Graphical Simulation and Visualisation Tool for a Distributed Robot Programming Environment

Félix-Étienne Trépanier and **Bruce A. MacDonald**
Department of Electrical and Electronic Engineering
University of Auckland, Private Bag 92019, Auckland, New Zealand
b.macdonald *at* auckland.ac.nz

Abstract

The Robotics Research Group of the University of Auckland is developing a robot programming environment using a Service Based Architecture [Woo, 2002; Woo *et al.*, 2003] to facilitate the description of robot behaviours. Simulation and visualisation tools allow programmers to test their robot behaviours at early stages and such tools are an important addition to any programming environment. This paper describes the development of a Graphical Simulation and Visualisation (GSV) Tool. Robotic applications can automatically test a specific robot algorithm using the CORBA Service offered by the GSV Tool. A GUI enables a human user to visualise the performance of the robot in a specific virtual environment.

1 Introduction

Robot programmers need to be able to visualise and test their robot behaviours on different robots operating in different environments at every development stage. Many simulation and visualisation tools have been developed for robotics development. Some are specific to the robot [Kimoto and Yuta, 1992; Kuffner *et al.*, 2000], although they lack the generality needed for broader experimentation. The current trend is to build general robot simulation and visualisation tools that can be used to test many types of robots operating in different environments. Chen *et al.* [1994] present an elaborate simulation tool with sensor emulation for a robot controlled by Action Scripts. 3d7 [Trieb and von Puttkamer, 1994] aims to simulate different configurations for autonomous mobile robots. At CMU, the Institute for Complex Engineered Systems has built a simulator integrated with a software framework to assist multiple mobile robot collaboration research [Dixon *et al.*, 1999]. Jim Wang from the University of California at Riverside presents in [Wang, 1997] a generic simulation platform for distributed robotic system experiments.

At the University of Auckland, the Robotics Research Group has developed a Service Based Architecture (SBA) [Woo, 2002; Woo *et al.*, 2003]. This software infrastructure defines robotic software as a collection of software components advertised on the network as CORBA service offers. This architecture is the backbone for the development of our distributed robot programming environment. Existing visualisation tools are not suitable for integration within a service oriented programming environment because they don't comply with the two main design principles of the Service Based Architecture: distributed software development and software reuse. Since the group currently investigates robot navigation and localisation tasks with different types of sensors, the tool must be able to emulate those sensors.

This paper presents the details of the GSV Tool Project. The second section presents the development methodology. Then, sections 3 to 6 present the four workflows: requirements, analysis and design, implementation, testing. Finally, section 7 presents an objective evaluation of the tool.

2 Methodology

The development of the GSV Tool implies software engineering. Since the tool is to be integrated in the robot programming environment, the design must be general enough to support many types of robotic applications. For these reasons, a software methodology, the Rational Unified Process (RUP), has been used. Since RUP targets mostly business and team software development, it has been simplified to suit a single developer in a research context, while still allowing future development by the group. The complete description of the methodology used for the realisation of the GSV Tool can be found in [Trépanier, 2003].

3 Requirements

Figure 1 shows how a GSV Tool can be deployed in the Service Based Architecture. The tool must be accessible to both human users (via a GUI) and robotic applications (via a simulation service). The program that controls a simulated robot is contained in a robot behaviour program. This program publishes an agent behaviour service so that the GSV Tool can control the execution of the robot behaviour. The GSV Tool also provides hardware emulators for the behaviour to interact with the simulated world as if it were operating in the real world. All the service offers are managed by a service broker application, the Trading Service application in the CORBA specifications.

The development of the GSV Tool requires the development of two service definitions: Agent Behaviour Service and Simulation Service. The requirements of the GSV Tool Project come from a use-case analysis. In total, 34 use-cases and 46 requirements have been described. This section briefly presents the key requirements of the two services and of the GSV Tool.

3.1 Agent Behaviour Service Requirements

The Agent Behaviour Service system is a CORBA Service definition for agent behaviours. This system allows applications to access robot behaviour programs. In this specific project, the GSV tool must be able to initialise an agent behaviour using this service [**RQ1**] and start, stop, pause and resume the behaviour [**RQ2**]. It is important to mention that the Agent Behaviour Service does not implement those requirements, but it provides an IDL interface definition for the agent behaviour implementations to support those functionalities.

3.2 Simulation Service Requirements

The Simulation Service system is a CORBA Service definition for simulation tools. This system provides access to the GSV Tool as a distributed software service. It allows the creation of a new simulation [**RQ3**]. This functionality involves loading an environment, setting the simulation parameters and adding robots by specifying their model and agent behaviour. It also allows a robotic application to start and stop a simulation [**RQ4**]. Moreover, the Simulation Service system allows the robot behaviour to interact with the virtual environment by giving each agent the ability to act and sense within the simulated world [**RQ5**]. Again, the Simulation Service does not implement **RQ3** to **RQ5**, but it provides the IDL interface definitions for the GSV Tool to support and publish those functionalities.

3.3 GSV Tool Requirements

The GSV Tool system consists of a software application that performs graphical simulations of virtual robots in virtual environments. This system implements the Simulation Service and uses the Agent Behaviour Service offers to control simulated robots. The tool must provide the simulation management and the graphical display of the evolving virtual world. The simulation tool can be started in server mode [**RQ6**]. This means that the GSV Tool registers itself to the service broker as a simulation service offer and waits for robotic applications to create a simulation. Otherwise, the GSV Tool is started as an application. Then, a user can create a simulation by loading the environment, setting the simulation parameters, importing the model for the robot and possibly importing an avatar model [**RQ7**]. When the simulation has been created, the simulation user can start, pause, resume and stop the simulation at any time [**RQ8**]. The user can also save the simulation so it can be replayed at a later time [**RQ9**]. If a simulation is already running on another GSV Tool, a simulation user can view the simulation by connecting to this other instance of the GSV Tool [**RQ10**]. The simulation must be in 3D so it looks realistic and allows more complex sensors such as cameras to be modelled [**RQ11**]. The GSV Tool implements the Simulation Service interface [**RQ12**].

4 Analysis and Design

A key specification for this project is to have real-time 3D graphics capabilities [**RQ11**]. One option is to create a real-time 3D engine by using graphics libraries such as OpenGL or OpenInventor. However, reusing a game engine that already supports real-time 3D graphics is a very appealing idea that could save a significant amount of development time. After some analysis on game engines available at low cost: Torque, Quake and Unreal Tournament, Torque [Garage Games] emerges as the best option. Even if it is not the most advanced engine, it has been chosen for the following reasons:

- The Torque Game Engine from Garage Games is already being used in the Architecture Department of the University of Auckland for real-time multi-user critique of architectural designs. Thus, the university already possesses the engine.
- It is the only option that provides the full source code with the engine. This gives more flexibility as it is possible to modify the game engine itself.
- It has been successfully used for 2 years within the university for projects such as the architecture design critique tool. There is expertise with this specific game engine within the Departments of Architecture, EEE, and Computer Science.
- Torque comes with a scripting language that can be used to slightly alter the game or to easily create GUIs.

Since Torque supports multi-player games, game recording and real-time 3D graphics, **RQ8**, **RQ9** and **RQ10** are met.

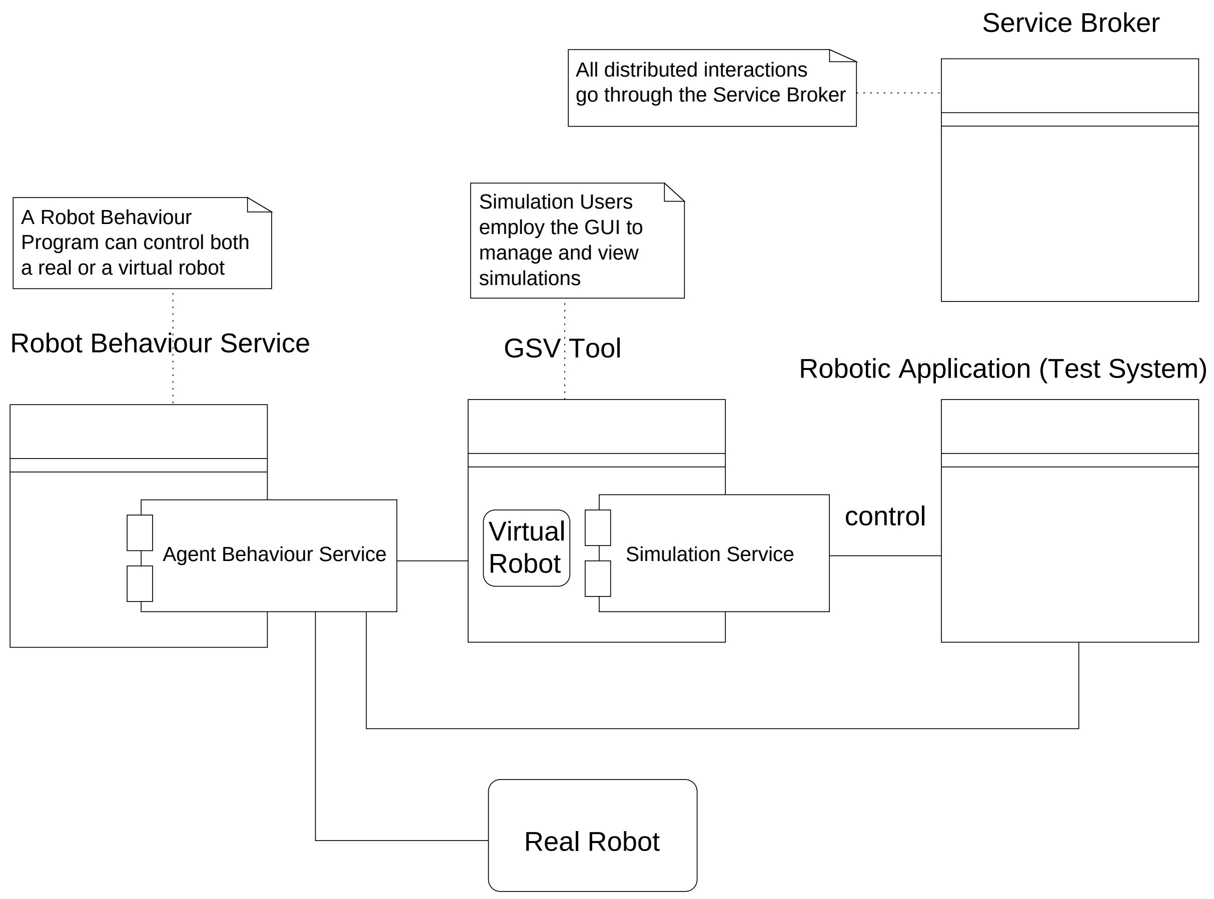
Service Broker
All distributed interactions go through the Service Broker
A Robot Behaviour Program can control both a real or a virtual robot
Simulation Users employ the GUI to manage and view simulations
Robot Behaviour Service
GSV Tool
Robotic Application (Test System)
Agent Behaviour Service
Virtual Robot
Simulation Service
control
Real Robot

Figure 1: A GSV Tool in the Service Based Architecture. A robot application under test may control the simulation, and may use the agent behaviour service to determine robot behaviours. A real robot may also use the robot behaviour service. Where a user controls the simulation and visualisation, the GSV tool may use the robot behaviour service to determine robot actions. All distributed communications occur via the service broker (links not shown).

4.1 Agent Behaviour Service Design

The Agent Behaviour Service consists of one interface. This interface has a *init*, *start*, *stop*, *pause* and *resume* method to fulfil requirements **RQ1** and **RQ2**.

4.2 Simulation Service Design

The Simulation Service has one main interface that allows a robotic application to manage a simulation. Since it must also allows Agent Behaviours to sense and act within a simulation using emulated sensors and actuators, it has interfaces that allow the creation of sensors and actuators (see figure 2). These interfaces fulfil **RQ3**, **RQ4** and **RQ5**.

4.3 GSV Tool Design

Figure 3 shows the simulation management design for the GSV Tool. Since the game engine already supports environment management, only a few classes are needed. The Simulation class is responsible for keeping the information about the current simulation such as the name, the duration, the current time and the list of robots in the simulation. It also responds to the user GUI commands to create and manage a simulation (**RQ7** and **RQ8**). Each robot entity (3D representation) in the simulation is managed by the AgentConnection class which derives from the Torque GameConnection class used for players. Since the GameConnection class implements most of the logic needed to control an entity in the simulation, the AgentConnection has only to manage the robot's movements. The behaviour of the robot is controlled via the Agent class. This class contains the agent name and, most importantly, a reference to the agent behaviour that controls the robot.

The class diagram also shows the simulation service implementation. The CorbaManager class has been designed to handle all the interactions with the Service Broker. Thus, this class is responsible to request the available Agent Behaviour Service offers from the Service Broker and to publish the GSV Tool as a Simulation Service offer on the network (**RQ6**). The CorbaManager class also implements the Simulation interface from the Simulation Service (**RQ12**). So all the requests to the Simulation interface are in fact received by the CorbaManager object which, in turn, forwards them to the Simulation class.

To provide access to emulators, three classes have been added to the design. The HardwareEmulator class provides access to both the ActuatorProxy and the SensorFactory which implement interfaces from the Simulation Service.

At this development stage, each robot can only use one high-level actuator to move. Thus, the ActuatorProxy class is used by the simulated robots to move within the virtual world. On the other hand, a single robot may need many instances of one type of sensor. Therefore, the SensorFactory is used to create a sensor emulator for each sensor the robot behaviour requires.

5 Implementation

The design model has been constructed with the Rational Rose C++ tool. The forward engineering functionality of the tool has been used to generate both the IDL and C++ class definitions presented in the previous section.

Figure 4 shows the relations between the GSV Tool and the Torque Game Engine framework. Briefly, the Torque Game engine framework consists of three subsystems. The Environment subsystem manages the state of the simulated world. Thus, each object in the virtual world is represented in the environment. This subsystem updates the world state and provides the scene for the Graphics subsystem. The user's display is produced by the Graphics subsystem which converts the world internal representation into a 3D graphical representation. The avatars (players) are controlled by the Player Management subsystem. This subsystem receives information from the user GUI, updates the state of the player object which in turn affects the environment. These three subsystems have not been modified. They provide the powerful Torque framework.

The implementation of the GSV Tool has been roughly divided into five subsystems. First, the GUI for the simulation user has been built using the Torque GUI editor. Then, the Simulation Management subsystem has been developed to provide the logic behind the user interface. Once the basic simulation management functionalities have been added, the robot management, a simple extension to Torque player management, has been implemented. Since the GSV Tool can also be used by robotic applications distributed on the network, the Simulation Service Implementation has been completed. Finally, to simulate the robots, the actuator and sensor emulators have been created.

The ActuatorProxy reuses the Torque movement mechanism to move robots. The positioning system sensor returns simply the global coordinates of the robot in the virtual world. The current implementation of the sonar sensor involves a simple ray cast from the sensor position up to the sensor maximum range using a built-in Torque function. If an object is hit, the distance is computed and returned only if it is greater than the minimum range, otherwise the maximum range is returned. Figure 5 shows a B21r robot using its sonar sensors in a simulation.

For the camera sensor, the scene is rendered from the robot point of view in a auxiliary OpenGL buffer that is not to be displayed. This buffer is then copied and sent to the requesting Agent Behaviour. Currently, the

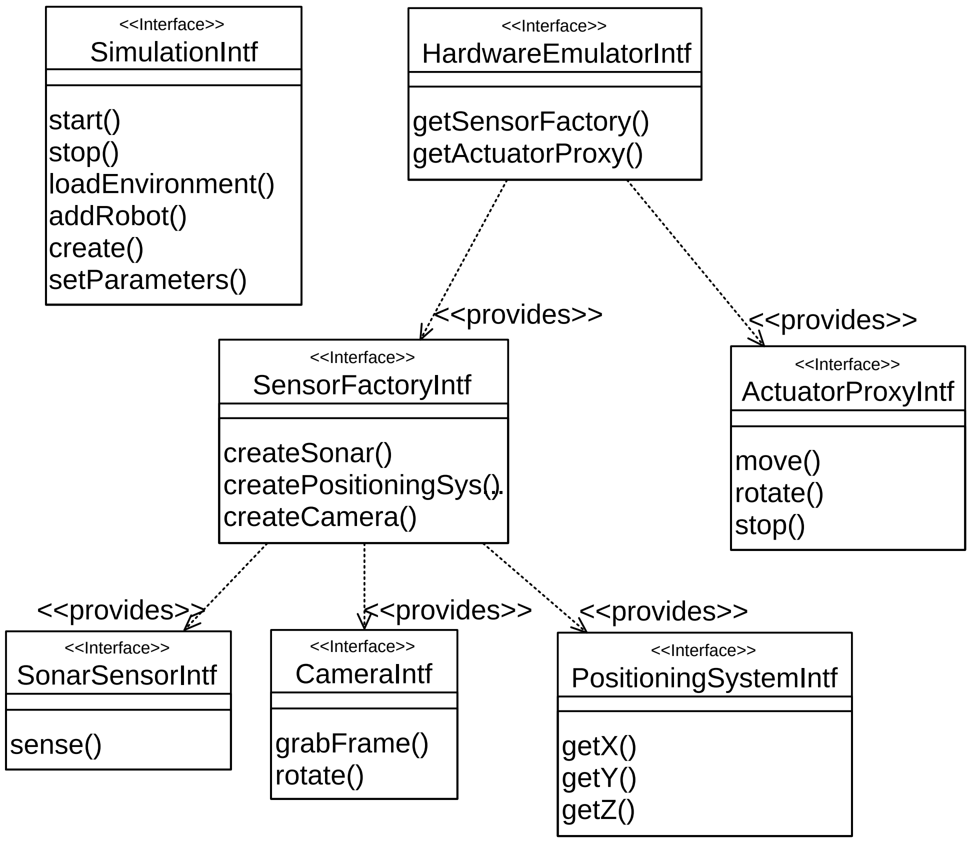

Figure 2: Simulation Service interfaces

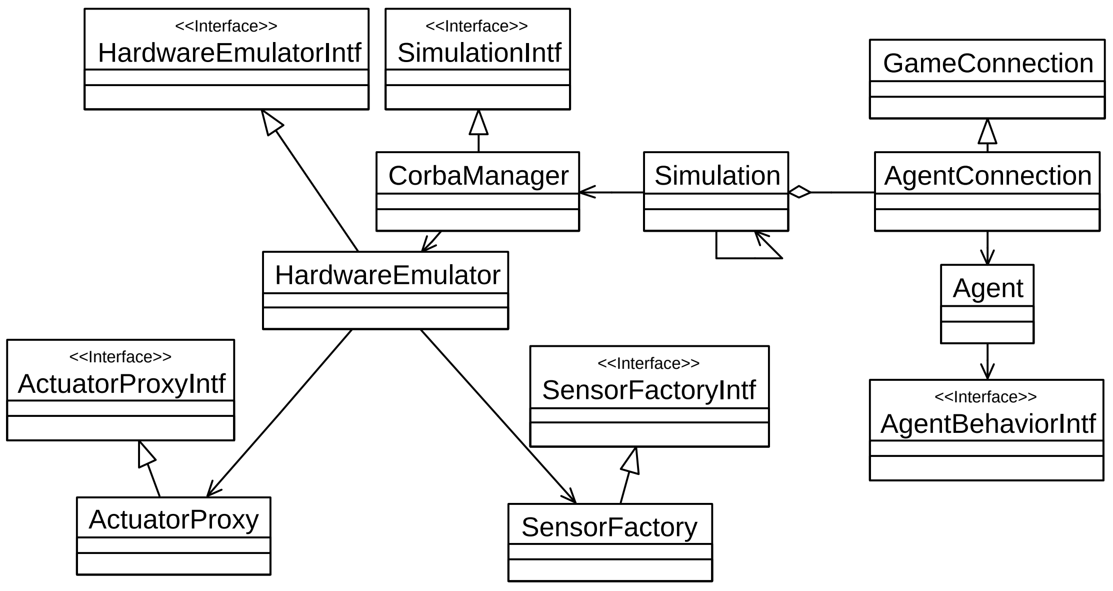

Figure 3: Simulation management class diagram

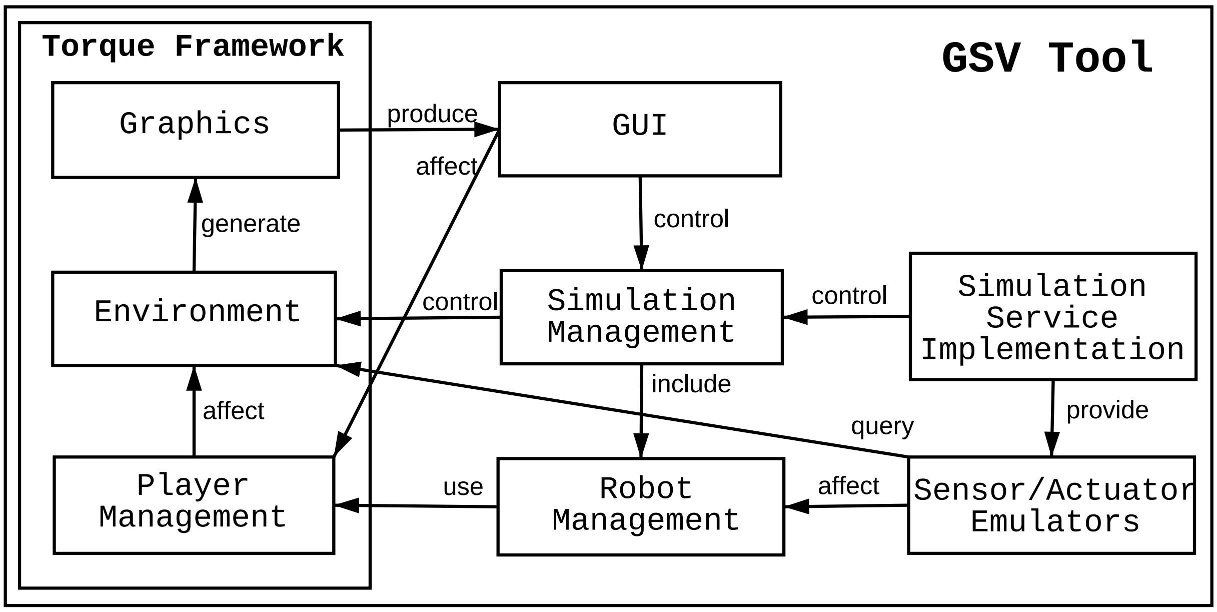

Figure 4: GSV Tool implementation and its relation with the Torque framework

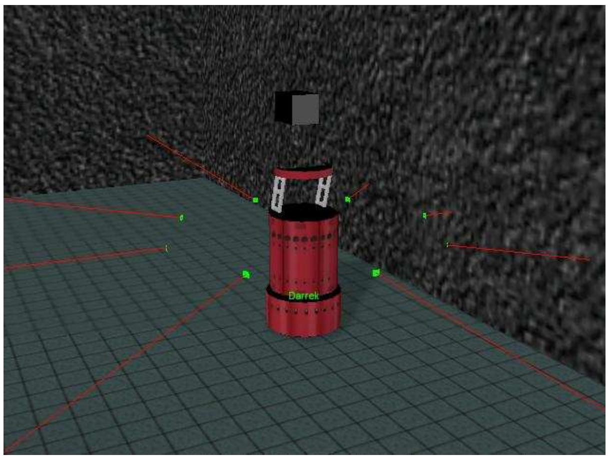

Figure 5: The B21r using its sonar sensors (each red line indicates a sonar reading length) and its camera (the black box on top of the robot) in a simulation

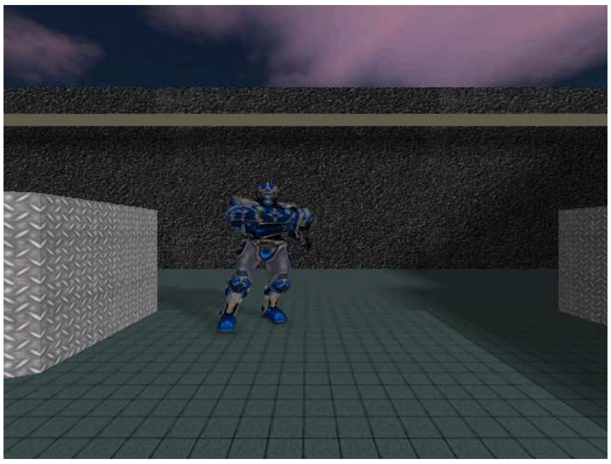

Figure 6: A picture taken by the B21r using its camera in a simulation

picture data is in the 8 bit RGB format. Figure 6 shows a picture taken by a robot.

Since each sensor is contained in one class and accessed through an interface, the implementation of the sensors can be easily changed without having any impact on the logic of the robot behaviours using them. New sensor can also be added by defining new interfaces and then creating the implementation in the GSV Tool.

6 Test

As noted before, the requirements come from both a use-case analysis and a non-functional requirement analysis.

The test phase of the methodology consists of creating at least one test case for each use-case and non-functional requirement. The complete test suite of 63 test cases is given in [Trépanier, 2003]. Torque profiler, the ORBacus Trader Administration Tool, a Test Agent Behaviour and a Test Robotic Application have been used to perform and validate the test cases. The test phase discovered 4 defects. Three of them have been fixed. The fourth defect is considered acceptable as the failure does not compromise the quality of the system. The programmer had to add some logic to the main program loop in order to add a camera sensor emulator, which breaks a non-functional requirement about the ease of extending the design. It is not considered serious.

7 Evaluation

Once the GSV Tool and the two services had been completed, an evaluation was conducted. The total number of lines of code written in this project is 2700. This seems small, however the Torque Game Engine has provided a useful basis on which to build the GSV Tool. Hence, a greater part of the programming effort has involved understanding and reusing the Torque Game Engine framework to its full potential. The Game Engine has more than 113,000 lines of code (including the scripts). Thus, even if the number of code lines added is small, the overall GSV Tool is large. The number of classes added to the Torque Game Engine is 18 and each IDL class is in fact an abstract interface. Since many of the 18 classes had to either inherit from a class in the Torque framework or from an abstract interface, the number of inheritance links is 23 which is relatively high. The Max Fan In value is 3. Usually, a Max Fan In value greater than 1 indicates a complex inheritance hierarchy. In this case, this is justified by the fact that all the classes that implement a CORBA distributed object interface must inherit from many CORBA interfaces: the object interface (one of the IDL interface definitions) and one CORBA interface. The overall average number of methods per class (5.88) and the average number of lines of code per method (15) are both low. This low complexity is highly desirable since the tool must be easily expendable for future research.

Some experiments have been conducted to measure the tool performance. One of these measures the GSV Tool performance with one to five robots having a *typical* behaviour. The *typical* behaviour reads 8 sonar sensors, and then moves forward or turns depending on the readings. After activating its virtual motors, it waits for 50 ms before the next sonar scan. Moreover, the behaviour takes a picture every second and then waits for 20 ms. This 20 ms delay represents the processing time of the image. Using this behaviour to control the simulated robots, the GSV Tool performance degradation is shown

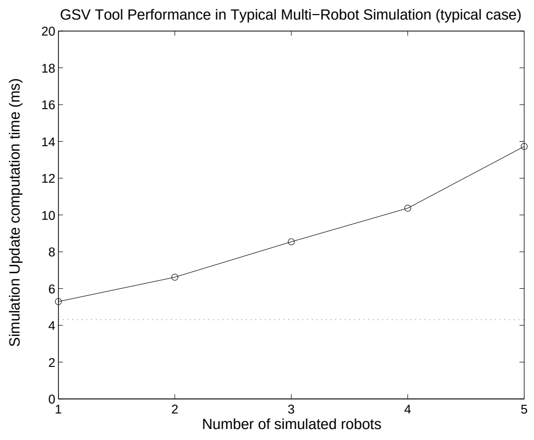

Figure 7: Effect of the number of robots on the GSV Tool performance (typical case)

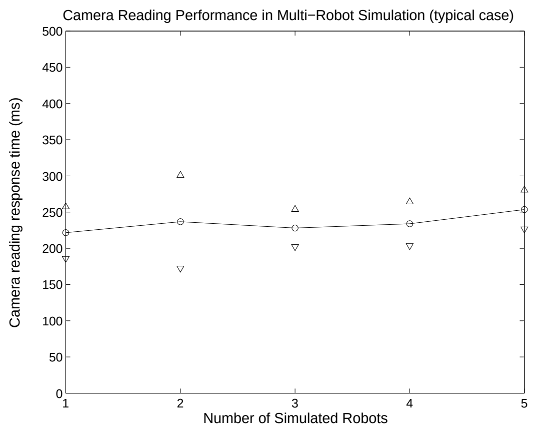

Figure 8: Camera reading response time (typical case)

in figure 7. Even with 5 simulated robots, the simulation state is updated on average in less than 20 ms which is fast enough to have a smooth display. But each time a picture is taken using the camera emulator the refresh takes longer. With five robots, the display appears slightly jerky. Figure 8 presents the average camera emulator reading response times for one robot. The camera emulator response time is relatively constant and thus predictable. This result also show that the number of robots does not influence significantly the performance of the sensor emulators.

The focus of our work has been to provide a tool for graphical simulation and visualisation. Robot devices have been implemented in the simulation to evaluate the tool's capabilities. The simulation models have not been systematically validated against real robot sensors and actuators. Current work includes physical modelling of robot devices, which will be validated against reality.

8 Conclusion

This paper has described the GSV Tool Project which helps humans to visualise robot behaviours in different environments. It is integrated in a broader robot programming environment supported by a Service Based Architecture. The GSV Tool can display any robot model controlled by a robot behaviour using a state-of-the-art game engine to render the virtual world thus giving an accurate 3D perspective of the evolution of the robots in the virtual environment. The GSV Tool also uses services registered to a Service Broker to control the simulated robots. Moreover, the tool can also register itself as a Simulation Service so robotic applications can use the tool without the intervention of a human. A simulation can be accessed by many researchers at the same time and the robots are controlled by behaviours, possibly from another research group, available from the service broker. In addition, a simulation can be recorded and played at a later time to show the results to a wider audience. The performance measurements have shown that the simulation of five robots with *typical* behaviour in the same environment seems to be the upper limit for the tool at this stage.

References

[Chen *et al.*, 1994] ChuXin Chen, Mohan M. Trivedi, and Clint Bidlack. Simulation and animation of sensor-driven robots. *IEEE Transactions on Robotics and Automation*, 10(5):684–704, October 1994.

[Dixon *et al.*, 1999] Kevin Dixon, John Dolan, Wesley Huang, Christian Paredis, and Pradeep Khosla. RAVE: A real and virtual environment for multiple mobile robot systems. In *Proc. of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1360–1367, 1999.

[Garage Games,] GarageGames.com, http://www.garagegames.com/. *Torque Game Engine.*

[Kimoto and Yuta, 1992] Katsumi Kimoto and Shin'ichi Yuta. A simulator for programming the behavior of an autonomous sensor-based mobile robot. In *Proc. of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1431–1438, 1992.

[Kuffner *et al.*, 2000] James J. Kuffner, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Graphical simulation and high-level control of humanoid robots. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'00)*, volume 3, pages 1943–1948, November 2000.

[Trépanier, 2003] Félix-Étienne Trépanier. A graphical simulation and visualisation tool for mobile robotics applications. Master's thesis, University of Auckland, July 2003. Under examination.

[Trieb and von Puttkamer, 1994] Rainer Trieb and Ewald von Puttkamer. The 3d7-simulation environment: a tool for autonomous mobile robot development. In *Proc. of the Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 358–361, 1994.

[Wang, 1997] Jing Wang. Methodology and design principles for a generic simulation platform for distributed robotic system experimentation and development. In *International Conference on Systems, Man and Cybernetics*, pages 1245–1250, 1997.

[Woo *et al.*, 2003] Evan Woo, Bruce A. MacDonald, and Félix Trépanier. Distributed mobile robot application infrastructure. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1475–1480, Las Vegas, October 2003.

[Woo, 2002] Evan Woo. A distributed application infrastructure for mobile robot. Master's thesis, University of Auckland, November 2002.