWL Web Interface shows that there is no case running after being cancelled.

5 KNOWN ISSUES

This section lists notable technical issues and – where possible – potential solutions to those issues.

5.1 Issues Specific to this Implementation

These are issues that are present due to the project being implemented in Unity.

5.1.1 Unity DLL Conflicts

The libraries referenced by the Veis, Veis.Data, and Spark DLLs cause errors with the Unity compiler (see Figure 59 and Figure 60). The Unity compiler output file indicates that the problem arises because these DLLs reference system libraries that are not supported by Unity (see Figure 60). The errors can be subdued by manually including the specific system libraries (see Figure 62). However, the manual inclusion of system libraries can cause conflicts with their Unity version counterparts when trying to use some Types (see Figure 63). The project will compile and run if no conflicting Types are used, but that removes access to some useful Types, for example System.Timers.Timer. One possible solution to these problems which stem from mismatched system references might be to recompile the Veis DLLs using the Unity versions of the necessary system libraries.

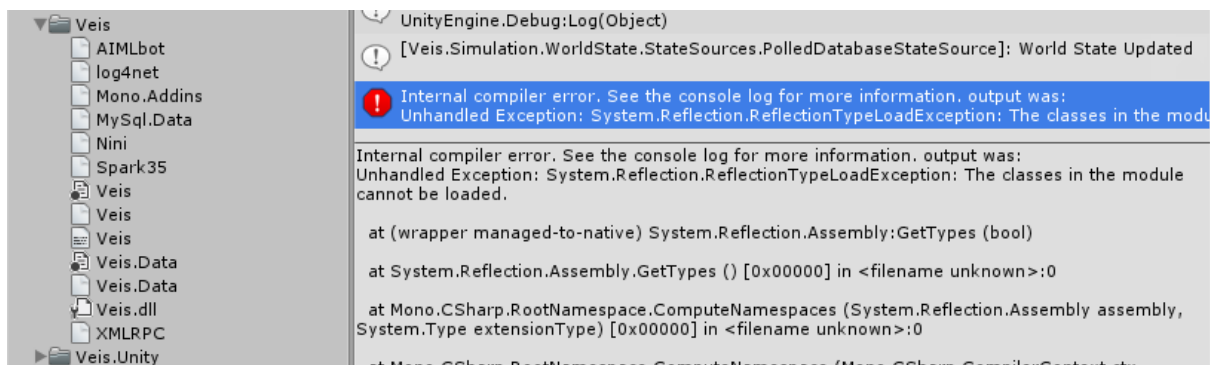


Figure 59. The Veis DLLs and their dependencies, and the error shown when Unity tries to compile them.

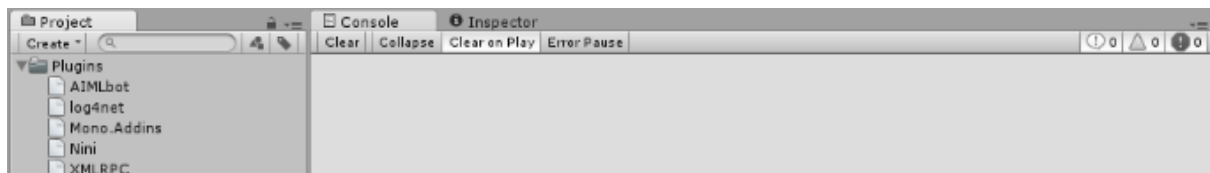


Figure 60. The Unity project will compile when the Veis, Veis.Data, and Spark DLLs have been removed.

5 KNOWN ISSUES

```
-----CompilerOutput:--stdout--exitcode: 1--compilationhadfailure: True--outfile: Temp/Assembly-CSharp.dll
The following assembly referenced from C:\Dev\inn690\Unity Project\Assets\Veis\Veis\Spark35.dll could not be
loaded:
    Assembly:   System.Web      (assemblyref_index=1)
    Version:    2.0.0.0
    Public Key: b03f5f7f11d50a3a
The assembly was not found in the Global Assembly Cache, a path listed in the MONO_PATH environment variable,
or in the location of the executing assembly (C:\Dev\inn690\Unity Project\Assets\Veis\Veis\).

Could not load file or assembly 'System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies.
Could not load file or assembly 'System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies.
The class System.CodeDom.Compiler.CompilerResults could not be loaded, used in System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089
The following assembly referenced from C:\Dev\inn690\Unity Project\Assets\Veis\Veis\Spark35.dll could not be
loaded:
    Assembly:   System.Configuration  (assemblyref_index=2)
    Version:    2.0.0.0
    Public Key: b03f5f7f11d50a3a
The assembly was not found in the Global Assembly Cache, a path listed in the MONO_PATH environment variable,
or in the location of the executing assembly (C:\Dev\inn690\Unity Project\Assets\Veis\Veis\).
```

Figure 61. The Unity Editor log indicates that some of the Veis DLLs reference system libraries not supported by Unity.

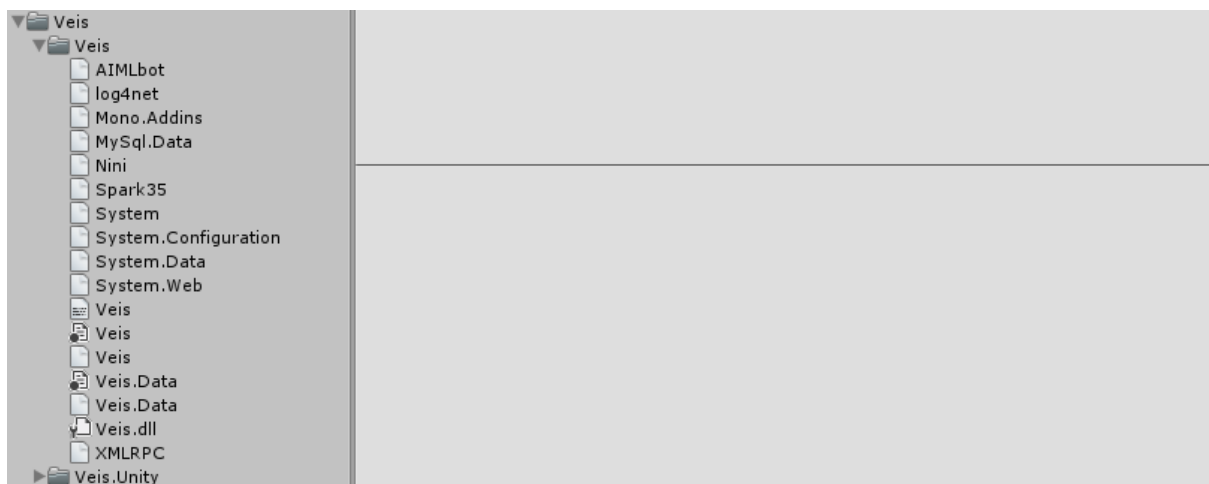


Figure 62. The Unity project will compile without error if the error-inducing system references are manually included.

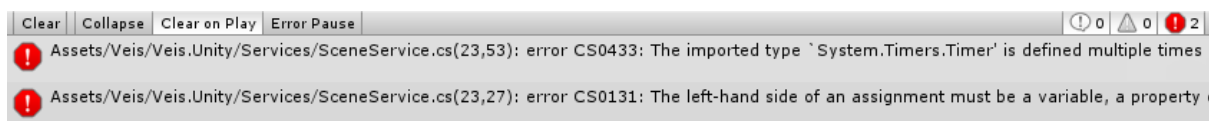


Figure 63. The Unity Editor will not compile when a Type is used that is contained in a system library that has both a Unity and non-Unity version present.

5.1.2 Unity Type Threading Issues

Unity reference Types such as Transform are not thread-safe and can only be accessed from the main application thread. This has already caused issues with some of Veis' architecture due to its use of threads (see Figure 64. Unity Types and methods cannot be accessed from outside of the main thread.); specifically, the Simulation updates the game world via the SceneService on a System.Timers.Timer, which raises events and is handled on a Thread Pool thread (not the main thread) [Microsoft 2014]. When the timer thread attempts to interact with Unity objects it breaks.

The problem has been overcome by using a very simple Producer-Consumer pattern. The timer thread queues instructions in the SceneService, and the main thread periodically checks and executes those instructions. Unfortunately as timers cannot be used due to their threaded nature

5 KNOWN ISSUES

the only way to get the main thread specifically to perform the actions is to have the call come from an Update method in the Unity scene. While the application does execute properly the tight coupling with the Unity scene is undesirable, and it is unknown how effective the Producer-Consumer pattern for handling Unity Type interaction will be as the project continues to grow.

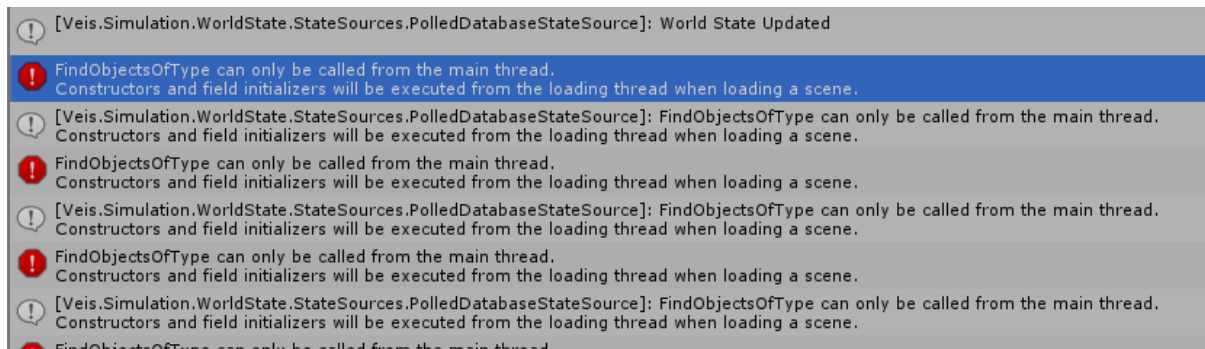


Figure 64. Unity Types and methods cannot be accessed from outside of the main thread.

5.2 Issues Not Specific to this Implementation

These are issues that were either present prior to this implementation or have not presented themselves due to the project being implemented in Unity.

5.2.1 World State Reset Not Automated

The methods for starting and resetting the Simulation only affect the Simulation properties such as the registered participant and tracked cases; these methods do not alter the World State database in any way. This means that there is no convenient way to set the World State database to its initial state; it must be done manually by reloading the World State database in PHPMysqlAdmin. The current situation is inefficient, and resetting the database automatically when the case begins or via the client interface should not be very difficult.

5.2.2 Veis Database Missing Some Asset Methods

Many assets have methods that can be enacted upon them (see Figure 65). Some of these methods are intended to move assets between locations. When an asset is to be moved it is communicated to the Simulation via a service call (see Figure 66). The possible locations for moving assets are defined in another table (see Figure 67). However the only asset relevant to the simulation that has a service call is PatientBill. The assets MachineXRay and RequestXRay both have move methods but are lacking service calls and location values. It is unclear whether these assets have not had these values entered yet, or whether they were actually not intended to be moved. At the moment the solution has been to place these assets wherever they are needed.

5 KNOWN ISSUES

		asset_name	method_name	TIMESTAMP
<input type="checkbox"/>	 Edit  Copy  Delete	BloodSampleVials	Add Blood	2013-09-13 02:34:49
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeeCup	Fill Coffee	2013-11-20 14:33:05
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeeCup	Move Cup	2013-12-05 16:25:42
<input type="checkbox"/>	 Edit  Copy  Delete	ExamReport	Add Report	2013-09-13 02:29:50
<input type="checkbox"/>	 Edit  Copy  Delete	GROComputer	Log Order	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	GROComputer	Process Order	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	GROComputer	Reset Order	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	MachineXRay	Execute XRay	2013-10-02 18:07:33
<input type="checkbox"/>	 Edit  Copy  Delete	MachineXRay	Move XRay Machine	2013-09-20 13:14:16
<input type="checkbox"/>	 Edit  Copy  Delete	PatientBill	Examine Visual	2013-09-06 11:23:08
<input type="checkbox"/>	 Edit  Copy  Delete	PatientBill	Move Bed	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	PatientBill	Test Hearing	2013-09-13 12:09:48
<input type="checkbox"/>	 Edit  Copy  Delete	ReportXRay	Add XRay Report	2013-10-02 18:32:04
<input type="checkbox"/>	 Edit  Copy  Delete	RequestPathology	Make Request	2013-10-02 13:58:54
<input type="checkbox"/>	 Edit  Copy  Delete	RequestXRay	Approve XRay Request	2013-10-02 16:35:50
<input type="checkbox"/>	 Edit  Copy  Delete	RequestXRay	Make XRay Request	2013-10-02 15:55:00
<input type="checkbox"/>	 Edit  Copy  Delete	RequestXRay	Move Request	2013-09-13 13:22:01
<input type="checkbox"/>	 Edit  Copy  Delete	Truck	Move Truck	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	Truck	Unload or Load goods	2013-09-01 13:07:39
<input type="checkbox"/>	 Edit  Copy  Delete	VitalSignsMonitor	Move Blood Pressure Band	2013-09-06 12:30:52
<input type="checkbox"/>	 Edit  Copy  Delete	VitalSignsMonitor	Move Mouthpiece	2013-09-06 12:30:52
<input type="checkbox"/>	 Edit  Copy  Delete	VitalSignsMonitor	Move Pulse Sensor	2013-09-06 12:31:01

Figure 65. Methods that can be enacted upon assets.
















		method_name	service_call	variable	value	TIMESTAMP
<input type="checkbox"/>	 Edit  Copy  Delete	Move Bed	Move	Bed to		2013-09-01 13:08:47
<input type="checkbox"/>	 Edit  Copy  Delete	Move Cup	Move	Cup to		2013-12-05 16:38:00
<input type="checkbox"/>	 Edit  Copy  Delete	Move Forklift	Move	Forklift to		2013-09-01 13:08:47
<input type="checkbox"/>	 Edit  Copy  Delete	Move Truck	Move	Truck to		2013-09-01 13:08:47
<input type="checkbox"/>	 Edit  Copy  Delete	Unload or Load goods	Move Goods	Truck from		2013-09-01 13:08:47

Figure 66. Services to be called when an asset method is to be enacted.




























































































		name	value	TIMESTAMP
<input type="checkbox"/>	 Edit  Copy  Delete	Actor	Doctor	2013-09-13 02:28:53
<input type="checkbox"/>	 Edit  Copy  Delete	Actor	None	2013-10-02 17:07:10
<input type="checkbox"/>	 Edit  Copy  Delete	Actor	Nurse	2013-09-13 02:28:53
<input type="checkbox"/>	 Edit  Copy  Delete	Actor	XRyTechnician	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	EmergencyRoom	2013-10-11 10:41:01
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	ExaminationRoom	2013-10-11 10:35:39
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	ICUBed02	2013-12-03 18:12:45
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	IntensiveCare	2013-09-13 12:07:26
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	Machine	2013-12-05 15:25:59
<input type="checkbox"/>	 Edit  Copy  Delete	bed_locations	RadiologyRoom	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Boolean	False	2013-09-06 11:54:27
<input type="checkbox"/>	 Edit  Copy  Delete	Boolean	True	2013-09-06 11:54:21
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeeCupState	Coffee	2013-11-20 14:33:43
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeeCupState	Empty	2013-11-20 14:33:43
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeePosition	ExaminationRoom	2013-12-05 15:41:19
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeePosition	Machine	2013-11-20 14:20:07
<input type="checkbox"/>	 Edit  Copy  Delete	CoffeePosition	Table	2013-11-20 14:20:07
<input type="checkbox"/>	 Edit  Copy  Delete	loading_status	Loaded	2013-09-01 13:07:57
<input type="checkbox"/>	 Edit  Copy  Delete	loading_status	Unloaded	2013-09-01 13:07:57
<input type="checkbox"/>	 Edit  Copy  Delete	Patient	PatientBill	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Patient	PatientChest	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Patient	PatientHead	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_list	BillGate	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_list	None	2013-10-02 14:07:22
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_list	SteveJob	2013-10-11 10:47:30
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_part	Head	2013-10-02 15:42:23
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_part	None	2013-10-02 15:42:12
<input type="checkbox"/>	 Edit  Copy  Delete	Patient_part	Torso	2013-10-02 15:42:12
<input type="checkbox"/>	 Edit  Copy  Delete	patient_state	False	2013-09-06 11:19:11
<input type="checkbox"/>	 Edit  Copy  Delete	patient_state	True	2013-09-06 11:19:11

Figure 67. Possible values that asset predicate labels can be assigned.

5.2.3 Simulation Client Connection with Veis Java Socket Server Unreliable

During case run-throughs the Veis Java Socket Server would occasionally lose the client connection (see Figure 68, Figure 69, and Figure 70). The problem appears to occur in the MessageHandler class which handles communications with the client; coincidentally, the disconnections seemed to occur when the Simulation would complete numerous goals in quick succession (which can happen when work item goal conditions are still satisfied from being completed in a previous case).

5 KNOWN ISSUES

A potential diagnosis could be that there are synchronisation issues in the Socket Server when sending and receiving work items quickly. Perhaps the message handling process is lengthy and gets interrupted by messages from the client. The `ConcurrentModificationException` might suggest that some values are being accessed simultaneously – perhaps across threads – which also suggests that the Socket Server isn't handling work item communication in a reliable manner.

```
[AgentCommunication] Agent Visualisation disconnected unexpectedly
java.util.ConcurrentModificationException
    at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:859)
    at java.util.ArrayList$Itr.next(ArrayList.java:831)
    at veis.system.MessageHandler.GetTaskQueue(MessageHandler.java:274)
    at veis.system.MessageHandler.run(MessageHandler.java:75)
A client has been disconnected. Current connections: 0
```

Figure 68. Veis Java Socket Server losing its client connection unexpectedly.

```
[YAWLListenerIX]:Server returned HTTP response code: 500 for URL: http://localhost:8080/resourceService/workqueuegateway
Server closed.
A client connected on port 4444 Current connections: 1
[AgentCommunication] Agent Visualisation disconnected unexpectedly
java.io.IOException: Server returned HTTP response code: 500 for URL: http://localhost:8080/resourceService/workqueuegateway
    at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1626)
    at org.yawlfoundation.yawl.engine.interface.Interface_Client.send(Interface_Client.java:150)
    at org.yawlfoundation.yawl.engine.interface.Interface_Client.executeGet(Interface_Client.java:71)
    at org.yawlfoundation.yawl.resourcing.rsInterface.WorkQueueGatewayClient.performGet(WorkQueueGatewayClient.java:95)
    at org.yawlfoundation.yawl.resourcing.rsInterface.WorkQueueGatewayClient.getQueuedWorkItems(WorkQueueGatewayClient.java:280)
    at veis.yawl.YAWLComm.ParseQueuedWorkQueue(YAWLComm.java:221)
    at veis.entities.workflow.Agent.UpdateQueues(Agent.java:126)
    at veis.yawl.YAWLComm.ProcessWorkItems(YAWLComm.java:283)
    at veis.system.MessageHandler.GetTaskQueue(MessageHandler.java:270)
    at veis.system.MessageHandler.run(MessageHandler.java:75)
A client has been disconnected. Current connections: 0
```

Figure 69. Veis Java Socket Server losing its client connection unexpectedly.

```
CompleteWorkItem: <success/>
[AgentCommunication] Agent Visualisation disconnected unexpectedly
java.util.ConcurrentModificationException
    at java.util.HashMap$HashIterator.nextEntry(HashMap.java:926)
    at java.util.HashMap$EntryIterator.next(HashMap.java:966)
    at java.util.HashMap$EntryIterator.next(HashMap.java:964)
    at java.util.HashMap.putAll(HashMap.java:646)
    at org.yawlfoundation.yawl.resourcing.rsInterface.WorkQueueGatewayClient.performGet(WorkQueueGatewayClient.java:94)
    at org.yawlfoundation.yawl.resourcing.rsInterface.WorkQueueGatewayClient.getQueuedWorkItems(WorkQueueGatewayClient.java:280)
    at veis.yawl.YAWLComm.ParseQueuedWorkQueue(YAWLComm.java:221)
    at veis.entities.workflow.Agent.UpdateQueues(Agent.java:127)
    at veis.yawl.YAWLComm.ProcessWorkItems(YAWLComm.java:283)
    at veis.system.MessageHandler.WorkItemAction(MessageHandler.java:296)
    at veis.system.MessageHandler.run(MessageHandler.java:77)
A client has been disconnected. Current connections: 0
YAWL server just say something to me...
```

Figure 70. Veis Java Socket Server losing its client connection unexpectedly.

5.2.4 Veis Java Socket Server Unnecessarily Computationally Intensive

The computer this implementation was developed on has an idle CPU core temperature of ~55°C (see Figure 71). This temperature is maintained while running the necessary web server and YAWL application. However, after running the Veis Java Socket Server for a minute or two the core temperatures rise quite dramatically. When a client connects the core temperatures rise to dangerously high levels of ~100 °C (see Figure 72).

5 KNOWN ISSUES

The exact cause of the extreme core temperature increase is unknown. According to the source code the Socket Server application runs a thread for listening to YAWL and a thread for each client connection. There are some while loops that may be the cause (see Figure 73 and Figure 74). Some brief investigation suggests that these could be a form of “Busy Wait” which is a computationally intensive method of doing nothing while waiting for conditions. Although it doesn’t directly hinder development there is a clear unnecessary strain on host machines that might have a relatively simple solution such as a CyclicBarrier.

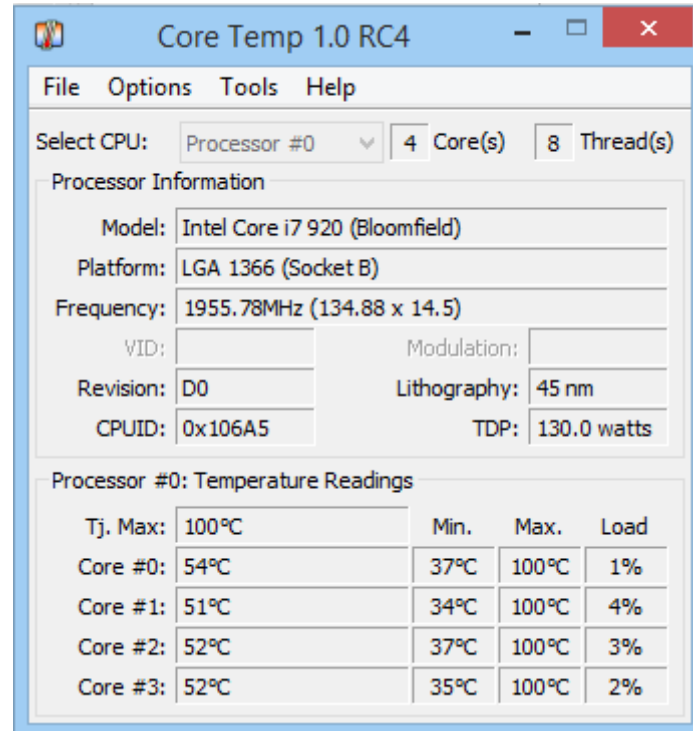


Figure 71. Host CPU core temperatures while not running Veis Java Socket Server.

5 KNOWN ISSUES

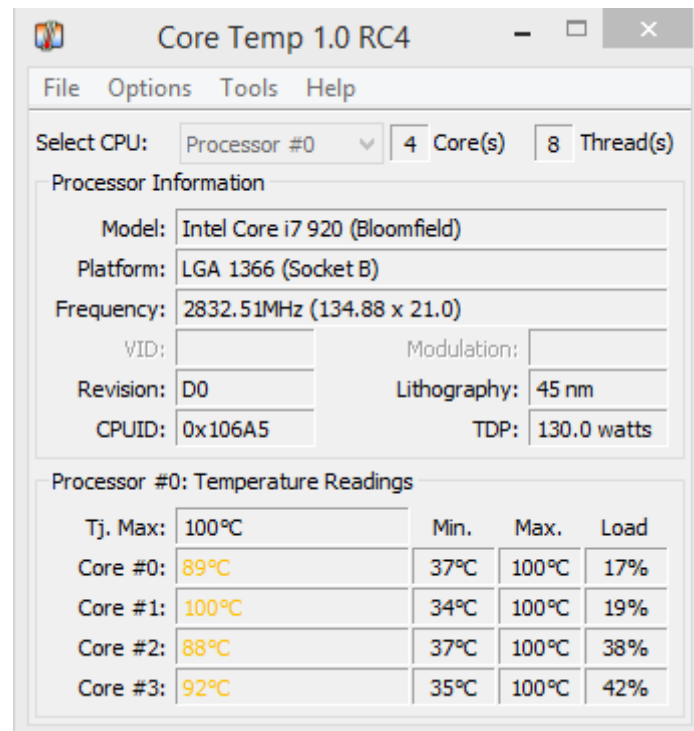


Figure 72. Host CPU core temperatures while running Veis Java Socket Server.

```
// Reads messages received from the visualization client, until an error
// occurs, or it receives
// a message telling it to stop listening (EndSession OR EndSimulation)
while(!clientSocket.isInputShutdown() && !clientSocket.isOutputShutdown()
|| !clientSocket.isClosed() && clientSocket.isConnected()) {
    if(in.ready()) {
        String inputLine = in.readLine();
```

Figure 73. Potential "Busy Wait" while loop in Veis Java Socket Server class YAWLSocketServerIX.

```
//Wait 1 second or until client sent something
while(!in.ready() && Calendar.getInstance().getTimeInMillis()-1000 < time) {
    //System.out.println("STH");
    // Do nothing
}
```

Figure 74. Potential "Busy Wait" while loop in Veis Java Socket Server class MessageHandlerOn4444.

6 FUTURE DEVELOPMENT

This section outlines possible further features based on the current Unity implementation.

6.1 Web Interface Integration

The Asset Interaction Web Interface (see Figure 12) is a series of web pages that are loaded in a web browser external to the Unity application. It serves its purpose well enough, but it is not necessary for it to be separate from the application. The separation causes a slight disconnect for the user between the Unity application GUI and the external asset interface; it also creates confusion from a development perspective in that a single important feature is not integrated with the rest of the application. The portability of the web interface is useful, but it is not such a large piece of the implementation that integrating it into the Unity application should be considered a significant risk.

6.2 Networking

One of the long-term goals for the project is to have multiple simultaneous users who can collaborate in the virtual environment and on cases. Facilitating multiple simultaneous users over a network is no small task; if it is to be implemented at all it should be as early in development as possible as it dramatically affects the approaches and considerations that must be taken.

Probably the most important decision to make is whether the application uses peer-to-peer or client-server networking. This is a decision that is closely tied to the project structure including pieces outside of the Unity application. Any application that has control over the Simulation will need to be able to communicate with YAWL (through the Veis Java Socket Server) and the database. Initially it would seem that only an authoritative server-client network solution would be suitable under such conditions. It does not seem practical for peer-to-peer applications to all have access to the same networked resources.

REFERENCES

Microsoft. 2014. Timer Class. Retrieved June 11, 2014 from <http://msdn.microsoft.com/en-us/library/system.timers.timer.aspx>

APPENDIX A PROJECT SETUP INSTRUCTIONS

These setup instructions assume that the necessary project files have been downloaded from this repository: <https://github.com/alexandercrichton/inn690.git>. Ensure that the root directory contains the following directories:

- PHP_MYSQL
- Unity Project
- YAWL

Web Server

1. Download and install WampServer from <http://www.wampserver.com/en/#download-wrapper>. Default settings are fine
2. Start WampServer
3. Use CMD to check that port 80 isn't in use. Run 'netstat -a -n' and make sure TCP [::]:80 isn't visible. If it is, the server won't be able to start. Doing a search on the executable and port 80 should resolve the matter.
4. Left-click on the WampServer tray icon and *Start All Services*.
5. Open a web browser and navigate to 'localhost/phpmyadmin'; username 'root'; no password.
6. Click the *Import* tab.
7. Under *File to Import* go to *Choose File*, open the desired file, and click *Go* down the bottom. Do this for these files:
 - a. <ProjectRoot>\PHP_MYSQL\veis_knowledge_base.sql\veis_knowledge_base.sql
 - b. <ProjectRoot>\PHP_MYSQL\veis_logging.sql\veis_login.sql
 - c. <ProjectRoot>\PHP_MYSQL\veis_world_states.sql\veis_world_states.sql
8. Now move the two folders in <ProjectRoot>\PHP_MYSQL\www to your WAMP server. It's likely to be C:\wamp\www.

YAWL

1. Ensure you have a recent JDK installed (must be JDK; JRE etc will not suffice).
2. Download YAWL from <http://www.yawlfoundation.org/>.
3. Run the YAWL installer. When asked about the Java installation make sure it points to your JDK.
4. Once installed make sure you have full permissions on all files in the YAWL install directory; if not some services may not be able to start.
5. Execute <YAWLDirectory>\bin\startup.bat.
6. Open a web browser and navigate to 'localhost:8080/resourceService'; username 'admin'; password 'YAWL'.
7. Click the *Org Data* tab.
8. Click the *Import Org Data from file* button in the top right (📁). Select *Choose File*, open <ProjectRoot>\YAWL\YAWL Specifications\YAWL Org Model.ybkb, and click *Import File*.
9. Click the *Cases* tab.
10. Under *Upload Specification* click *Choose File*, open <ProjectRoot>\YAWL\YAWL Specifications\TramaCentreA.yawl, and click *Upload File*.

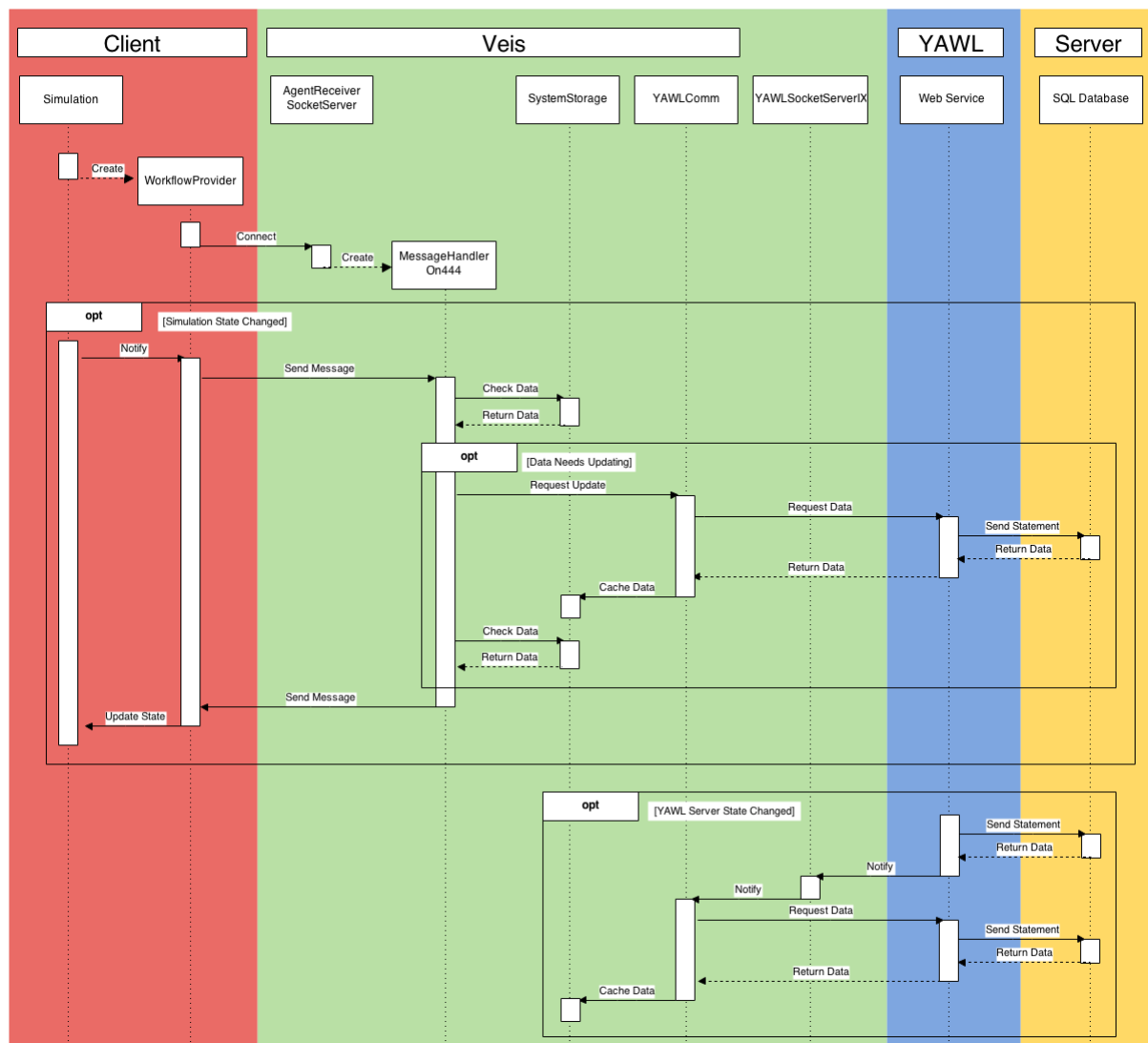
Veis Java Socket Server

1. Execute <ProjectRoot>\YAWL\veis_java_socket_server\veis\Run.bat.

Unity Application

2. Execute <ProjectRoot>\Unity Project\Hospital Simulation.exe.

APPENDIX B VEIS JAVA SOCKET SERVER COMMUNICATION HANDLING



A 1. Sequence Diagram describing the communication process between the Simulation and YAWL as handled by the Veis Java Socket Server.