Unity Hospital Workflow Simulation Client Project

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

2 Project Background

2.1 Introduction

Software for simulating hospital environment. Intended to be used for training purposes.

Participants role-play scenarios. Uses automated workflow tools to delegate and track tasks.

2.2 Technical Overview

- YAWL/MySQL backend
- Veis/Simulation
- Client (previously OpenSim)

2.3 Project Motivations

Move away from relying on buggy open source software. Move toward more dedicated, stable, effective development tools.

2.4 Comparison of Development Options

Outline effectiveness and relevance of game development tools (e.g. complete, streamlined solutions)

Pros and cons of possible tools: Unity, UDK, CryENGINE, Game Maker Studio, Source

Comparison table as appendix

3 Implementation Design

3.1 Scope

- Locally hosted
- Single user
- Single existing case
- Single participant
- Rudimentary environment
- User affects simulation via existing web interface
- Game world and simulation update together
- Assets can move as intended
- Case can be completed

3.2 Unity Simulation Implementation

3.3 Modifications to Veis Architecture

- Changed the way database info was stored and read from a DLL config to an XML file

4 Implementation Execution

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

4.1 Relevant Interface Overview

Each of the relevant interfaces that are shown are explained here.

4.1.1 Unity Simulation Client Graphical User Interface

Figure 1 shows the graphical interface of the Unity Simulation client 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. Information about any current case and work items is located in the top-left corner of the interface. Assets that can be interacted with are distinguishable by their bright blue, green, or red colour. Particular assets can be easily identified by their asset name which is rendered as floating text layered on top of each visible asset.

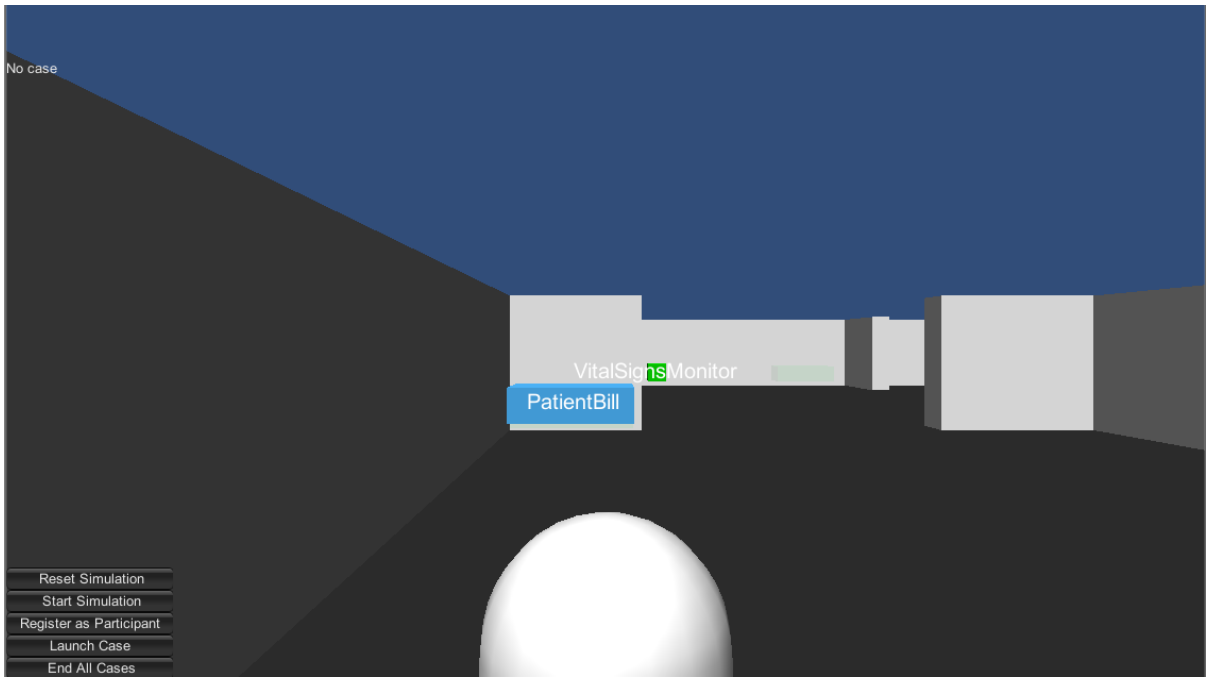

Figure 1. Application interface as seen by the user when the Unity scene starts

4.1.2 Veis Java Socket Server Console Interface

Figure 2 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.

Figure 2. Veis Java Socket Server console window immediately after opening

4.1.3 YAWL Web Interface

Figure 3 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.

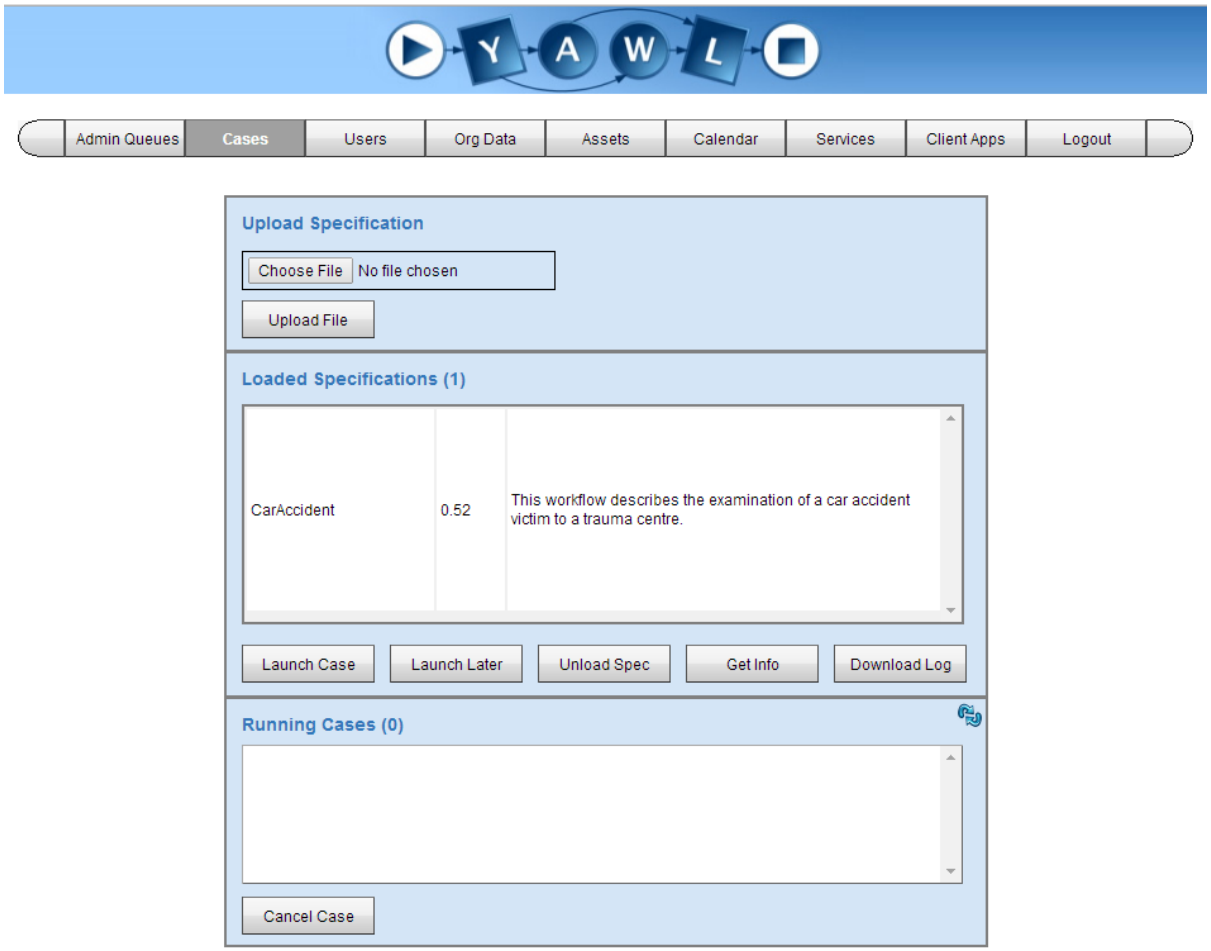

Figure 3. YAWL web interface view of case management section with no case running

Figure 4 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.

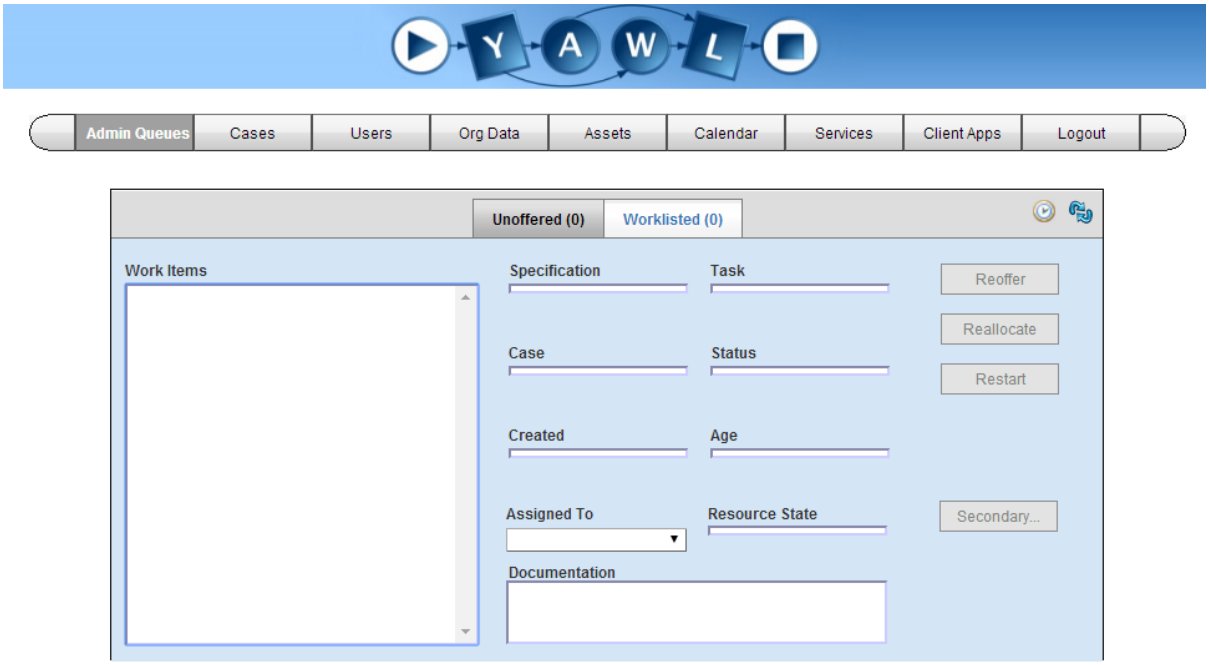

Figure 4. YAWL web interface view of current work items with no case running

4.1.4 World State Table View

Figure 5 Shows a view of the 'world_states' table, part of the Veis MySQL database, accessed through the PHPMyAdmin Web Interface. Simulation assets are defined in the Veis MySQL database. The 'world_states' table 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.

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

Figure 5. MySQL database table view of asset world states with its initial values

4.1.5 Asset Service Routine Table View

Figure 6 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.

	key	priority	asset_key	service_routine	world_key	TIMESTAMP
Edit Copy Delete	132	1	2297ca59-bfe2-49ac-9265-e4af9b35b5fb	Move:Bed to=ExaminationRoom	1	2014-06-09 23:12:42

Figure 6. MySQL database table view of asset service routines.

4.2 Unity Simulation Client Demonstration

This section demonstrates the functional aspects of the Unity Simulation Client. 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

4.2.1 Client Start

As shown in Figure 1 when the Unity Simulation Client 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 7 confirms that the Unity Simulation Client successfully establishes communications with the Veis Java Socket Server. The client is now able to send and receive messages to and from YAWL.

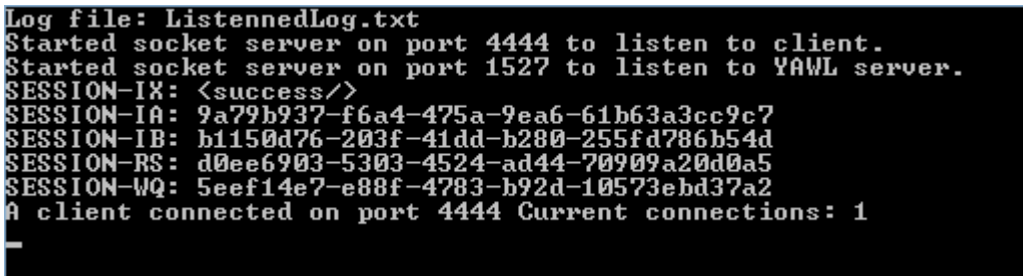

Figure 7. A Simulation client has successfully connected to the Veis Java Socket Server.

4.2.3 Registering as a Participant

Figure 8 shows the user pressing the Register as Participant button in the Unity Simulation Client interface. This process is handled internally within the Simulation. YAWL is not notified of any participants. The Simulation tracks the participants and passes work items to them. If the user does not register as a participant they will not be sent or be able to complete any case or work item.

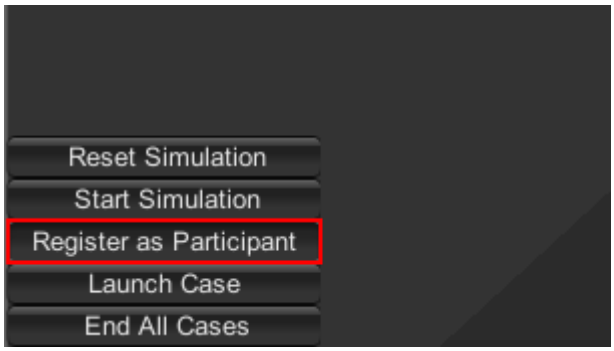

Figure 8. The user assigns a registered case participant to their avatar.

4.2.4 Launching the Case

Figure 9 shows the user pressing the Launch Case button. Figure 10 shows the client 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.

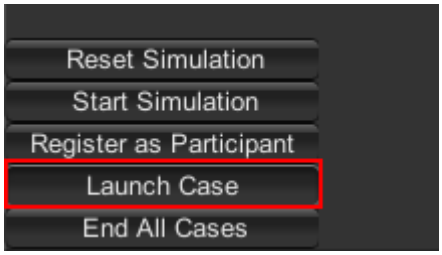

Figure 9. The user launches the case.

Figure 10. The interface updates to show the newly launched case, the user's first work item, and the work item's condition for completion.

Figure 11 confirms the corresponding case has been launched and is running in the YAWL Web Interface. Figure 12 shows the corresponding work item has been delegated. Figure 13 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.

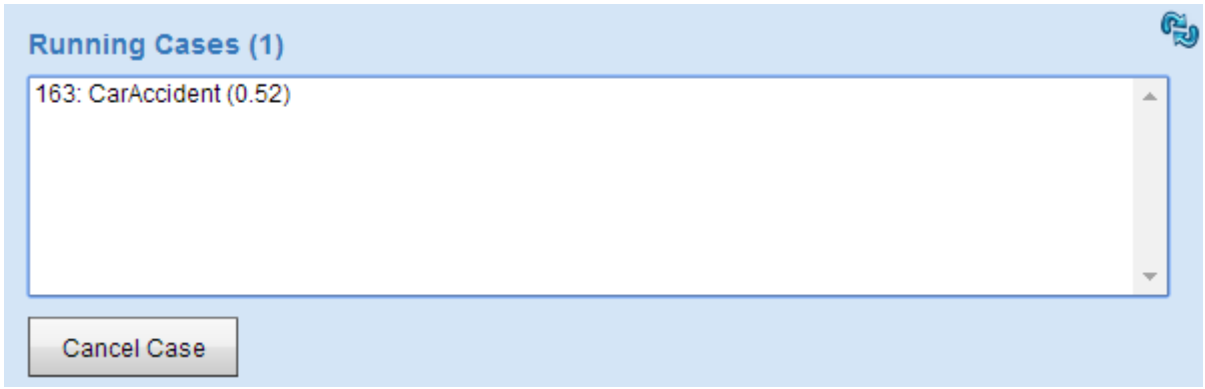

Figure 11. The YAWL Web Interface shows the case running.

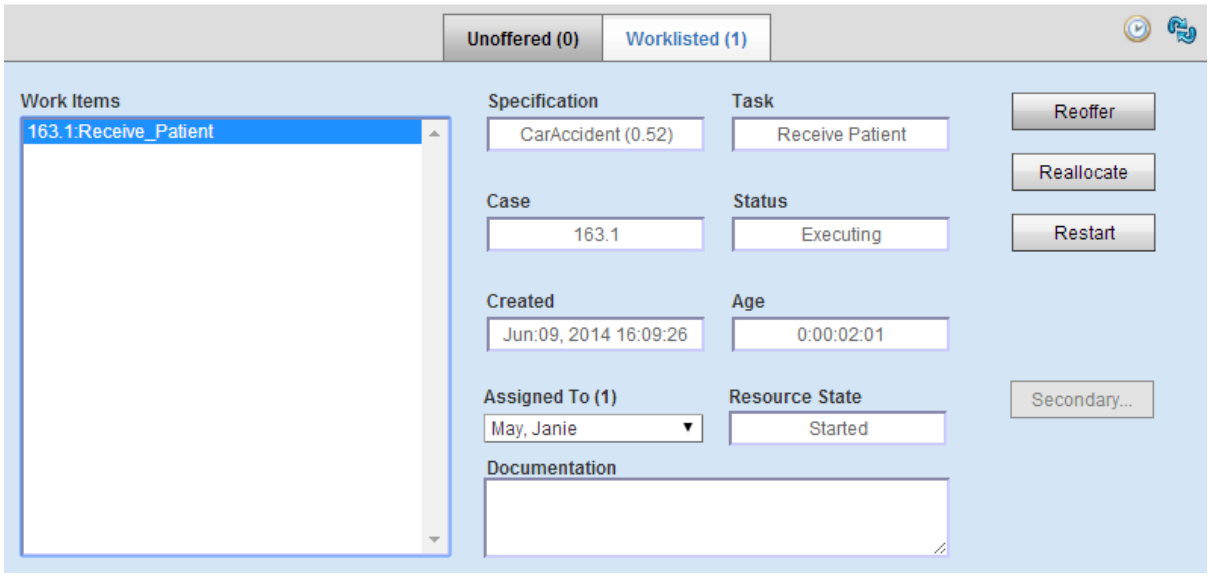

Figure 12. The YAWL Web Interface shows the work item delegated to the user.

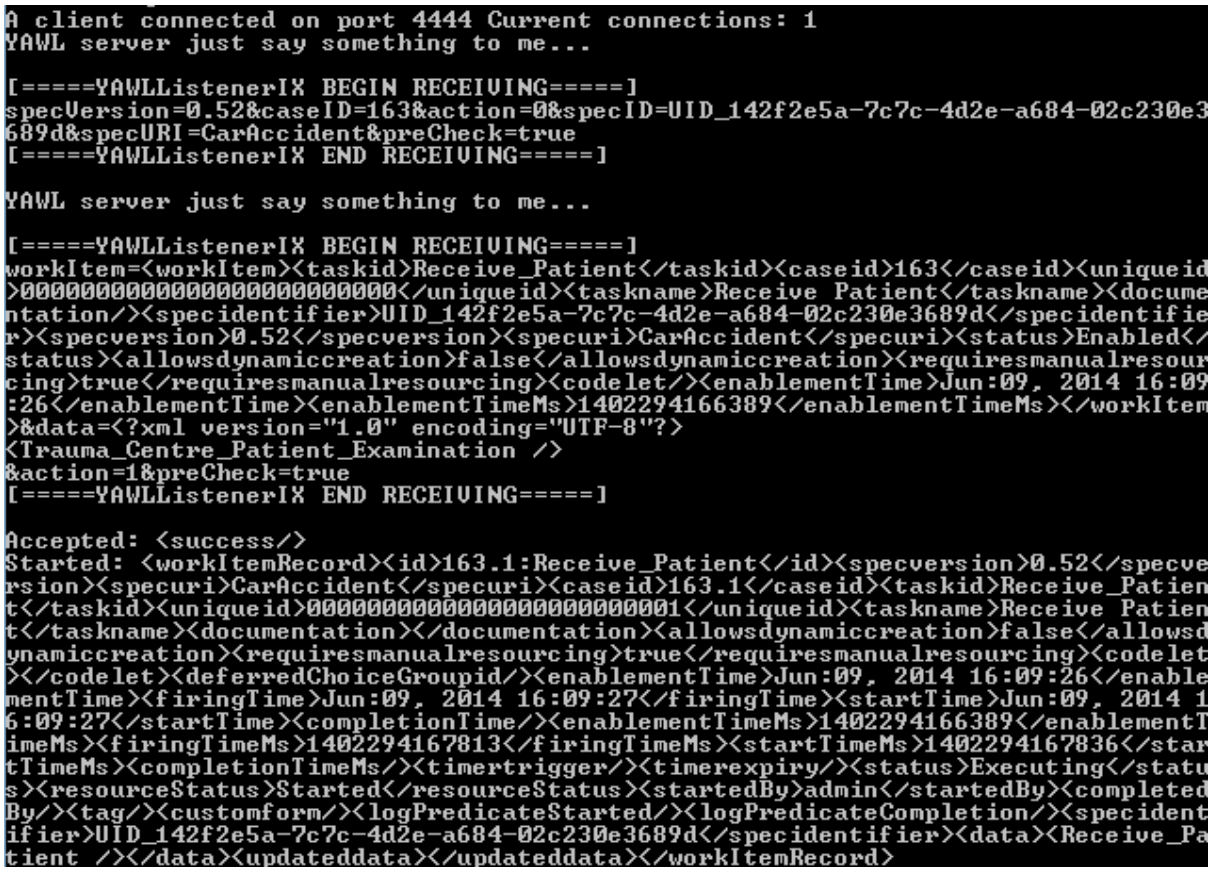

Figure 13. The Veis Java Socket Server shows the data received from YAWL and sent to the Simulation.

4.2.5 Completing the First Work Item

Figure 14 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 15 outlines the Unity GameObject that represents the asset PatientBill in the client. The user can press on the GameObject to launch the asset interaction.

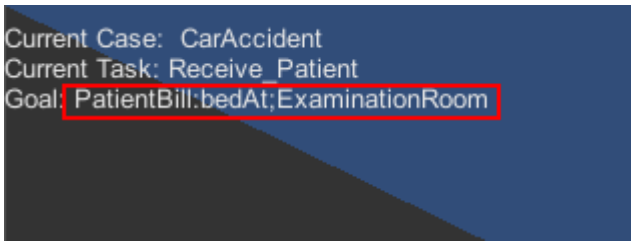

Figure 14. The client interface shows the asset state required to complete the work item

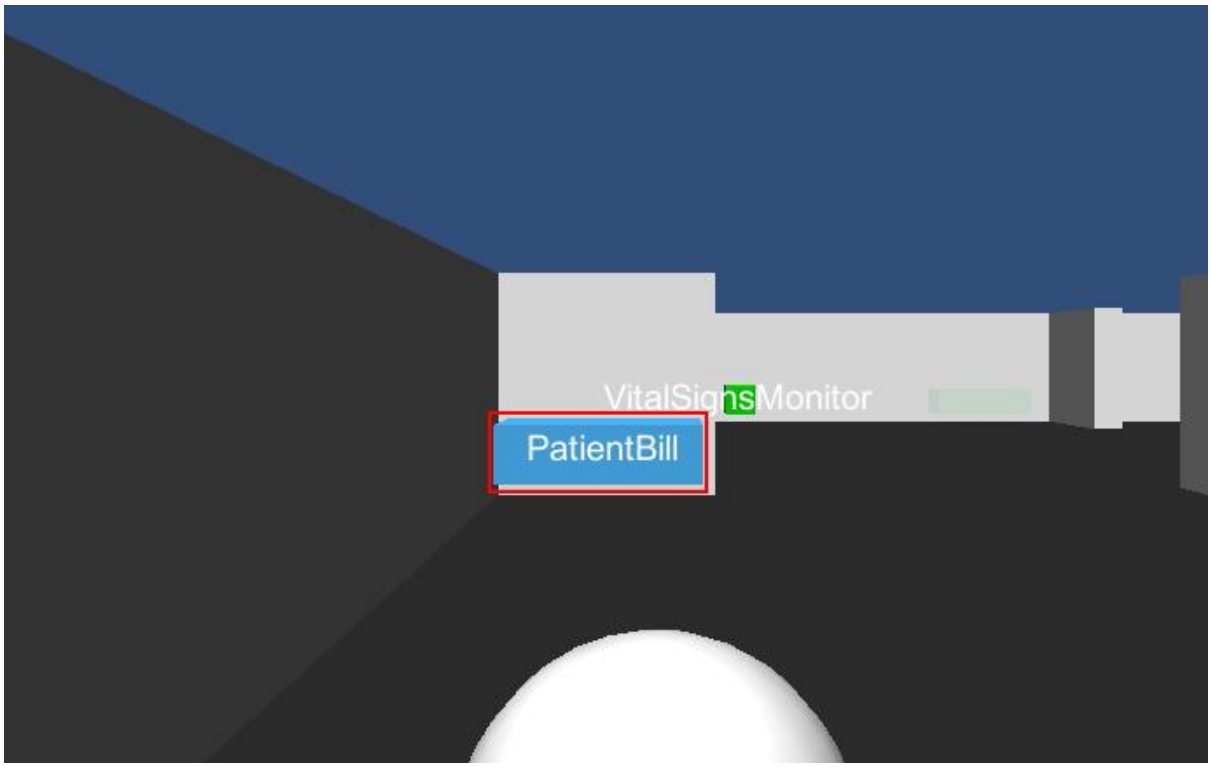

Figure 15. Outlining the asset PatientBill that is part of the first work item.

When the user presses on the PatientBill GameObject the Asset Interaction Web Interface is launched (see Figure 16). 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 17). 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 18). The user selects from the available values the one they intend to assign to the predicate label (see Figure 19). 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 20). The user is finally prompted to close the Asset Interaction Web Interface (see Figure 21).

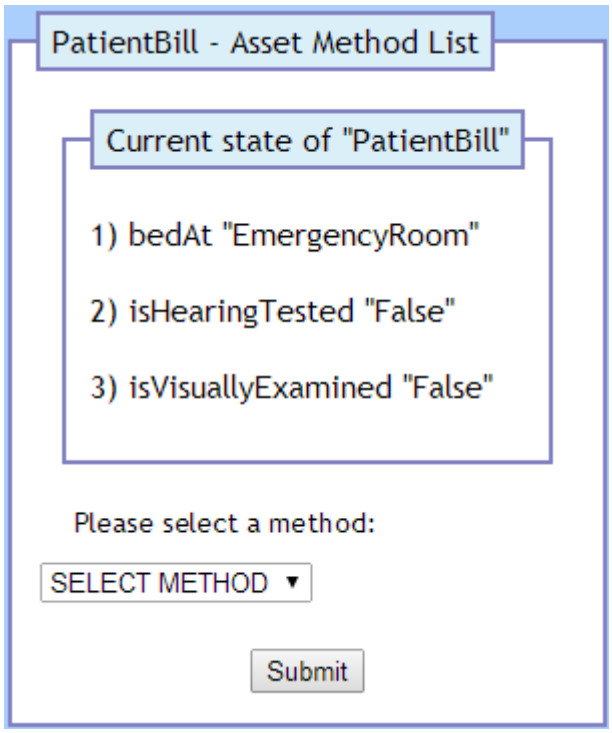

Figure 16. The Asset Interaction Web Interface when launched by pressing on the asset PatientBill.

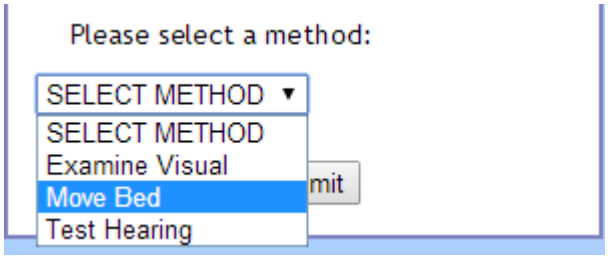

Figure 17. The available asset methods for the asset PatientBill.

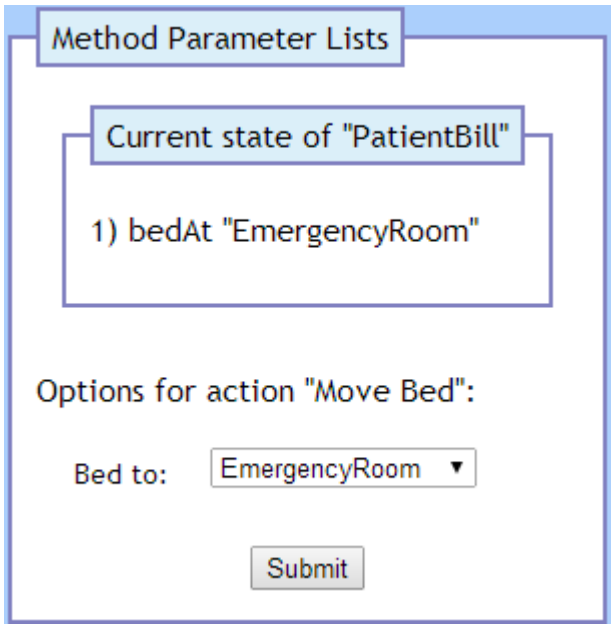

Figure 18. The current value for the selected predicate label bedAt.

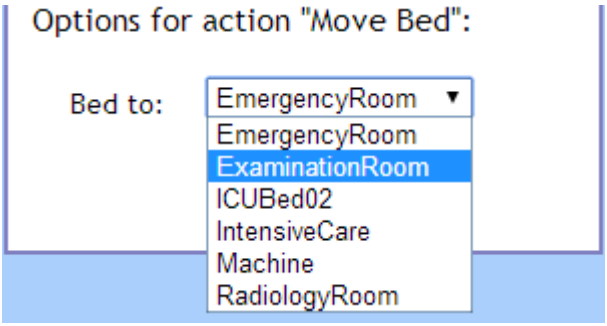

Figure 19. The available values for the predicate label bedAt.

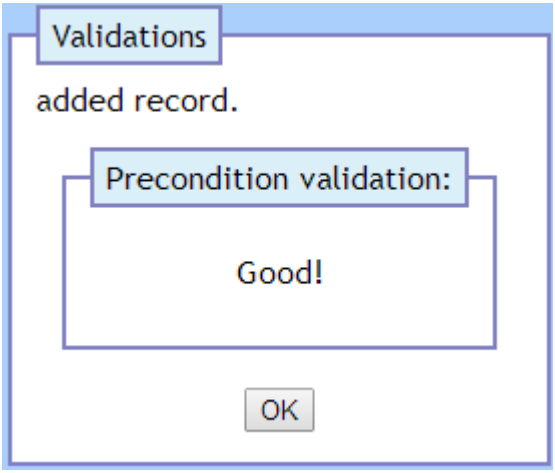

Figure 20. The Asset Interaction Web Interface has updated the corresponding asset value.

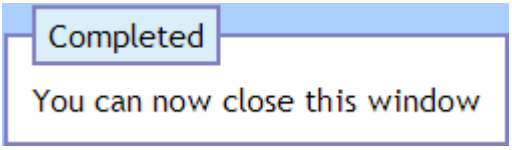

Figure 21. The Asset Interaction Web Interface has been completed for the asset.

The World State table is updated (see Figure 22). The asset PatientBill has its predicate label bedAt changed to ExaminationRoom. An action is inserted into the Asset Service Routine table (see Figure 23). 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 24). The Veis Java Socket Server shows the messages from YAWL to the Simulation (see Figure 25). 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.

	world_key	asset_name	predicate_label	value	TIMESTAMP
Edit Copy Delete	1	VitalSignsMonitor	bandAt	PatientArm	2014-01-26 03:29:49
Edit Copy Delete	1	PatientBill	bedAt	ExaminationRoom	2014-06-09 16:47:16
Edit Copy Delete	1	CoffeeCup	cupAt	Table	2014-01-26 03:29:49
Edit Copy Delete	1	CoffeeCup	cupContains	Empty	2014-01-26 03:29:49
Edit Copy Delete	1	RequestPathology	forPatient	None	2014-01-26 03:29:49
Edit Copy Delete	1	BloodSampleVials	Has	None	2014-01-26 03:29:49
Edit Copy Delete	1	PatientBill	isHearingTested	False	2014-01-26 03:29:49
Edit Copy Delete	1	PatientBill	isVisuallyExamined	False	2014-01-26 03:29:49

Figure 22. The World State table has been updated for the asset PatientBill.

	key	priority	asset_key	service_routine	world_key	TIMESTAMP
Edit Copy Delete	132	1	2297ca59-bfe2-49ac-9265-e4af9b35b5fb	Move:Bed to=ExaminationRoom	1	2014-06-09 23:12:42

Figure 23. The move action to be performed on the asset PatientBill.

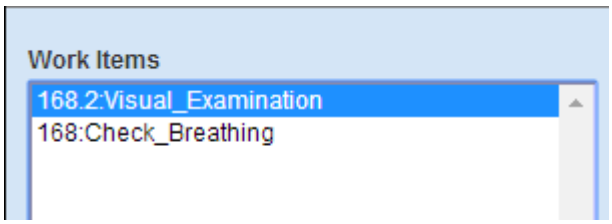

Figure 24. The work item Receive_Patient has been completed and new work has been delegated.

Figure 25. The Veis Java Socket Server passing new work items from YAWL to the Simulation.

The Simulation client has updates to show the new work item and its predicate label value required to complete it (see Figure 26). The GameObject for the PatientBill asset moves in the Unity Scene (see Figure 27). 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.

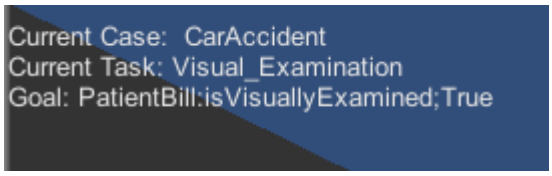

Figure 26. The Simulation client interface has updated to show the new work item Visual_Examination and the action required to complete it.

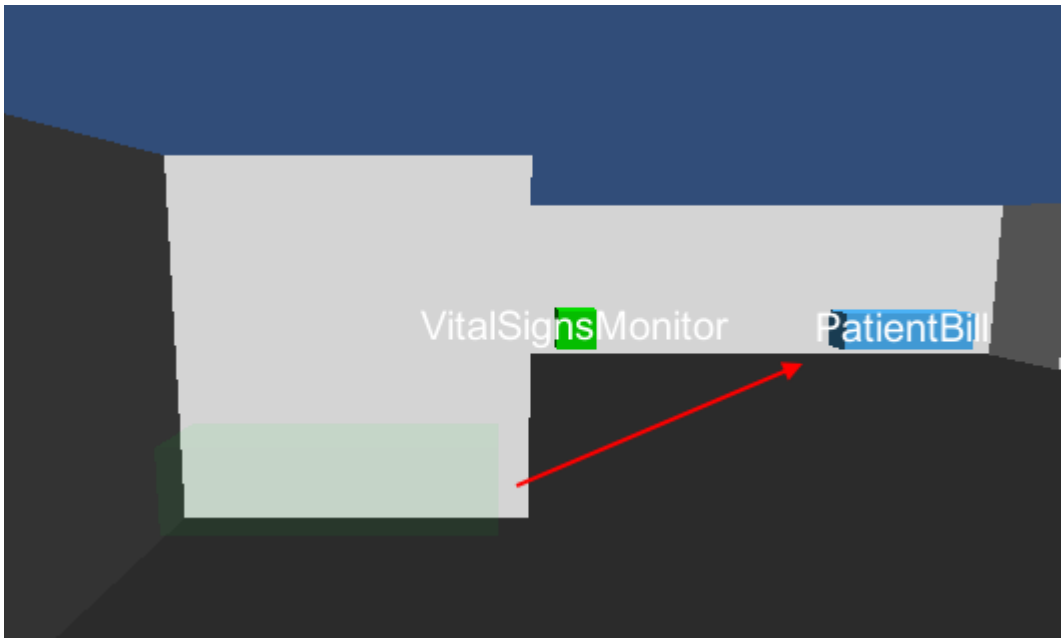

Figure 27. The GameObject for the asset PatientBill has moved to its new position.

4.2.6 Completing the Remaining Work Items

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

After completing the first work item the user moves into the Examination Room (see Figure 28). 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 29).

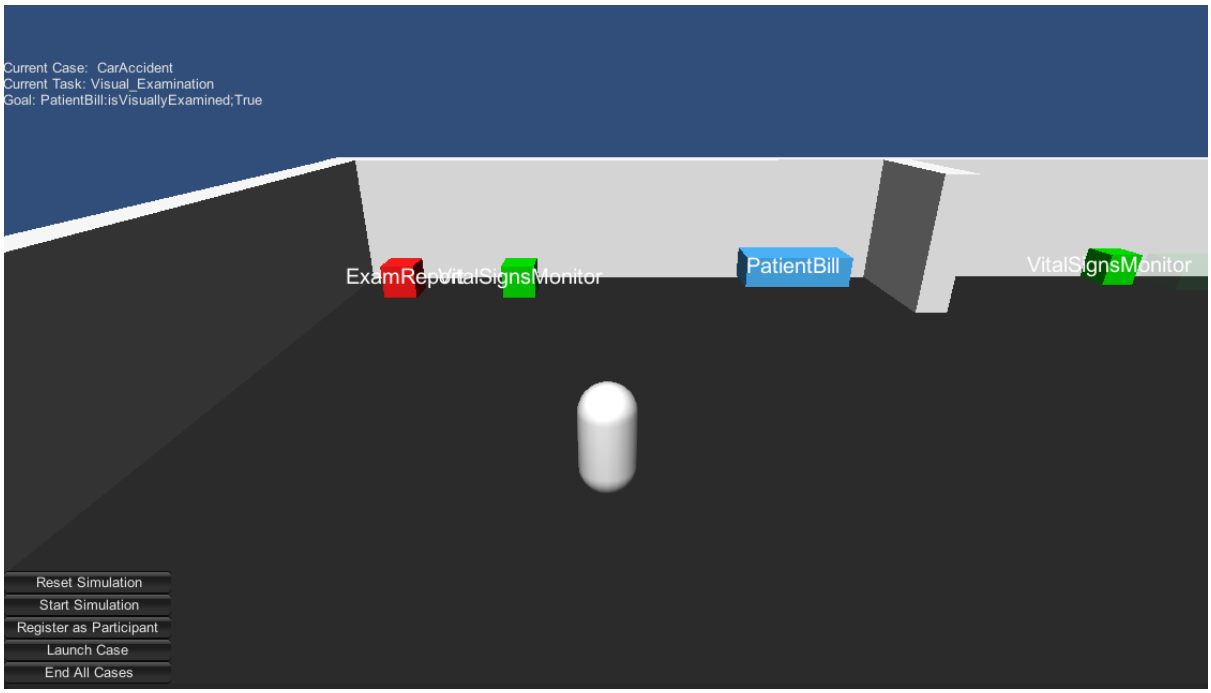

Figure 28. The user has moved their avatar to the Examination Room.

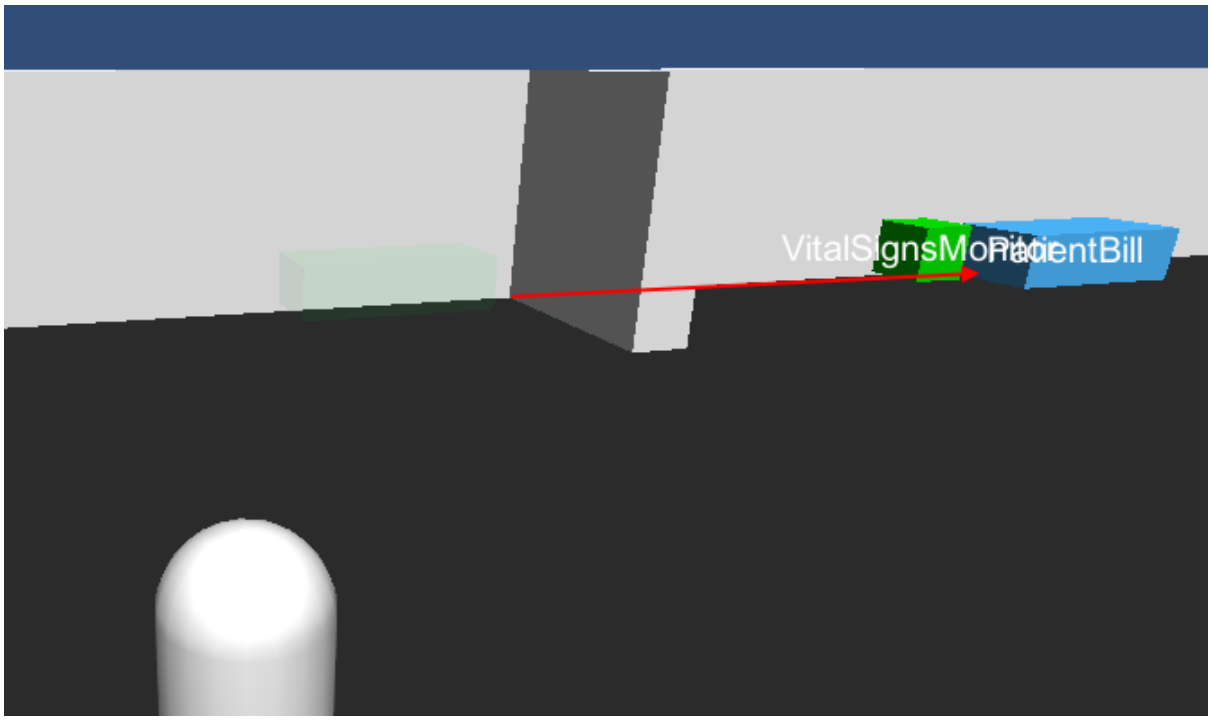

Figure 29. The asset PatientBill has moved from the Examination Room to the Intensive Care room.

The user moves into the Intensive Care room (see Figure 30). The assets in this room are:

- PatientBill
- VitalSignsMonitor
- BloodSampleVials
- RequestXRay
- RequestPathology

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 31).

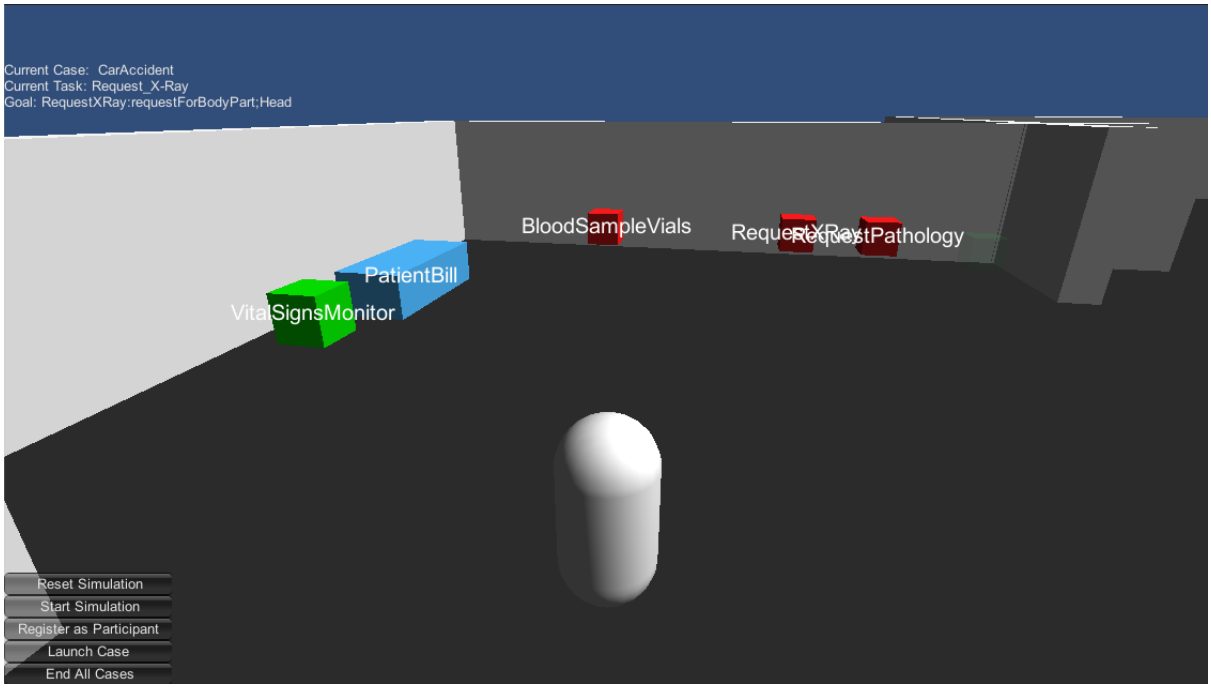

Figure 30. The user has moved their avatar to the Intensive Care room.

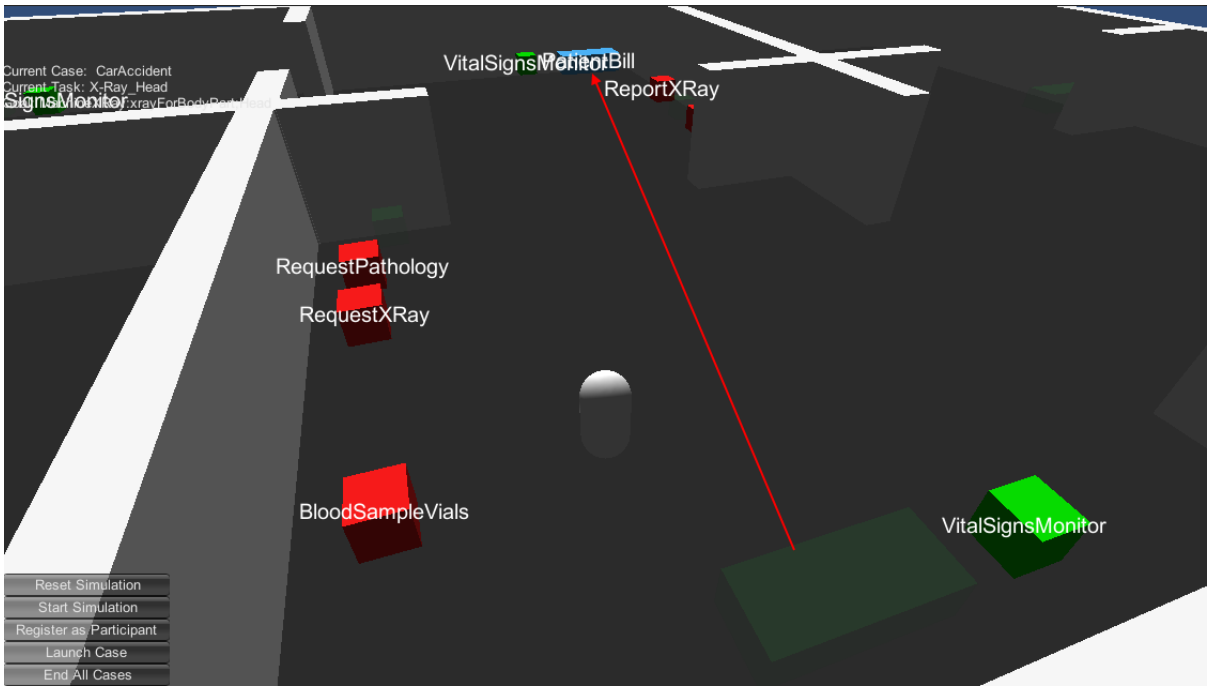

Figure 31. The asset PatientBill has moved from the Intensive Care room to the Radiology Room

The user moves into the Radiology Room (see Figure 32). 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 33).

Figure 32. The user has moved their avatar to the Radiology Room.

Figure 33. 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. The case is now completed. The Simulation client interface updates and no longer shows a case (see Figure 34). The YAWL Web Service confirms that there are no longer any work items (see Figure 35) and there is no

case running (see Figure 36). The Veis Java Socket Server indicates that the case has been completed (see Figure 37).

Figure 34. The Simulation client interface updates to show that there is no longer a case running.

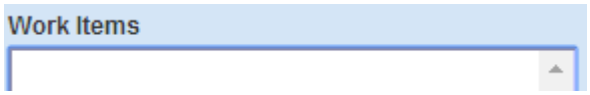

Figure 35. The YAWL Web Interface shows that there are no remaining work items.

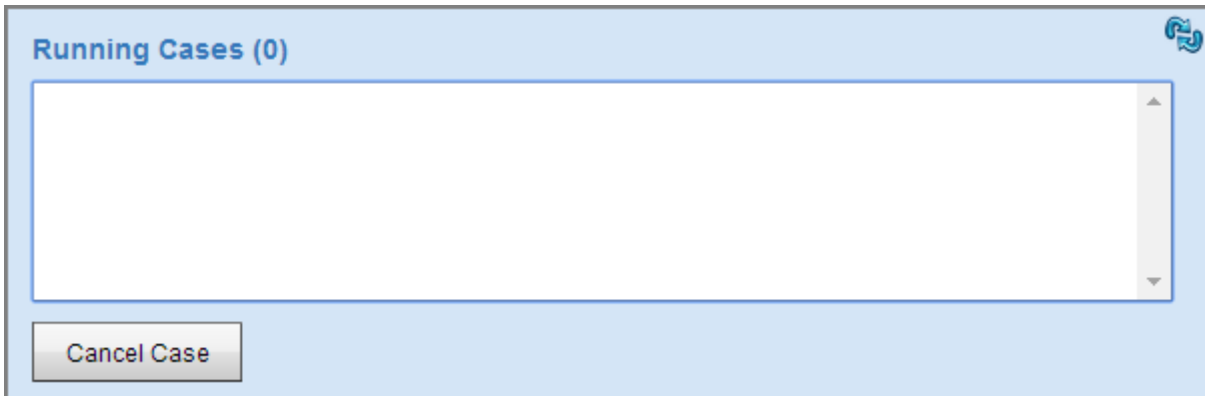

Figure 36. The YAWL Web Interface shows that there is no case running.

Figure 37. 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 38). 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 39).

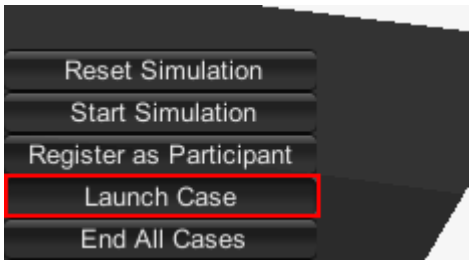

Figure 38. The user can press the Launch Case button to launch another case.

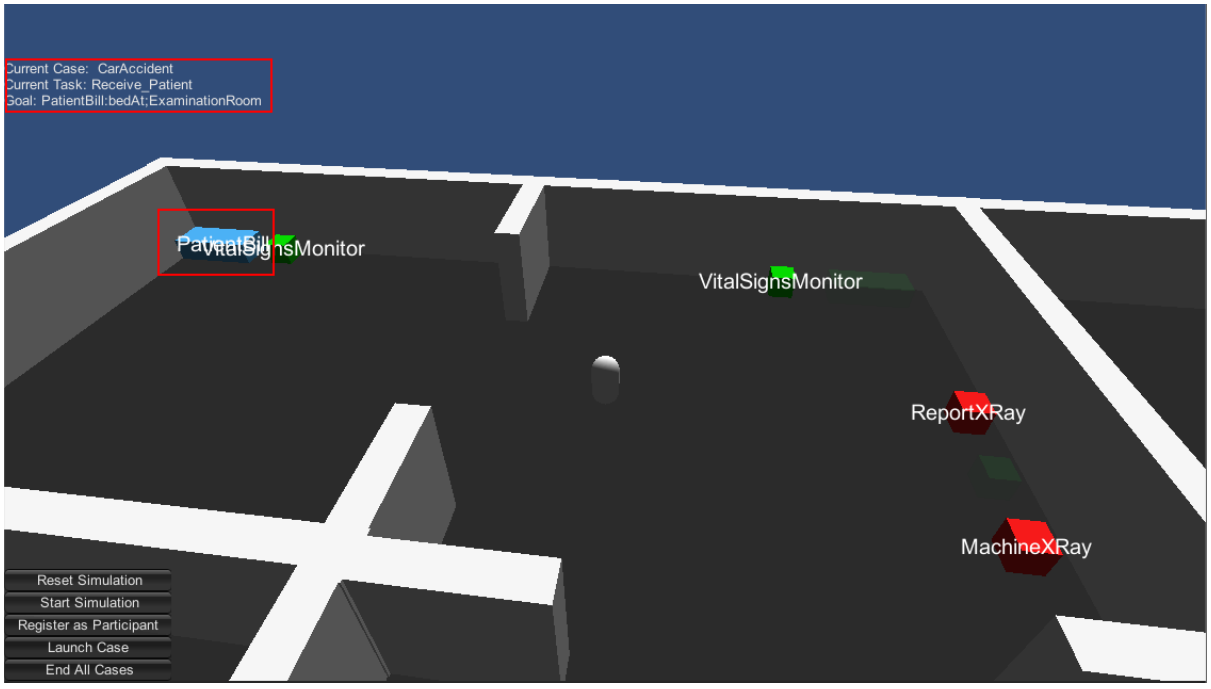

Figure 39. Launching a subsequent case works but dynamic assets will not move to their start position.

4.2.9 Cancelling a Case

A running case may be cancelled regardless of what work items have been completed (see Figure 40). The Simulation client will show that there is no case running (see Figure 41). The YAWL Web Interface confirms that the case has been cancelled (see Figure 42).

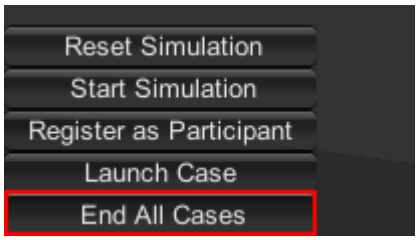

Figure 40. The user can press the End All Cases button to cancel a running case.

Figure 41. The Simulation client showing that no case is running after being cancelled.

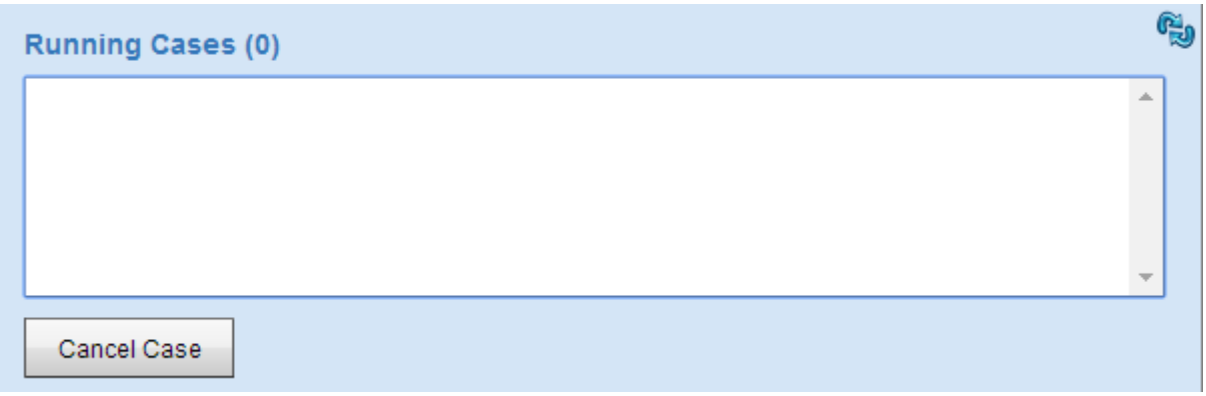

Figure 42. The YAWL Web Interface shows that there is no case running after being cancelled.

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 43 and Figure 44). The Unity compiler output file indicates that the problem arises because these DLLs reference system libraries that are not supported by Unity (see Figure 44). The errors can be subdued by manually including the specific system libraries (see Figure 46). However, the manual inclusion of system libraries can cause conflicts with their Unity version counterparts when trying to use some Types (see Figure 47). 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.

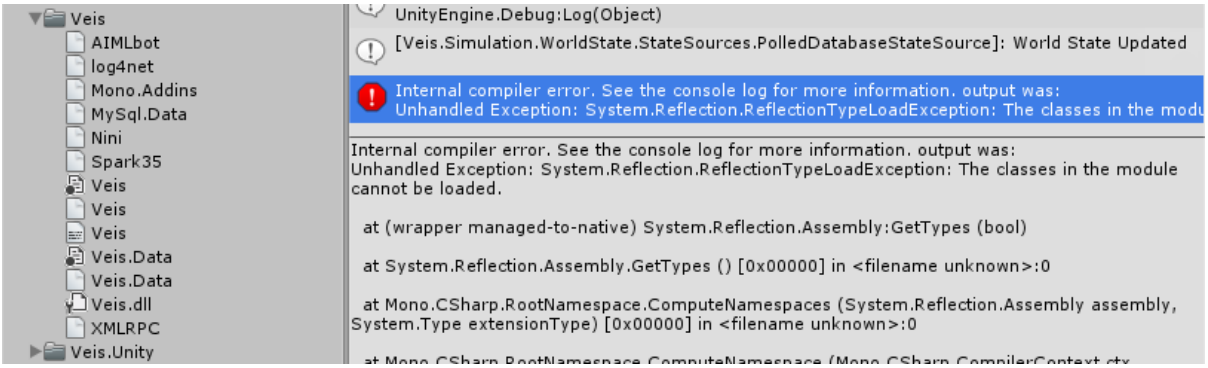

Figure 43. The Veis DLLs and their dependencies, and the error shown when Unity tries to compile them.

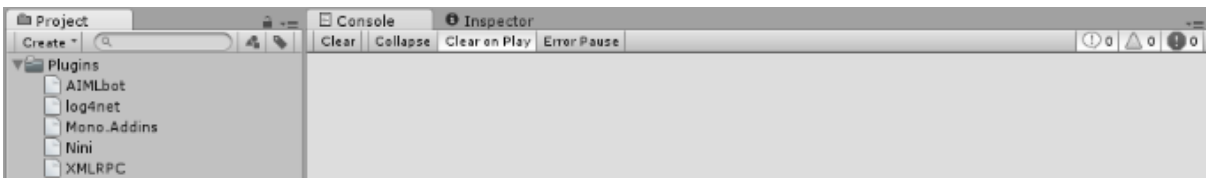

Figure 44. The Unity project will compile when the Veis, Veis.Data, and Spark DLLs have been removed.

```
-----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 45. The Unity Editor log indicates that some of the Veis DLLs reference system libraries not supported by Unity.

Figure 46. The Unity project will compile without error if the error-inducing system references are manually included.

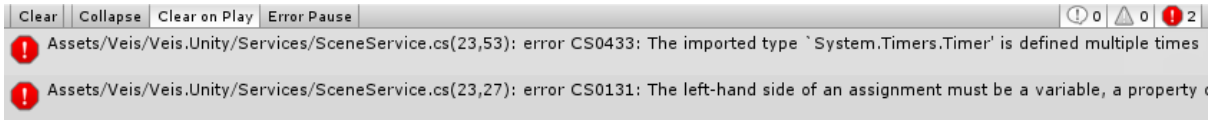

Figure 47. The Unity Editor will not compile when a Type is used that is contained in a system library that has both a Unity and non-Unity version present.

5.1.2 Unity Type Threading Issues

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.