CS4099 Major Software Project Final Report
Crossing Reality, Two Worlds At A Time:
A Toolkit For Building Cross Reality Worlds

Christopher John Davies

15/04/2011

Abstract

This project presents a software toolkit that facilitates development of cross-reality systems leveraging wireless sensor networks, actuators and virtual worlds. The toolkit allows physical and environmental changes in the state of a real world location to be sensed and reflected in a simulation of that location in a virtual world, whilst certain avatar-induced actions in the virtual world are reflected in the real world by actuators. The toolkit is demonstrated by implementation of an example use case scenario.

Declaration

"I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

"The main text of this project report is 19,950 words long, including project specification and plan.

"In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright of this work."

Contents

List of Figures

Chapter 1

Introduction

Virtual worlds are a genre of online community, often now comprising a computer-based three-dimensional graphical environment, in which (usually multiple) users can interact with each other and create and interact with objects [1]. One obvious attraction of virtual worlds is their suitability for simulating and mimicking real world locations.

By introducing readings from a Wireless Sensor Network (WSN) distributed throughout a real world location into a virtual world simulation of that location, the state of the simulation can be made to dynamically respond in real time to environmental changes in the real world; light, temperature, humidity, movement, etc. Likewise actions performed by an avatar in the simulation can be made to affect the real world location by use of actuators; controlling lights, radiators and air conditioning units, activating door openers, etc.

This combination of augmented virtuality (introducing sensor readings to a virtual world) and augmented reality (commanding real world actuators from a virtual world) gives rise to a specific type of mixed reality [2, 3] situation known as cross-reality. [4, 5] These terms are explained in further detail in Chapter 3.

This project presents the design and development of a software toolkit that allows creation of such cross-reality systems leveraging WSNs, actuators and virtual worlds.

This document is a full report on the development of the toolkit, from the initial context and literature survey, through the software engineering process by which the solution is designed, the implementation of this design achieved and an analysis of the effectiveness of the system at fulfilling the specification.

1.1 Problem Definition

The core aims of this project are;

- To produce a software toolkit that facilitates development of cross-reality systems leveraging WSNs, actuators and virtual worlds. This acts as a middleware between virtual world server software, WSNs and actuators,

allowing developers to easily build their own cross-reality systems using similar technologies, with the underlying communications between the different components abstracted away from their view and concern.

- To describe several example use cases where this toolkit could be applied, demonstrating the need for and the suitability of the toolkit.
- To demonstrate the functionality and success of the toolkit by applying it to one of the example use cases.

1.2 Project Success

The main success of the project lies in the software toolkit that is presented. The level of the desired functionality that this toolkit provides is demonstrated by the implementation of an example use case and the accessibility that the toolkit presents to other developers seeking to build a similar cross-reality system.

1.3 Example Use Cases

Cross-reality has potential applications in many fields including entertainment, security, industry visualisation, equipment training, collaborative working and troubleshooting of equipment, remote interaction with dangerous environments, building management and monitoring energy usage. Beneath are three example use cases that this project's toolkit could be used to aid in development of.

1.3.1 Remote Building Management

A family home with WSN nodes distributed throughout the rooms and with actuators attached to or integrated with common household objects and appliances such as the oven, washing machine, thermostat and lighting, would allow family members to ascertain the state of the house even when they are away from home and to remotely alter this state when they desire.

For example if the family were holidaying at winter and were worried about the temperature of the house dropping too low in their absence and risking frozen pipes or damp then they could log in to the virtual world from their holiday destination to visit the simulation of their house which would visually represent the current temperature of the house, perhaps by changing the colour of the walls (there is nothing that requires the simulation to be completely faithful to the original that it represents). They could then manipulate an interface in the simulation that sends commands to an actuator that interfaces with the central heating thermostat in the real house to increase the temperature to the desired level.

A more everyday example would be one member of the family leaving her place of work and logging in to the virtual world before she left the office to turn the oven on via an actuator that interfaces with the real oven and virtual

world representation of the oven, so that it reaches temperature by the time she arrives home and she can immediately begin cooking her dinner.

This use case could even provide a convenient solution to the stereotypical mistake of 'leaving the iron on'.

1.3.2 Monitoring and Control in Dangerous or Inaccessible Environments

An environment that is hazardous for a person to enter to perform some required actions could be simulated using a virtual world, where a WSN whose nodes are capable of withstanding far greater hardship than a human and are distributed throughout the environment provide real time updates of the state of the environment to the simulation and actuators allow control of devices within the real environment from the safety of the virtual world, without a human ever having to set foot in the environment and risking their safety.

A similar analogy to this is a situation in which monitoring &/or control of equipment in a hard or impossible to access location is required, such as the top of a mountain or underwater.

1.3.3 Active Energy Saving

A company worried about their energy consumption and carbon footprint could simulate their premises in a virtual world and distribute a WSN throughout the premises as well as attaching actuators to common sources of energy wastage such as lights that are left on when there is nobody present.

An employee could monitor the virtual world simulation after the offices have closed and if any lights had been left switched on, they could switch them off by issuing a command via an appropriate interface in the simulation to an actuator in the real building. Obviously the employee could just walk around the premises and turn off any lights they come to, but the cross-reality solution has obvious incentives when the premises are physically large (such as a 50-storey tower block), spread across a geographically diverse area (such as an entire university campus) or spread across multiple different security regimes that a single employee might not be allowed to move between.

1.4 Summary

Cross-reality is a combination of augmented reality and augmented virtuality into one situation at the same time. Such a situation can be achieved by introducing sensor readings collected by a WSN into a virtual world simulation whilst at the same time allowing commands from the virtual world to trigger actuator actions in the real world.

This project presents the design and development of a software toolkit that allows these different technologies to be more easily brought together to greatly

aid the creation of such systems, demonstrating its success by implementing an example use case of a cross-reality situation.

1.5 Report Outline

The remainder of this report is organised as follows.

- Chapter 2 outlines the goals and objectives of the project from a high-level perspective.
- Chapter 3 presents a survey of background literature and similar projects, describing the work already done in this area.
- Chapter 4 formally defines the properties that the software solution must have in the form of a requirements specification document.
- Chapter 5 discusses the software engineering process adopted for development of the software solution, with justifications for these choices.
- Chapter 6 covers ethical considerations associated with the work.
- Chapters 7 and 8 give in depth coverage of the design and implementation of the toolkit.
- Chapter 9 presents a critical evaluation of the implemented toolkit, with respect to the original objectives and to related work.
- Chapter 10 provides a summary of the project, emphasising achievements and drawbacks, along with discussion of future directions in which the work could be taken.

Several appendices are also included after the main body of the report.

Chapter 2

Objectives

This chapter outlines the major goals of the project from a high-level perspective, free from details of the specific technological aspects and decisions of the implementation.

Objectives are classified in order of their importance, with primary objectives representing the absolute minimum that must be achieved in order for the project to be considered successful. Secondary objectives are expected to be achieved in order to demonstrate the success of the project. Tertiary objectives will improve the overall finish and polish of the project, but are not integral to its functionality.

2.1 Original Objectives

Original objectives are those that are envisaged from the beginning of the system's lifecycle, before design and implementation of the toolkit begins.

2.1.1 Primary Objectives

1. Develop a software toolkit that facilitates bidirectional informational or media flow between WSNs, actuators and virtual worlds, including;
   (a) An interface for communication from WSN to virtual world.
   (b) An interface for communication from virtual world to actuators.
2. Develop a WSN application to sense real world physical and environmental data and relay these to the virtual world via interface 1a.
3. Design simple 3D user interfaces for virtual world simulations that logically present avatars with the ability to trigger actuator commands via interface 1b.

2.1.2 Secondary Objectives

1. Build a virtual world simulation of a real world location in which one of the example use cases can be enacted.
2. Deploy WSN nodes (installed with the application 2.1.1 2) to this real world location.
3. Deploy actuators to this real world location.
4. Use 2 to visualise real time physical and environmental changes in the real world location in the simulation 1.
5. Use 3 to effect real time physical and environmental changes upon the real world location in response to avatar actions in the virtual world simulation 1.
6. Perform 4 and 5 simultaneously.

2.1.3 Tertiary Objectives

1. Expand toolkit 2.1.1 1 &/or WSN node application 2.1.1 2 to support more physical and environmental stimuli.
2. Expand virtual world simulation 2.1.2 1 to cover more of the real world location, making use of more WSN nodes and actuators.

2.2 Emergent Objectives

Emergent objectives are those that become apparent later during development, such as when the design highlights a required feature missing from the original requirement set as the eventuality had not been foreseen, are brought up in advisor discussion, or are required as a dependency of a certain design decision or implementation approach. These objectives are approved with the supervisor before inclusion in the project.

2.2.1 Primary Objective

1. Become knowledgeable and proficient enough with the emergent technologies involved in creating cross-reality situations (WSNs, actuators and virtual worlds) to be able to design and develop the toolkit so that it exploits their capabilities, acknowledges their restrictions and so that it considers the development environments and methodologies that developers who work in these fields will be accustomed to working with.

   This represents one of the most ambitious objectives as it requires substantial time to be devoted to in-depth research and practice into 3 distinct, new technologies.

2.2.2 Secondary Objective

1. Design and build physical actuator hardware to allow control via 2.1.1 3 of electrical devices in the real world location.

2.2.3 Tertiary Objectives

1. Develop 'virtual' sensor objects in the virtual world to aggregate/average readings from multiple available nearby real world WSN nodes.

2. Further develop and expand upon 2.1.1 3 to allow greater overall control of the toolkit from within the virtual world simulation, facilitating easier 'authoring' of a cross-reality system from a single interface.

2.3 Summary

The primary objectives of this project are concerned with the development of a software toolkit to facilitate easier creation of cross-reality systems leveraging WSNs, actuators and virtual worlds, by abstracting over the underlying communications and organisation between the different technologies. To achieve this it is necessary to perform substantial research, experimentation and familiarisation with these constituent technologies that are integral to creation of cross-reality environments.

The secondary objectives are concerned with using the toolkit developed to create a cross-reality system, thus demonstrating the toolkit's functionality and success. Any tertiary objectives that are met extend the demonstration to further test the toolkit with a larger deployment, or improve the accessibility of the toolkit to developers.

Chapter 3

Context Survey

In this chapter cross-reality and the technologies that are involved in creating a cross-reality system are discussed. For each of virtual worlds, WSNs, actuators and cross-reality a background and history is provided, the current work described and critically evaluated and relationships between this work and the project are identified.

3.1 Virtual Worlds

Virtual worlds are a genre of online community in which users can interact with each other and with objects and the virtual environment around them [1]. Today the term virtual world elicits the idea of a three-dimensional graphical environment, however multiplayer real-time virtual worlds have existed in the looser definition in various forms for many years, right back to the early days of text-only Multi-User Dungeons (MUDs) that first began to appear in the late 1970's. Today there are numerous self-proclaimed virtual worlds such as Second Life as well as more traditional competitive video games that also present the user with the concept of a virtual world for the gameplay to partake within, such as the popular genre of immersive Massively Multiplayer Online Role-Playing Games (MMORPGs).

Arguably the first attempt to create a large scale commercial multi-user virtual environment with a graphical interface was Lucasfilm's 1986 *Habitat*, which was designed to present *"a real-time animated view into an online simulated world in which users can communicate, play games, go on adventures, fall in love, get married, get divorced, start businesses, found religions, wage wars, protest against them, and experiment with self-government."* Habitat proved to be a rich source of insight into the reality of implementing serious, commercially viable virtual worlds and future generations of virtual world builders hopefully benefited from Habitat's developers' experiences and mistakes. [6]

June 23rd, 2003 heralded the release of Second Life from Linden Lab, which includes as part of its software 3D modelling tools to allow users ('residents')

to create virtual objects and also to add interactivity to these objects using an object-oriented scripting language called the Linden Scripting Language (LSL). This permits much interesting research, particularly in the fields of education [7, 8, 9, 10, 11, 12] and collaborative learning [13], data visualisation [14, 15], process simulation [16, 17], Human Computer Interaction (HCI) [18] and more traditional computer science research into Second Life itself [19, 20, 21]. Examples of scientific projects that maintain a presence within Second Life include SciLands, *"a mini-continent and user community in the virtual world platform Second Life devoted exclusively to science and technology"* [22], Nature Publishing Group's Elucian Islands Village [23] and Virginia Tech's Second Life Aided Training Environment (SLATE) [24].

The OpenSimulator (OpenSim) project was founded in January 2007, fueled by the vision of the potential for an open source 3D virtual world server. When the Second Life client was released as open source in the same month, the project became a feasible reality as it would no longer entail writing both a server and client application as the Second Life client could now be used. OpenSim allows any developer to run their own virtual world server with far greater control, permissions and expandability than Second Life, making it more accessible and expansible by the research community and perfect for use in this project.

Messinger et al[25] provide a good background and literature survey into the past, present and future of virtual worlds, exploring why they present such an interesting research area, represent a 'frontier' in social computing and are an important path for design of future (3D) user interfaces [26].

3.2 Wireless Sensor Networks

A WSN consists of spatially distributed sensor nodes that monitor physical and environmental conditions such as light intensity, temperature, humidity, sound levels, etc. and communicate these sensor readings back to a main location, usually via cooperative usage of short-range radios [27].

Akyildiz et al. have published prolifically in the area of WSNs [28, 29, 30, 31], Wireless Multimedia Sensor Networks (WMSNs) [32] and Wireless Mesh Networks (WMNs) [33, 34]. Baronti et al [27] and Yick et al [35] also provide good WSN surveys.

To quote an overview from Akyildiz, *"Recent advances in micro-eletro-mechanical systems technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multifunctional sensor nodes that are small in size and communicate untethered in short distances. These tiny sensor nodes, which consist of sensing, data processing, and communicating components, leverage the idea of sensor networks based on collaborative effort of a large number of nodes."* [28]

WSNs are an active area of research in Computer Science and present interesting possible applications in a range of fields; area monitoring, environmental monitoring, industrial monitoring and fleet monitoring to name some examples. They are also a useful tool for collecting information for an augmented virtuality

system from a physically large area, where traditional sensors wired directly to a basestation computer may be inconvenient or impossible or where the ability to dynamically reposition the sensors without having to worry about the connecting infrastructure is required.

A WSN can be quickly and easily deployed into a particular environment to collect data about that environment and forward these data between nodes to a basestation from where the data can be processed and subsequently used to affect a virtual representation of that environment or of the things in it.

3.3 Actuators

The term 'actuator' covers a wide range of mechanical devices that move or control a mechanism or system, usually by electrical, hydraulic or pneumatic energy [36]. One of the simplest examples of an actuator is an electric motor or a solenoid, but the term also covers more complex concepts such as electroactive polymers which can change size &/or shape when an electric current is applied to them.

The concept of combining actuators with existing WSN technology has begun to be explored by the concept of Wireless Sensor/Actuator Networks (WSANs) [37, 38, 39, 40, 41]. However it is not always desirable to combine sensing and actuation into a single device or architecture. Concerns about the reliability of wireless communications are grave when actuation may involve large, powerful, expensive &/or dangerous equipment for which connection of the actuator to the controlling computer by a traditional cable may be more desirable or even required by safety regulations or law. Naturally there will often be the less extreme case where the position of sensor nodes will simply not be the same as the position of devices that one wishes to actuate. Because of these reasons the combination in a single system of separate WSN and actuator architectures and technologies is expected and the toolkit this project presents is designed with this in consideration.

3.4 Cross-Reality

Cross-reality is the situation created by the multi-directional flow of data or information between the real world and a virtual world, caused by augmented reality and augmented virtuality taking place *at the same time*.

Coleman [4] describes cross-reality as an arrangement wherein there is *"informational or media exchange between real- and virtual-world"*, in particular emphasising a shift from asynchronous, single-directional exchange to synchronous, multi-directional exchange; real to virtual as well as virtual to real. The first example manifestation provided, Eolus One, is a real-estate company that uses sensor networks distributed throughout buildings under their management to collect real-time physical and environmental data that are presented to their

users via a 3D 'virtual command center' based on the OpenSim platform, resulting in greater efficiency in local administration of actual buildings.

Lifton et al [5] describe cross-reality as *"implementations that render and manifest phenomena between real world and virtual environments via densely embedded sensor and actuator networks"* and rather colourfully as *"precipitating when diverse and ubiquitous sensor and actuator networks meet pervasively shared online virtual worlds, where phenomena freely tunnel between real and contrived continua at a multitude of 'wormholes' opened by densely deployed networked devices, seamlessly adapting the level of immersion to match a variable ecology of available interfaces and user context or preference"*. Their work at the MIT Media Laboratory focuses on browsing and interacting with real-time sensors and actuators installed in inhabited structures, with these captured data visualised in Second Life. Lifton also touches on existing commercial implementations of cross-reality, such as IBM's data center operation visualisation in OpenSim [42] and VRcontext's ProcessLife technology that allows remote browsing and influence of industrial processes in real time [43].

Some of the work conducted at the Human Interface Technology Labs (HIT-Labs) in America, Australia and New Zealand is also partially relevant to cross-reality [44, 45, 46, 47].

It is worth contrasting the above corroborating explanations of cross-reality to the related terms of *mixed reality*, *augmented reality* and *augmented virtuality*, which all appear in the literature. Milgram and Kishino [48] not only provide one of the first published definitions of mixed reality but also present the idea of a *virtuality continuum* comprising a linear scale situating an entirely real environment at one extreme and an entirely virtual environment at the opposite extreme (figure 3.1). Occupying any position between these two extremes is considered to be mixed reality, a term for the merging of real and virtual worlds to some extent. A position closer to the real environment extreme constitutes belonging to the subclass of augmented reality (where the environment is predominantly real with virtual augmentations) while a position closer to the virtual environment extreme constitutes belonging to the subclass of augmented virtuality (where the environment is predominantly virtual with augmentations from the real world) [2, 3]. Combining both augmented reality and augmented virtuality into one system to the result of having informational or media exchange in both directions at the same time, from real to virtual as well as virtual to real, is the fundamental criterion for creating a cross-reality environment.

All of these definitions expand upon the conventionally held view of *virtual reality*, where the environment is entirely virtual and completely immersing, allowing the user to interact with a completely synthetic world. Koleva et al. [49] describe the concept of *traversable interfaces* that allow users to transition between real and virtual worlds, dynamically repositioning themselves along this virtuality continuum.

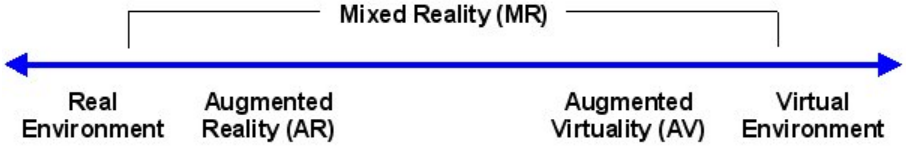

Figure 3.1: Milgram and Kishino's *virtuality continuum*

Coutrix and Nigay [50] provide further detailed definition into mixed reality and the *mixed interaction model*, describing formal formulaic constructions for the linking between real and virtual objects. Kim et al [51] also define cross-reality and explain the importance of a Ubiquitous Sensor Network (USN) middleware to facilitate easy development of the bidirectional information flow between real and virtual worlds that cross-reality depends upon. There are direct allusions between this USN that they describe and the toolkit that this project presents.

3.4.1 MPEG-V

MPEG-V is a standard currently under development by the Motion Picture Experts Group (MPEG) for *"Information exchange with Virtual Worlds"* [52], which *"intends to provide a standardized global framework and associated interfaces, intermediate formats definitions and the like, to enable the interoperability between virtual worlds (as for example Active Worlds, Second Life, IMVU, Google Earth, Virtual Earth and many others) and between virtual worlds and the real world (sensors, actuators, vision and rendering, robotics (e.g. for revalidation), (support for) independent living, social and welfare systems, banking, insurance, travel, real estate, rights management and many others)"* [53]. The intended standard is broadly divided into three parts (also see figure 3.2);

1. Architecture
2. Interfaces between virtual worlds
3. Interfaces between virtual worlds and the physical world

Part 3. is envisaged to provide *"interfaces with peripherals that allow for extension of the experience created in the virtual world into the physical world and vice versa"* [53] so draws obvious analogies to this project. ISO/IEC Committee Draft 23005-2 [54] and Final Committee Draft 23005-3 [55] contain further technical details for Control Information and Sensory Information respectively, whilst Timmer et al. [56] cover the standard from a higher level perspective.

MPEG-V is still under development and has not yet been ratified, however this project presents a partial implementation of MPEG-V by meeting some of the requirements laid out in the 'Requirements for MPEG-V' document [57].

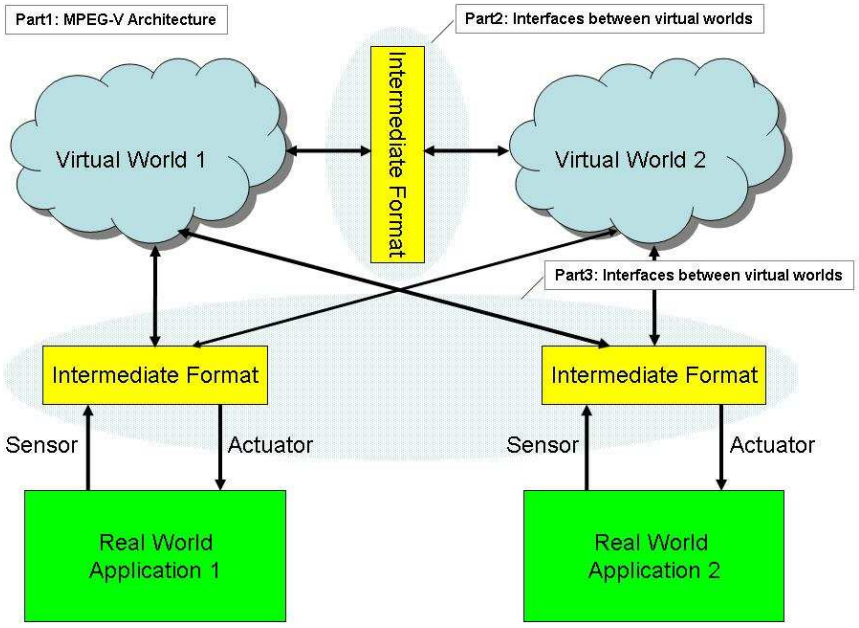

Figure 3.2: Proposed MPEG-V standard

3.5 Present Applications of Cross-Reality

Surveying existing attempts to implement cross-reality systems is necessary to provide metrics against which to critically compare the solution.

3.5.1 Commercial

Existing commercial implementations of cross-reality are limited, but include the aforementioned examples of Eolus One [4], IBM's datacentre visualisation [42] and VRcontext's ProcessLife [43] (see 3.4). Implementations of *mixed reality*, rather than full *cross-reality*, are more common; 'Building and Employing Cross-Reality' was the title of the Works in Progress section of the IEEE Pervasive Computing magazine in late 2009 [58], however of the five examples covered in the article none would be considered cross-reality by the definitions of this report, but would all constitute a lesser mixed reality.

3.5.2 Academic

There is greater interest and activity within the field from academia. MIT are one of the field's most active institutions, with their Responsive Environments

Group at the Media Laboratory responsible for some of the founding work of the field.

One could see MIT's Dual Reality Lab that resulted in the publication of Lifton's 2007 PhD entitled 'Dual Reality: An Emerging Medium' [59] as the beginning of cross-reality research. Their original definition of dual reality is as follows;

> *" 'Dual reality' is the concept of maintaining two worlds, one virtual and one real, that reflect, influence, and merge into each other by means of deeply embedded sensor/actuator networks. Both the real and virtual components of a dual reality are complete unto themselves, but are enriched by their mutual interaction. The dual reality Media Lab is an example of such a dual reality, as enabled the Plug sensor/actuator network that links our actual lab space to a virtual lab space in the Second Life online virtual world."* [60]

The PLUG platform was a bespoke combination of WSN node and actuator, which could sense several different phenomena and independently switch 4x separate mains outlets for control of electrical equipment [61]. An application of PLUGs to distributed, decentralised automatic conversation shielding is described in [62]. A labelled photograph of a completed PLUG is included as Figure 3.3.

Dual reality work with the PLUG evolved into the cross-reality initiative and the Ubiquitous Sensor Portal project; [63]

> *"We have built a sensor network composed of 45 'Ubiquitous Sensor Portals' distributed throughout the real-world Media Lab. Each portal, mounted on pan/tilt platform, has an array of sensors, as well as audio and video capabilities. Video is acquired via a 3 MegaPixel camera above a touch screen display."* [64]

A photograph of one of these Ubiquitous Sensor Portals is shown in figure 3.4.

This represents one of the most ambitious cross-reality projects to date, combining live bi-directional video transmission between real and virtual worlds, transmission of audio from the real world to the virtual world and even detecting when a user is present at a Portal in the real world and representing this presence in the virtual world through use of an infrared sensor;

> *"Each portal will have a related incarnation in SecondLife... ...SecondLife visitors can see live video from any of the portals in the real world... ...They can also communicate with the real-world portal by touching a 'Talk' button on the portal – at this point, communication from SecondLife to the real world is through text, but if the request to talk is granted in the real world, audio from the sensor board will also stream from the real world to the virtual portal,*

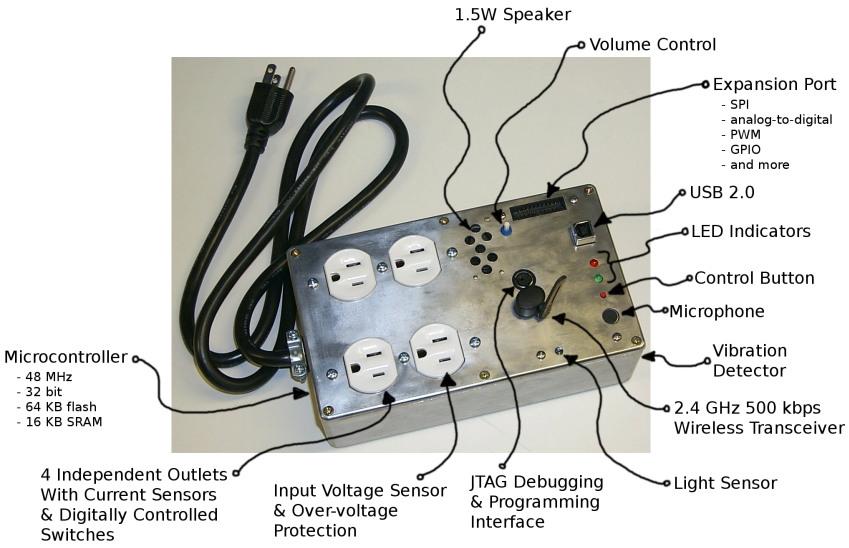

Figure 3.3: MIT 'Dual Reality' PLUG sensor/actuator hardware platform

enabling 2-way communication.... ...The virtual portals will also show 'ghosts' at the portals' sides to reflect the presence of particular individuals standing in front of the real-world portal (as detected by IR from their badge)." [64]

The latest work from MIT is the ongoing DoppelLab project which aims to address the current lack of comprehensive visualization tools for fusing disparate data produced by WSNs which are now being deployed about homes, offices and research institutions;

"Using the popular game engine Unity3D, we transformed architectural models of the MIT Media Lab complex into a browsing environment for real-time sensor data visualizations, as well as an open-ended platform for building visual applications atop those data." [65]

The DoppelLab is still in its early stages and is yet to publish − the only information about the project is contained in a short video posted to the project's home page. [65] Although the home page and video mention that *"applications in turn become sensor-driven interfaces to physical world actuation and control"* and even mention that the visualisation of their own Media Lab building discovered a previously unknown problem with the HVAC which was then attended, there is no mention of whether actuators actually interface directly with the

Figure 3.4: MIT Cross-Reality Ubiquitous Sensor Portal

system to produce a cross-reality situation, or whether this 'actuation' is the result of a human observer performing an action based upon an observation of the system's visualisations on the computer screen.

3.5.3 Conclusions of Present Applications

With the possible exception of the DoppelLab all of the examples discussed in 3.5.1 and 3.5.2 represent bespoke, one-off implementations of cross-reality systems that do not attempt to define standards, protocols, architectures, interaction models or deployment plans to aid future development of similar systems by other researchers and institutions.

The extent that the DoppelLab project will actually aid development of

cross-reality systems by researchers outside of the project is yet to be determined as details of the project are still limited.

This represents a significant shortcoming in the current state of cross-reality research as there is no standardised way for a developer to deploy a cross-reality system − they must either implement a bespoke solution of their own or build upon the framework of previous bespoke solutions which may only be available to researchers in the same institution as they were created.

This project addresses this shortcoming by allowing future development and deployment of cross-reality systems to be undertaken faster and with more ease, by providing an abstracted implementation of many of the required underlying functionalities that allow the separate technologies to communicate and interoperate.

3.6 Summary

There are many existing virtual worlds which present immersive, three-dimensional online communities in which users interact with each other and interactive objects around them. There are frameworks for supporting augmented reality, such as ARToolKit [66] and for supporting augmented virtuality, such as UC Berkeley's Tele-Immersion [67]. WSN and actuator technologies have evolved to the state where they are widely available and supported with interfaces and APIs for a plethora of programming languages. All of these reasons have led to numerous implementations of cross-reality, some commercial but the majority in academia.

However there has been little work conducted on bringing together a formal framework with these existing technologies to allow for easy development of cross-reality systems that leverage WSNs, actuators and virtual worlds. Implementations up to now have been bespoke, or based on previous bespoke work. MPEG-V represents the best effort so far for addressing this shortcoming, however it is still in its development stages.

This project addresses the shortcoming now by presenting a toolkit that allows WSNs, actuators and virtual worlds to be easily combined into a single cross-reality system, by presenting existing technologies with standardised interfaces for interaction with each other, abstracting over the underlying functionality needed for them to communicate and interoperate. This is a partial implementation of MPEG-V and supports the arguments for why such a standard is required for cross-reality research and development to progress.

Chapter 4

Requirements Specification

This chapter formally defines the properties that the software solution must have and adopts the form of a *requirements specification document* based on the outline and description provided by Sommerville in 'Software Engineering' 5th edition [68].

Requirements are divided amongst *user requirements* and *system requirements*. User requirements are statements in natural language of the services the system must provide and constraints on its operation. System requirements define what must be implemented by forming a detailed description of the system's functions, services and operational constraints.

Requirements are further classified as either *functional* or *non functional*. Functional requirements are statements of services that the system should provide, how it should react to particular inputs and how it should behave in particular situations. Non functional requirements are constraints on the services &/or functions that the system offers and can include timing constraints, constraints on development processes and methodologies, standards, etc.

4.1 Preface

This document serves as an outline of the project to create a toolkit facilitating development of cross-reality systems leveraging WSNs, actuators and virtual worlds, to be used by any interested developer in the field of cross-reality research and development.

4.2 Glossary

- Software toolkit − a collection of software utilities, frameworks and deployment models to facilitate easier and faster development when working with particular systems and technologies by abstracting lower level functionality, connectivity and logic behind pre-defined developer interfaces.

- Virtual worlds − a genre of online community in which users can interact with each other and with objects and the virtual environment around them. Today's virtual worlds usually comprise immersive, three-dimensional environments. [1] (See 3.1)

- Wireless Sensor Network (WSN) − *...low-cost, low-power, multifunctional sensor nodes that are small in size and communicate untethered in short distances... ...which consist of sensing, data processing, and communicating components...* [28] (See 3.2)

- Actuator − mechanical device that moves or controls a mechanism or system, usually by electrical, hydraulic or pneumatic energy [36]. Examples include electric motors and solenoids. (See 3.3)

- Cross-reality − an arrangement wherein there is synchronous, multi-directional informational or media exchange between real and virtual worlds [4]. (See 3.4)

4.3 Introduction

The scope of this document is a high-level overview of the features inherent in the core of the project. These are liable to change due to clarifications of requirements and introduction of new requirements at later dates.

The result of the project is a software toolkit that any developer wishing to create a cross-reality system using WSNs, actuators and virtual worlds can make use of to more easily achieve their aims, by making use of logically presented interfaces that abstract over lower level connectivities and mappings between the different hardware and software platforms and architectures involved.

4.4 User Requirements Definition

The 'users' of this project are in fact developers themselves: developers interested in creating cross-reality systems leveraging WSNs, actuators and virtual worlds. Therefore it is presumed that the users have a fair amount of domain specific technical knowledge and this is reflected in the relatively high technical level of the user requirements.

Requirements of the system as identified by analysis and extrapolation of objectives (see Chapter 2) are as follows;

- Intuitive − the toolkit should present logically designed, intuitive and easy to use interfaces to the underlying implementation of connectivity and mapping between WSN and virtual world and between virtual world and actuators.

- Extensive − the toolkit should allow for the easy integration and interoperation of different WSN, actuator and virtual world platforms; it should

not be artificially restricted to any one particular sensor node vendor, virtual world server software, actuator platform, etc.

- Responsive − the time between a sensed physical or environmental event in the real world being reported to the appropriate interface by a WSN to the time that this information is made available for the virtual world to act upon should be small. Likewise the time between a command in a virtual world being executed and presented to the appropriate interface to the time that this command is made available for an actuator to act upon should be small.

- Easily deployed − the toolkit should be easy for a developer to deploy into their development environment and not require them to make substantial changes to this environment (for example, through the installation of many new software packages) nor to their approach and methodology towards development.

4.5 System Architecture

The core functionality of the system is in providing connectivity and mapping, abstracted behind intuitive interfaces, for;

- WSNs to communicate with a virtual world to allow for data collected by sensor nodes to be acted upon by a virtual world simulation to create an augmented virtuality situation,

- Virtual worlds to communicate with one or more actuators to allow for commands from a virtual world to augment reality via control of one or more of these actuators.

A basic overview of the architecture is shown in the architectural model figure 4.1 in section 4.7, System Models. The connection labelled **A** represents communication from a WSN to a virtual world whilst the connection labelled **B** represents communication from the same virtual world to one or more actuators.

4.6 System Requirements Specification

What follows is the formal specification of the system's requirements, categorised between functional and non-functional.

4.6.1 Functional requirements

1. The system must facilitate communication from a WSN to a virtual world via an interface.

2. The system must facilitate communication from a virtual world to one or more actuators via an interface.

3. The system must facilitate 1 and 2 simultaneously.

4.6.2 Non-Functional requirements

1. The system must not be restricted to any particular WSN, actuator or virtual world platform, vendor, model, etc.
   (a) The system must be developed in such a way that it presents interfaces that allow, within reason, any WSN, actuator or virtual world platform, vendor or model to be used, by presenting well defined interfaces and making use of common communications protocols.
2. Interfaces and communication protocols should make use of widely available technologies and standards where possible and not rely on proprietary or commercial solutions to enable communication between WSN and virtual world, or between virtual world and actuator.
   (a) Where possible open standards and open source software should be used.
3. Interfaces should be simple, intuitive, coherent and easy to implement.
4. Communications protocols should be simple, efficient and well documented.
5. The system must be as easy as possible to deploy.
   (a) The system should not require substantial changes to an existing development environment.
      i. The system should not require many new software packages to be installed.
   (b) The system should not require a long learning/training period for a developer who is familiar with the underlying technologies, of WSNs, actuators and virtual worlds, to become a proficient user.
6. The system must operate as responsively as possible.
   (a) There must be minimal lag between an event being reported to the appropriate interface by a WSN and that information being made available to the virtual world.
   (b) There must be minimal lag between an actuator command being reported to the appropriate interface by a virtual world and that command being made available to the appropriate actuator to act upon.

4.7 System Models

Figure 4.1 is the aforementioned architectural model that provides an overview of the system architecture.

Figure 4.2 is a data flow diagram that shows, from a high level perspective, how data is processed in the system. The WSN takes readings of physical and environmental conditions such as light intensity, temperature, humidity, etc. and sends these readings to the virtual world. Before these readings reach the virtual world server they are processed by some intermediary processing so that what the virtual world server receives are nicely formatted readings that conform to some predetermined standard, rather than raw readings straight from the sensors which could be of any number of different formats depending on what type of sensor took them. Commands from the virtual world are sent from the virtual world server to the actuators; the virtual world sends generic commands that conform to a predetermined standard, which are then processed by intermediary processing into commands for the particular actuator platform that is the target of the commands. In this fashion the virtual world server does not need to know what specific actuator platform is the target of the commands and can simply send in a generic standard.

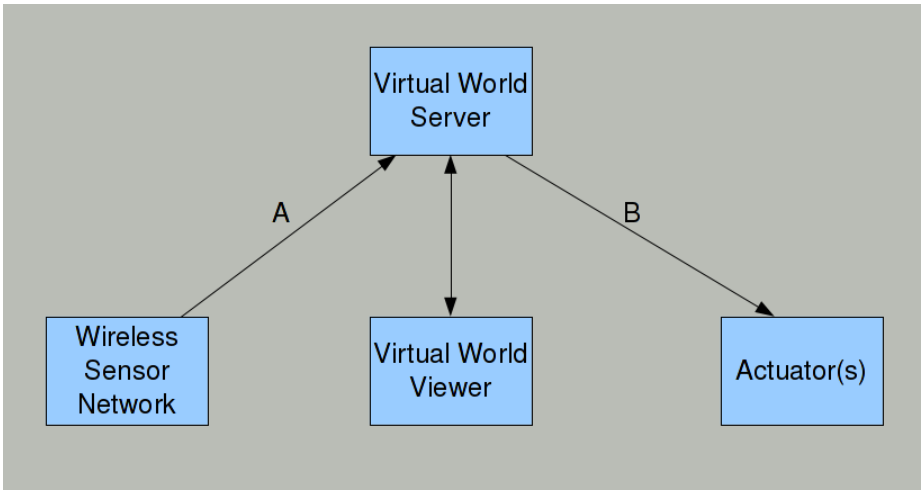

Figure 4.1: Architectural model

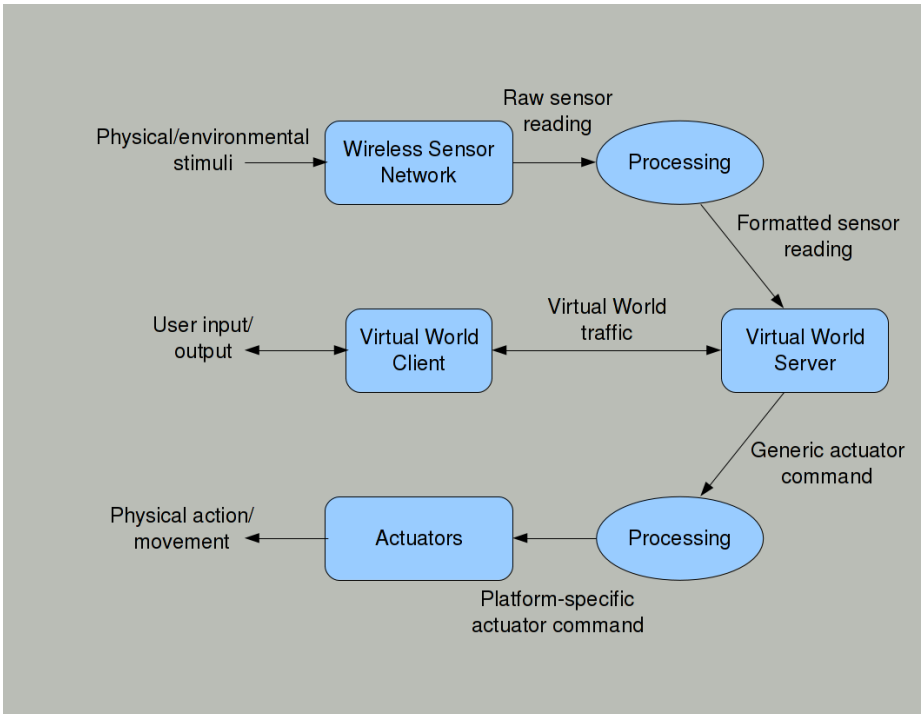

Figure 4.2: Data flow diagram

4.8 System Evolution

By making use of well defined interfaces to underlying logic, changes to the toolkit's functionality are possible without a notable impact on a user's interaction with it; as long as the interfaces themselves remain unchanged, the implementation of the computation and logic beneath can change.

By adopting a good development strategy of keeping the implementation as 'modular' as possible, changes to one part of the system do not have a grave impact on other parts of the system.

By making use of open source software that is freely available for many platforms and is accepted (even preferred) by the developer community, longevity of the system is ensured.

4.9 Summary

This chapter has formally defined the properties that the software solution must have, in the form of a requirements specification document. The requirements are divided into user requirements, which are statements in natural language of the services the system must provide and constraints on its operation, and sys-

tem requirements, which form a detailed description of the system's functions, services and operational constraints. Requirements are further categorised as either functional or non functional.

System models are provided that illustrate the architecture of the system and how data is processed in the system.

Chapter 5

Software Engineering Process

This chapter describes the software engineering approaches adopted for development of the project. Whilst it would be unrealistic to follow a recognised formal approach to the letter, relations can be made between the approach chosen and formal and semi-formal models of software engineering, along with justification for these choices and for diversions from these formal processes.

5.1 Overview

Because of a lack of prior experience working with WSNs, actuators and virtual worlds, it is necessary to spend considerable time during the early stages of the project attaining proficiency with these technologies through experimentation and by creating prototype software that is ultimately thrown away and does not feature as part of the final solution. This process allows the capabilities and restrictions of each technology to be ascertained, as well as providing first-hand experience of the development methodologies and practices associated with them, which are integral requirements for ensuring that the toolkit is implemented correctly.

After this learning process a more traditional waterfall approach is taken to continue development of the solution, once the required knowledge and experience to correctly design and implement it is under belt.

5.2 Throwaway Software Prototyping

When incremental delivery of a software product is not required, as is the case with this project where the entire codebase is submitted in one final deliverable, prototypes of the software can be produced to demonstrate concepts, try out design options and generally find out more about the problem and its possible

solutions. This is particularly relevant to the nature of this project, where several emergent technologies are combined together in a manner for which there is not a recognised or documented approach and for which there are many possible solutions.

Software prototypes can be used at various stages throughout the system's lifecycle; during the requirements engineering phase to help with elicitation and validation of system requirements, during the system design process to explore particular features, solutions and to support user interface design, and during the final testing phase where a prototype can be tested against the final deliverable to further confirm validity. [68] Prototypes can spawn ideas, reveal new requirements (as well as omissions in the original requirements) and highlight strengths and weaknesses in the design.

The Requirements Specification of this project (see Chapter 4) does not require in-depth knowledge of the capabilities of the technologies involved − the requirements are written at a higher level of abstraction. However at the design stage a better knowledge and understanding of the technologies is required before the design itself can progress.

Without investing considerable time into prototyping with the technologies involved, the design will be poorly executed and won't exploit the technologies to their utmost, or worse still will make assumptions about the availability of features that in reality are lacking and will cause problems during implementation.

Because of this the design stage is preceded by a substantial stage of software prototyping, with numerous prototypes created before design even begins, to attain enough insight into the technologies to allow for a successful design to be drawn up.

5.3 Visualisation with the Waterfall Method

The main development cycle of the system roughly follows that of the classical waterfall model of software development. Visualising the software prototyping stage as an addition to this model results in the arrangement shown in figure 5.1 where the requirements stage is separated from the design stage by a distinct stage of prototyping.

This prototyping stage is returned to many times during the design stage, as new requirements or desired features are discovered and further prototyping is required to ascertain whether the technologies support such features and how they can be designed and later implemented.

Some prototype code is inevitably present in the final solution itself, as indicated by the direct connection in figure 5.1 between the prototyping and implementation stages.

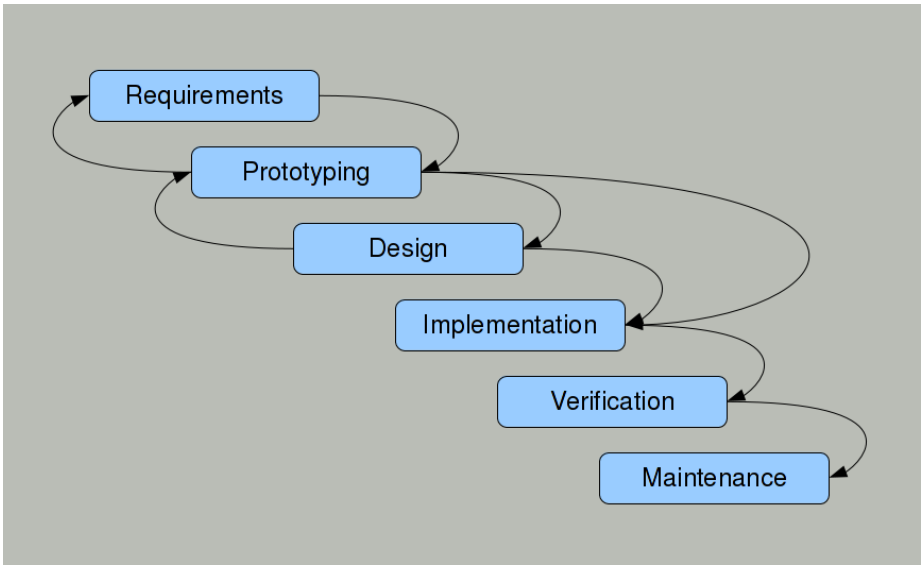

Figure 5.1: Modified waterfall model.

5.4 Visualisation with Gantt Chart

The software engineering approach can also be visualised with a Gantt chart as figure 5.2 which allows the separate stages of the project to be shown against the major deadlines. The row for prototyping has substantial 'slack' indicated by an arrow, as prototypes are ultimately used right through into the testing and verification of the system.

5.5 Summary

The software engineering process is largely dictated by the high learning content of the project with regards to the technologies involved. Without committing the time to becoming familiar with these technologies through throwaway prototyping the design and subsequent implementation directly suffer.

Although investing such a substantial amount of the overall time available to the project into writing code that is ultimately thrown away means that the project initially progresses slowly, it means that the implementation ultimately progresses more fluidly and the final solution is of much higher quality.

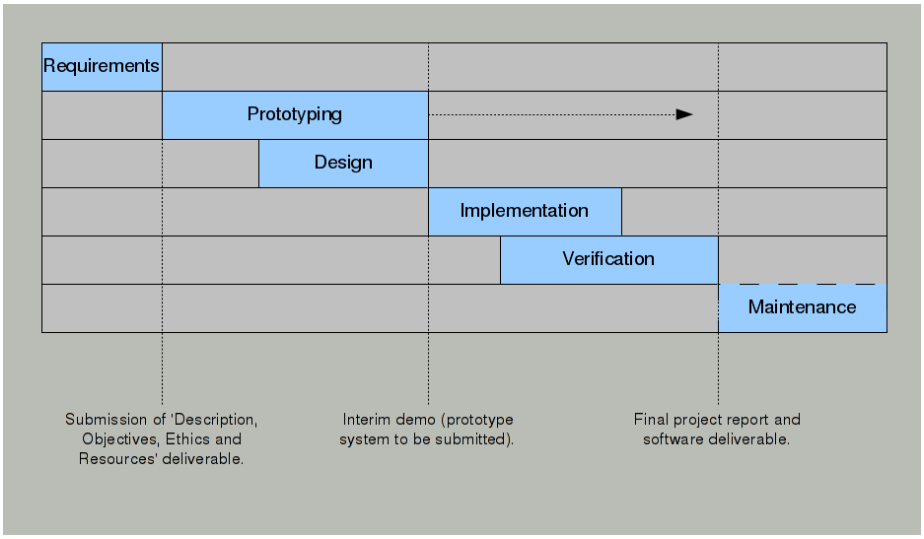

Figure 5.2: Gannt chart of software engineering approach.

Chapter 6

Ethics

The product of this project is a software toolkit that facilitates development of cross-reality systems leveraging WSNs, actuators and virtual worlds. This toolkit is used in the creation of an example use case implementation of a cross-reality system.

Whilst a cross-reality system could easily involve people, unbeknownst to them, the demonstration implemented here is in a controlled environment into which unwitting individuals cannot stumble and thus unintentionally and unwillingly become part of the system. Even for those that were to participate in the cross-reality system, no personal data is collected nor stored by the system.

If the scope of the demonstration were to be increased, there would be ethical considerations to mull over possible interactions between individuals and the system, however in the restricted demonstration environment that is in effect there are no concerns.

Chapter 7

Design

This chapter indicates the structure of the system, presenting the design itself along with decisions made and justifications for them, with discussion of available alternatives and why they are not ultimately used. Figure 7.1 shows the architecture of the final design and what follows is a decomposition of this architecture with discussion of each constituent component.

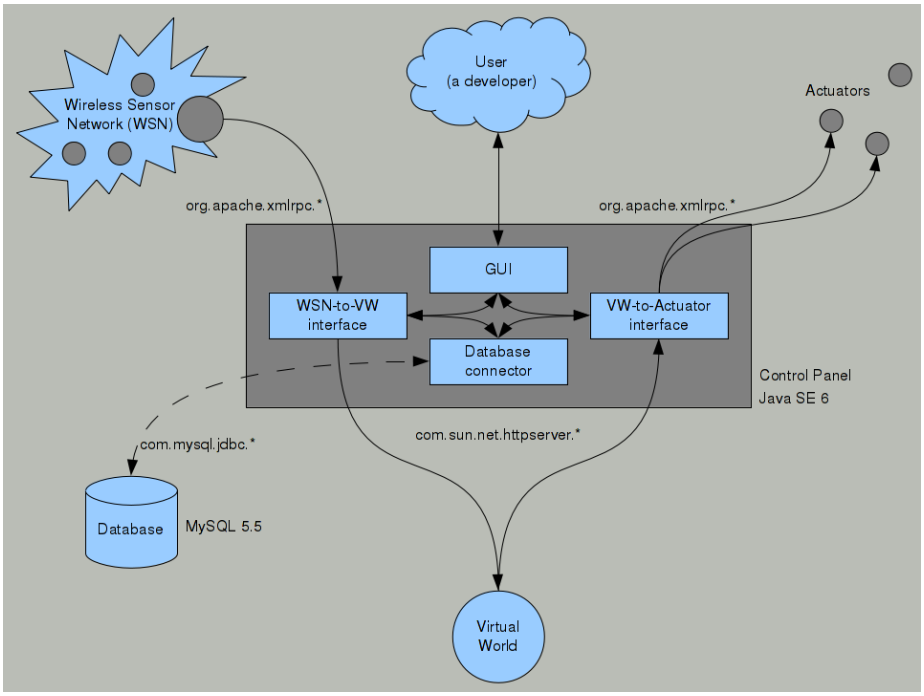

Figure 7.1: The architecture of the final implemented design of the toolkit.

7.1 Overall System Architecture

The overall architecture of the system from a more high-level perspective than figure 7.1 is depicted in figure 7.2 and shows how the system is split into its main logical components and how these are distributed, along with the connectivity between them.

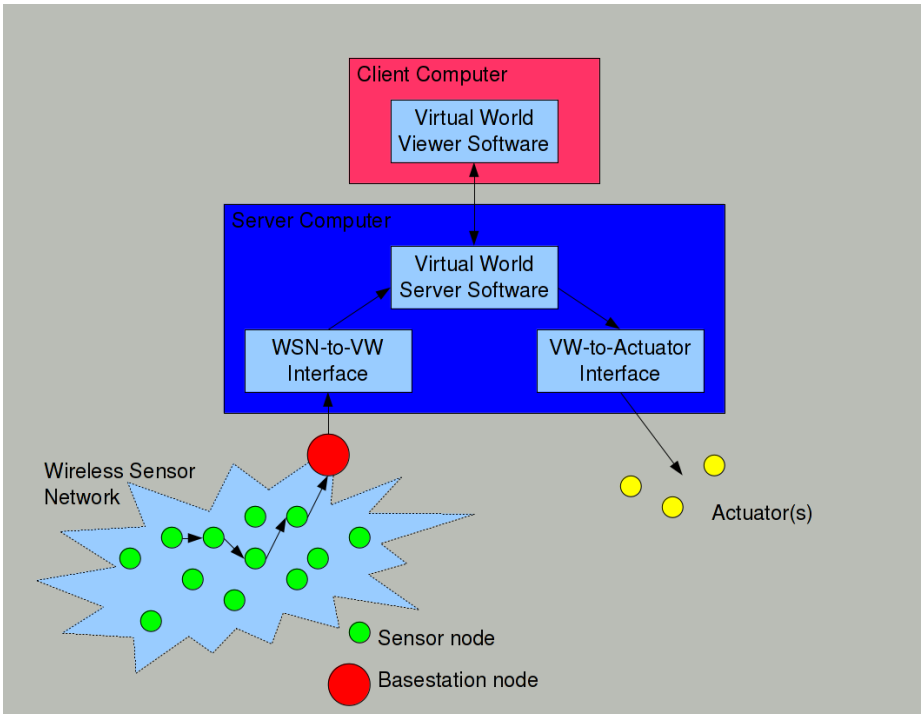

Figure 7.2: Overall system design

The system is composed of 4 main components;

1. A WSN. This consists of;
    - Exactly 1 *basestation node*. This node is connected physically to a computer, most likely via a wired serial or USB connection. This node is responsible for receiving communications on its short-range radio interface from the *sensor nodes* that make up the rest of the WSN and subsequently passing these up to the computer that it is connected to.

- At least 1, but probably more, *sensor nodes*. These are not connected physically to any computer. They are responsible for using their sensors to detect physical and environmental conditions and to communicate these to the basestation node by making use of their short-range radios to wirelessly forward the traffic amongst themselves until it reaches the basestation node.

  This is one of the most common operating paradigms for a WSN, with many physically unconnected nodes forwarding sensor readings on to a single connected basestation or 'gateway' node which can then provide these to the larger computer infrastructure.

2. One or more actuators. The concept of an actuator is quite vague and high-level, covering a great many possible incarnations and connectivity options; there are many actuator interfaces that connect to a computer via a serial or USB connection and allow the computer to control one or more electric motors, solenoids, etc.. The specifics of these connections are outwith the scope of the discussion at this point.

3. A computer that acts as *server*. A virtual world requires a piece of virtual world server software to be run, such as the Second Life server, or OpenSimulator, so the assumption of the existing presence of such a computer in a system that wishes to make use of the toolkit is fair.

4. A second computer that acts as *client*. This computer is responsible for running a piece of virtual world client software, referred to as a 'viewer', such as the official Second Life Viewer or the Hippo OpenSim Viewer.

It is possible for the server computer and the client computer to be the same machine, however the usual practise of running a virtual world, particularly one where multiple users are connected simultaneously, is to have a dedicated server computer that does not simultaneously run virtual world viewer software to connect to itself.

It is also *expected* for there to be more than one client computer connected to the server computer, however for the sake of diagrammatic simplicity figure 7.2 depicts just one such client computer. Each of these client computers runs its own instance of a piece of virtual world viewer software, with each possibly using a different viewer as some or all of the others or using a different version of the same viewer as some or all of the others, all connecting to the same single server computer where the virtual world server software is executing.

The design of the toolkit does not hamper the ability of a virtual world server to support multiple simultaneous connected clients, nor does the situation where a server has multiple connected clients affect the operation of the toolkit.

7.2 Introduction to Interfaces

The implementation is focussed around the 2 interfaces, shown in figure 7.2 as residing on the server computer, that sit between the virtual world server

software and the WSN and the actuators.

7.2.1 Distributed Nature of the Interfaces

In reality it will not always be convenient to host these interfaces on the same physical server computer as the virtual world server software is executing upon. Particularly if the virtual world server software is executing on a remote machine out of the jurisdiction of the developer making use of the toolkit, she may not have the required permissions to deploy the interfaces onto this computer. In this situation there are no architectural nor technical reasons within this design for why the interfaces cannot run on an intermediary computer separate to the remote computer upon which the virtual world server software is executing. This situation is depicted in figure 7.3 which presents a simplified subset of the components contained in figure 7.2 but where the interfaces are hosted on an intermediary server computer between the WSN and actuators and the server computer that the virtual world server software is executing upon.

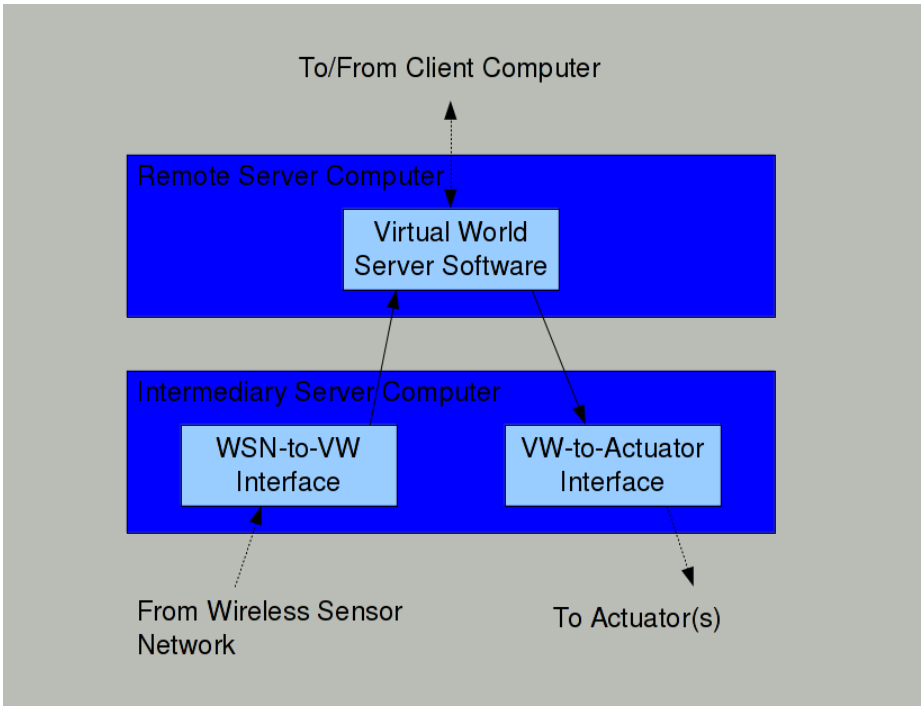

Figure 7.3: Intermediary server architecture

7.2.2 Connectivity of the WSN Basestation Node &/or Actuator Controller

In figure 7.2 the WSN and actuators are depicted as separate discrete entities that communicate directly with the server computer upon which their interfaces are hosted. Similarly in figure 7.3 the WSN and actuators are shown as communicating directly with the intermediary server computer.

As has been discussed previously the basestation node for the WSN and conceivably some form of control interface for the actuators (referred to as the actuators *controller* to avoid confusion with *interface* used elsewhere in this chapter) are most likely connected directly to a computer via a wired serial or USB connection. Naturally having this WSN basestation node &/or actuator controller connected directly to the server computer or intermediary server computer is seldom desirable. These server computers are likely to be remote machines locked in a data centre and this is unlikely to be a location where one would want to deploy a WSN or setup actuators.

Because of this the design allows either or both of the WSN basestation node and the actuator controller to be connected directly to a computer or computers remote to the server computer(s), but at the same time maintains the ability to connect them directly to the server computer(s) as it is possible in a research environment for it to be desirable to deploy the entire system upon a single testbed machine on a researcher's desk. It is also possible to have the WSN basestation connected to one computer and the actuator controller to a different computer, for maximum deployment flexibility.

7.2.3 Example WSN Basestation Node and Actuator Controller Connectivities

Different possible configurations of WSN basestation node and actuator controller connectivity that the design supports are depicted by figures 7.5 through 7.9, with figure 7.4 presenting the key to these diagrams. Please note that this is not the exhaustive set of possible situations that are supported, but an example of the flexibility that is supported in terms of heterogeneous deployment architectures.

Single Server Computer

The situation of a single server computer as in figure 7.2 with both the WSN basestation node and the actuator controller directly connected to this single server computer is shown by figure 7.5. This arrangement is conceived of a development environment within a researcher's lab, perhaps using a single desktop computer with both the WSN basestation node and actuator controller directly connected to it and also running the virtual world server software, perhaps with a separate laptop computer running the virtual world viewer software.

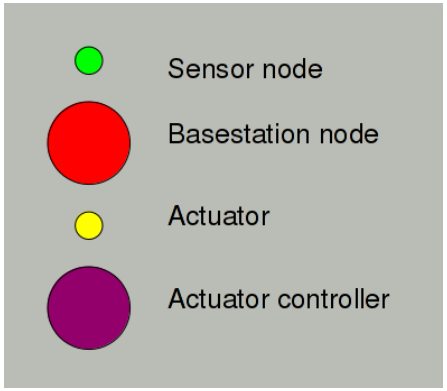

Figure 7.4: Key for connectivity diagrams 7.5 to 7.9.

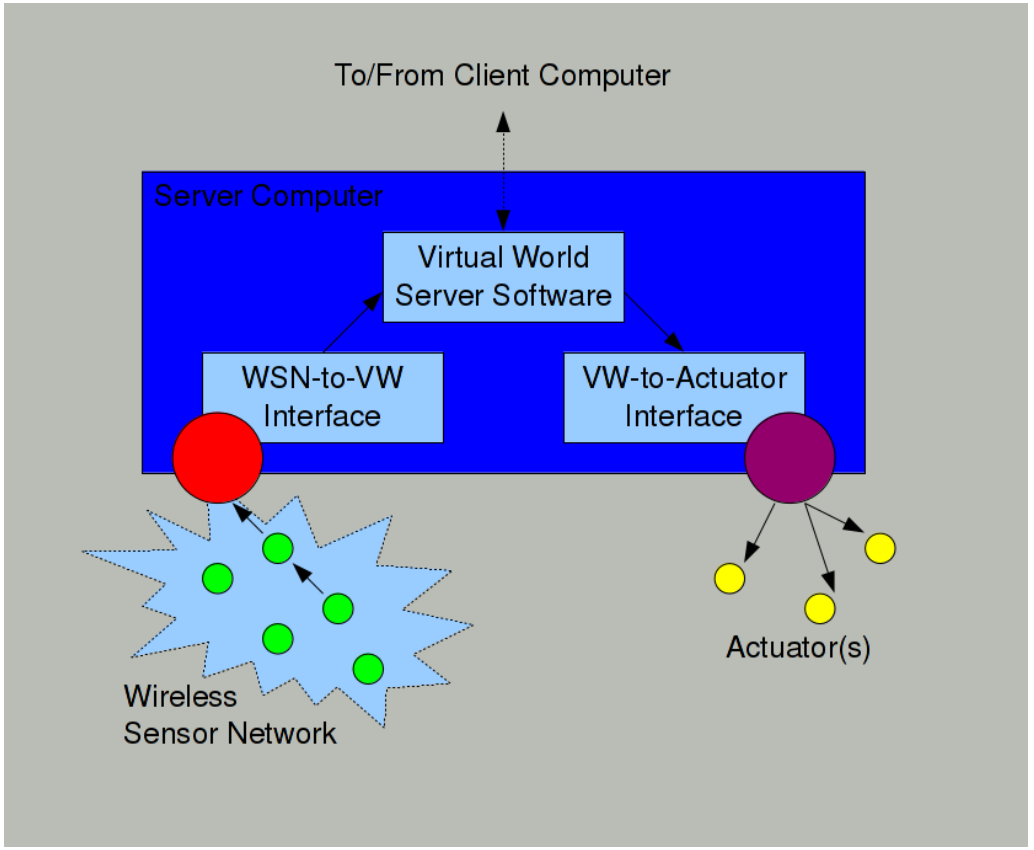

Figure 7.5: Single server with direct connection of WSN basesation node and actuator controller to server computer.

Remote and Intermediary Server Computers

An alternative situation of a separate remote server computer and intermediary server computer as in figure 7.3 with both the WSN basestation node and the actuator controller directly connected to this intermediary server computer is shown in figure 7.6. This arrangement is conceived of a deployment where the developer has restricted access rights to the server computer upon which the virtual world server software is executing, so deploys an intermediary server more under her own control to deploy the interfaces onto and where the location of the intermediary server is amiable for the deployment of WSN and actuators. This could be the result of a researcher making use of an existing virtual world server in their department that is situated in a machine room or otherwise out of their control, so deploying the rest of the system onto an intermediary server on their desk, benefiting by not having to run her own virtual world server software on this intermediary server.

Single Server Computer, Separate non-server Computer

Another situation, with a single server computer as seen in figure 7.2 but with both the WSN basestation node and the actuator controller connected to a computer separate from the server computer is shown by figure 7.7. This arrangement is likely where the developer has permissions to deploy the interfaces upon the same server computer as the virtual world server software is executing upon but where this server computer is not in a convenient position for deployment of the WSN or actuators, so makes use of a separate computer situated in a more convenient position to directly connect the WSN basestation node and actuator controller to.

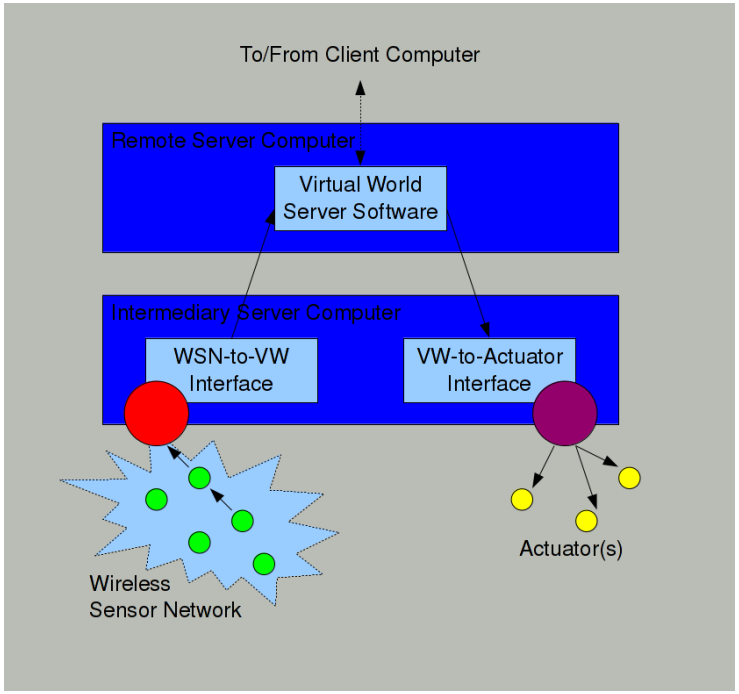

Figure 7.6: Intermediary server with direct connection of WSN basesation node and actuator controller.

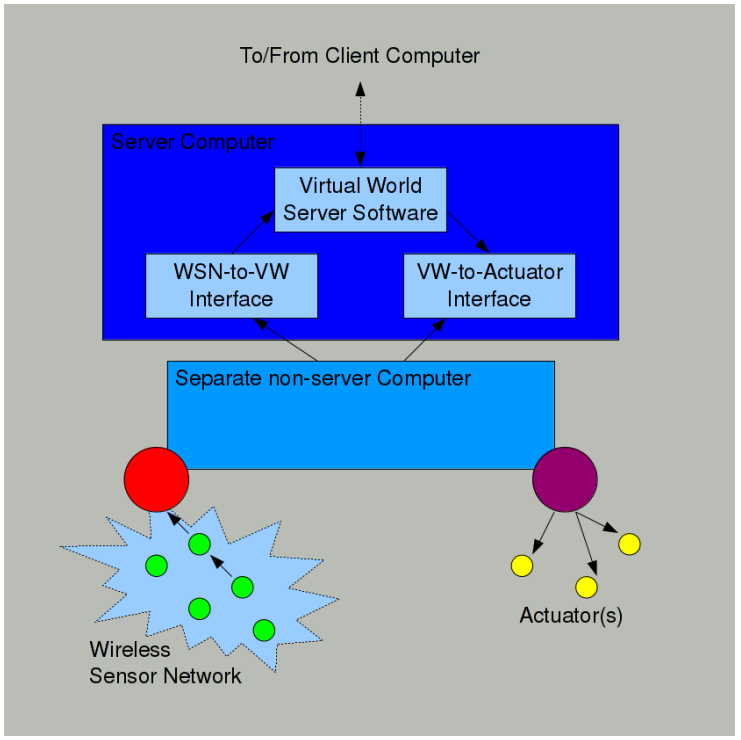

Figure 7.7: Single server with direct attachment of WSN basesation node and actuator controller to a computer *separate* from the server computer.

Remote and Intermediary Server Computer and Separate non-server Computer

A more exotic situation that is still supported even though it is less likely, of the separate remote server computer and intermediary server computer as in figure 7.3 but with the WSN basestation node connected to the intermediary server computer whilst the actuator controller is connected to a computer separate from both server computers as was introduced in figure 7.7, is shown by figure 7.8.

Single Computer for both Client and Server

Finally figure 7.9 shows the simplest client/server architecture where a single computer acts as both client and server; running the virtual world server software and the virtual world viewer software, hosting the interfaces and also connecting directly to both the WSN basestation node and the actuator controller. This is the simplest arrangement for a researcher to test or develop with the system, as it requires only a single physical computer. This architecture is used by the early software prototypes of the toolkit's development, before progressing onto the more complex architectures described in this section.

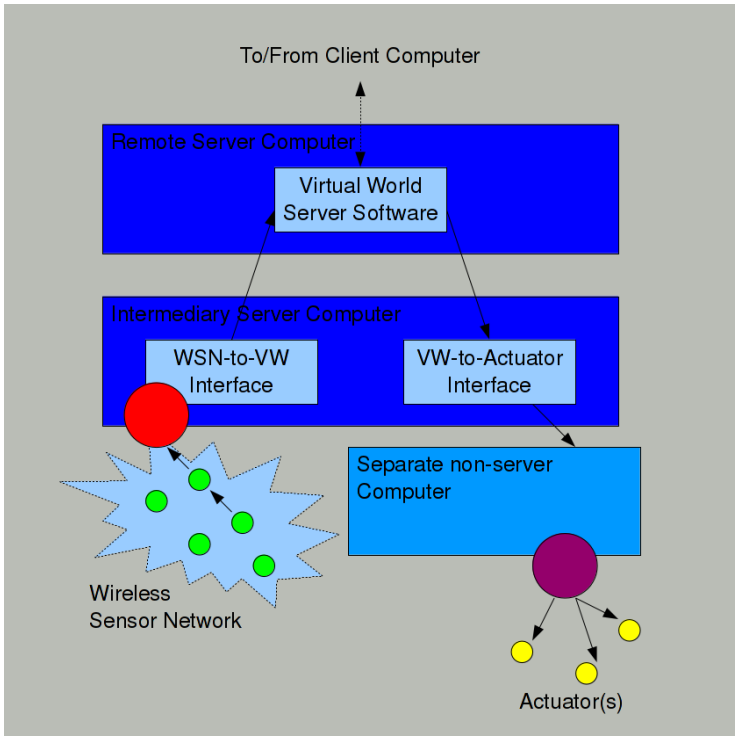

Figure 7.8: Intermediary server with direct attachment of WSN basestation node to the intermediary server, but the actuator controller to a computer separate from both servers.

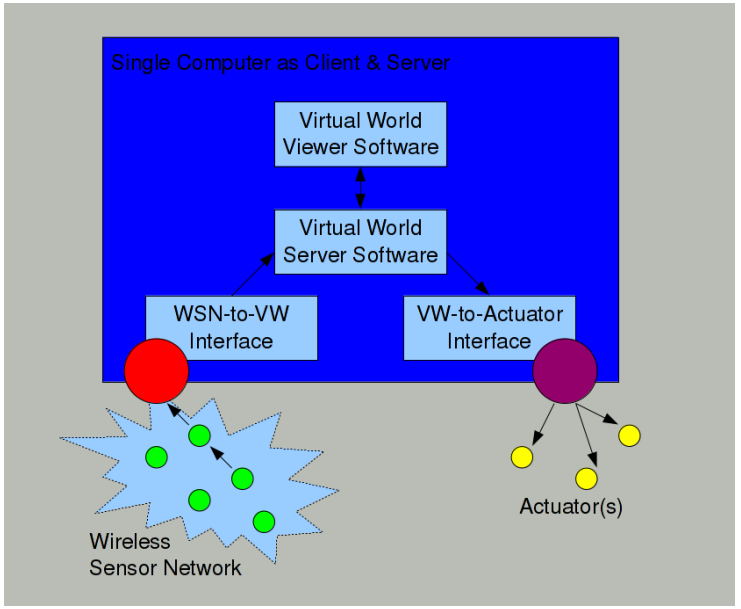

Figure 7.9: Single machine acts as both client and server, with direct attachment of WSN basestation node and actuator controller.

7.2.4 Directivity of Communication to/from Interfaces

As the directivity of the arrows in figure 7.2 between the interfaces and the virtual world and WSN/actuators indicates, these interfaces are simplex, facilitating communication only in one direction. For the WSN to virtual world interface the direction of communication is from the WSN to the virtual world and for the virtual world to actuator interface the direction of communication is from the virtual world to the actuators. One can think of the WSN providing 'input' into the virtual world and the virtual world providing 'output' to the actuators. When both of these communication channels are active simultaneously, so that there is information being communicated from the WSN to the virtual world through the WSN to virtual world interface *at the same time* as there is information being communicated from the virtual world to the actuators via the virtual world to actuators interface, a cross-reality environment is created.

Naturally there is nothing in the toolkit that actively prevents a non cross-reality environment from operating. For example, if a developer wanted to use just the WSN to virtual world communication and not the virtual world to actuators communication, the toolkit does not seek to prevent her from doing so.

7.2.5 Summary of Distributivity

What sections 7.2.1 to 7.2.4 identify is that it is of vital importance to the success of the toolkit for the individual components, particularly the interfaces between WSN and virtual world and between virtual world and actuators, but also for the connectivity of the WSN basestation node and the actuator controller to their host computer(s), to be as flexible as possible. A developer should be able to deploy the interfaces to any computer that they see as appropriate and likewise be able to connect the WSN basestation node &/or the actuator controller to any computer they see as fit.

This requirement is due to the inherent heterogeneity of different development and deployment environments. If the toolkit were to require a specific arrangement of client &/or server computers, or place restrictions on which computers the WSN basestation node &/or actuator controller can be directly connected to, it would prove near useless as the likelihood that an interested developer either already has such an environment set up or would be willing to set one up in exactly this fashion is slim to none.

7.3 Control Panel

It is apparent considering the design of the interfaces that simply creating them without some sort of intuitive interface to control and manage them will present any user with an administration and configuration nightmare. Thus a *Control Panel* is needed to help users with this administration of the interfaces. The architecture of the Control Panel in relation to the two interfaces is shown in figure 7.10. The Control Panel contains or encapsulates both of the interfaces,

providing a single application that can be deployed on to the server computer; this makes deployment simpler than if the Control Panel were a discrete entity to the 2 interfaces, as deployment would then involve 3 separate applications.

7.3.1 Restriction of Control Panel Design

The design decision of encapsulating the interfaces within the single Control Panel entity does mean that the two interfaces cannot be deployed onto separate server computers, however with the supported distributivity of connections for the WSN basestation node and actuator controller, it should not pose any problems having both interfaces restricted to the same logical server computer, as this computer should be accessible from any computer or computers that the WSN basestation node and actuator controller are connected to.

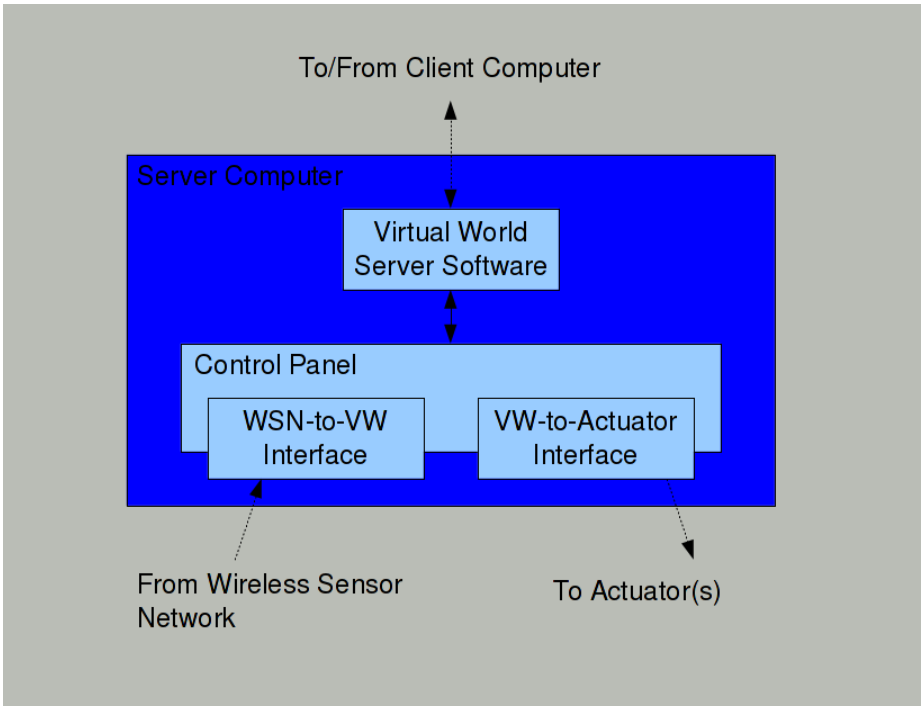

Figure 7.10: Architecture of Control Panel in relation to the two interfaces.

7.3.2 Features of the Control Panel

From a high-level perspective, the Control Panel acts as a central management utility that provides the user with the ability to;

1. Define and manage communications between a WSN and a virtual world, through control of the WSN to virtual world interface.

2. Define and manage communications between a virtual world and actuators, through control of the virtual world to actuators interface.

3. Define and manage connection to a database for maintaining information about connections between WSN nodes and actuators.

7.3.3 Design of the Control Panel

A command line interface (CLI) would not be sufficient to achieve all of the requirements from 7.3.2 in an intuitive and efficient manner. One possibility is to implement the Control Panel within the virtual world itself, however as the toolkit is supposed to be able to operate in a manner independent of any particular virtual world software it would present an incredibly difficult, or more likely impossible, task to implement such a complex application that could be deployed in any virtual world environment.

Another possibility is to implement the Control Panel as some sort of web application. This has the benefit that the Control Panel would be easily accessible from within a Web browser no matter where the server is located and would not require the server to have any graphical desktop environment. Furthermore, a server that is already running a piece of virtual world server software probably has a well-known URI so accessing this web application would present minimal problems if the Control Panel were deployed upon the same computer.

However the complexity of the Control Panel's functionality and in particular the amount of state that it has to maintain about the system as a whole, particularly in a deployment where there is a WSN comprised of hundreds of sensing nodes all bombarding the WSN-to-VW interface with readings, means that a traditional standalone GUI application written in Java is a better solution.

7.3.4 Problem of a Standalone Application Control Panel

Developing the Control Panel as a standalone application may present a problem for deployment as the server that a developer wishes to use to host the Control Panel may not have a graphical desktop environment installed or running, particularly if they wish to deploy it to an existing virtual world server that is running headless in a machine room. There are two solutions to address this issue, should it occur;

1. Make use of some sort of graphical desktop sharing application, such as Microsoft's Remote Desktop Protocol (RDP), Virtual Network Computing (VNC) or X session forwarding over Secure Shell (SSH).

2. Probably a somewhat better solution, is to later port the standalone desktop application in Java to a Java applet or Java servlet using Apache

Tomcat, such that it can be hosted on a headless server and accessed by a client via a Web browser.

Research situations, such as that depicted in figure 7.9 where a graphical desktop environment is available locally, are the most likely situations for the toolkit to actually be initially used within, until the field of cross-reality research involving virtual worlds, WSNs and actuators matures toward including more real-world deployment scenarios. As such it is not a notable detraction from the overall quality and success of the toolkit that the Control Panel is implemented as a standalone application.

7.4 WSN to Virtual World Interface

The WSN to Virtual World interface (referred to as WSN-to-VW interface from here for the sake of brevity) is responsible for presenting a WSN with a method of reporting sensor readings to a virtual world.

More accurately, or at a lower level of abstraction, the WSN-to-VW interface actually presents the WSN with a method of reporting readings to the *Control Panel* and the Control Panel then decides how to handle these reports, forwarding them on to the virtual world, possibly in a different format or by a different method, where it sees fit to do so. Naturally this two tier arrangement is invisible to the WSN itself.

7.4.1 WSN-to-VW Interface Protocol

Because of the distributivity of the system as identified in 7.2.1 the WSN basestation node is not necessarily directly connected to the same physical computer as the WSN-to-VW interface is executing upon. This means that the interface must be accessible and invokable remotely over a network, with the most obvious solution being some sort of Remote Procedure Call (RPC) protocol.

Initial prototypes made use of HTTP POST as a quick and easy way to deliver the sensor readings to the Control Panel, due to the ease with which HTTP clients and servers can quickly be set up in many high-level programming languages, including Java. However HTTP POST does not represent the most suitable RPC methodology; indeed, POST is not really designed to be used for RPC at all, but rather for uploading a file or submitting form contents to a web page, so using it as the delivery protocol for sensor readings from the WSN to the Control Panel is somewhat an abuse of its functionality.

Instead XML-RPC is used to report sensor readings from the WSN to the Control Panel. The justification behind this choice is severalfold;

- There are XML-RPC libraries available for many languages, including Java, C/C++, Perl, Python and a whole myriad of others, which means that developers are not overly restricted in what languages they can easily make use of to relay sensor readings from a WSN (most likely via direct communication with a basestation node) to the Control Panel.

- XML-RPC uses XML for encoding and HTTP as the transport, which makes it very simple and accessible to extend and to debug compared to other more proprietary protocols. Any developer familiar with even just basic HTML should be able to look at an XML-RPC request and be able to figure out what is going on.
- Should a developer wish to use a language that does not already have an XML-RPC client implementation, the XML-RPC specification is only around 7 pages in length and very easy to understand and simple for any competent programmer to implement relatively quickly.

SOAP is overly complex for the simple task at hand, that is basically to just report sensor readings to the Control Panel and receive a response back informing the client of success or failure. SOAP requires the creation of WSDL service descriptions, choosing between multiple different encodings (where XML-RPC has just one) and also has a complex security model when security is not of huge concern to this application at this time.

7.4.2 WSN-to-VW Interface Procedure Declaration

The necessary information for a report from the WSN to the Control Panel is quite limited;

- The source or 'id' of the reading. This is a string identifier for which sensing node in the WSN this particular reading comes from. This must be unique amongst nodes of the same WSN.
- The 'type' of sensor reading. Many WSN platforms provide nodes that host multiple different sensors, including light, temperature, humidity, etc. A distinction can be made here between 'node' and 'sensor', where a single node hosts one or more sensors which each senses a different physical attribute of the surrounding environment. Type must be unique amongst sensors on the same node.
- The value of the reading itself.

Thus the procedure declaration for reporting a sensor reading from the WSN to the Control Panel is;

```
Boolean ControlPanel.sensed(String id, String type, Integer
                            value)
```

From background research into WSNs it is safe to assume that the value returned by a particular sensor will always be able to be represented by a numeric type such as an integer. Most platforms report readings from their nodes' sensors as hexadecimal values of varying bit length depending upon the range and accuracy of the particular sensor.

The `Boolean` return type is used to inform the client (whatever software is sending the sensor reading report to the Control Panel) whether the request is successful or not.

7.4.3 Control Panel to Virtual World Protocol

7.4.1 and 7.4.2 detail the design of communication of sensor readings from the WSN itself to the Control Panel via the WSN-to-VW interface, however this is not the whole story of the transfer of information from WSN to virtual world. Once a reading has been received by the Control Panel (and an appropriate confirmation returned to the XML-RPC client that sent this report to indicate whether the report was successful or not) it is forwarded on to the virtual world (obviously at the discretion of the Control Panel, as the user may wish to disable certain nodes via the Control Panel rather than by physically powering them off, for example).

An obvious choice of protocol to achieve this forwarding is XML-RPC, however on further inspection this would not actually be a sensible decision.

One of the reasons that XML-RPC is a suitable design decision for the communication from WSN to the Control Panel is the wide availability of XML-RPC libraries for popular programming languages, such that an implementation of an XML-RPC client will most likely be available to a developer when they write an application to take readings from a WSN and send it to the WSN-to-VW interface, no matter what language they use. As the Control Panel is written in Java, there is a well regarded XML-RPC library from the Apache Web Services Project that provides an XML-RPC server for the Control Panel to make use of to receive these reports of readings. This Apache library also includes an XML-RPC client, so using XML-RPC to send the reading once received at the Control Panel to the virtual world would prove simple. However now that virtual world servers are beginning to become involved, it is necessary to check the support for XML-RPC in the popular virtual world server software that the toolkit has to interact with.

The two most popular virtual world servers for research of the type that the toolkit targets are Second Life and OpenSim. Neither has strong support for XML-RPC.

- Linden Lab, the creators of Second Life, have declared XML-RPC support deprecated due to the non-scalable implementation that their developers adopted − using a single server to act as an XML-RPC gateway for the entire service, which resulted in very poor performance as requests were queued at this server which could not handle the volume of requests. Linden Lab suggest that developers make use of HTTP in place of XML-RPC, as Second Life has a complete and stable set of HTTP functions for both server and client operations [69, 70].

- Because of the reduced adoption of XML-RPC in Second Life development compared to usage of HTTP, the OpenSim developers decided not to place much importance on the development of XML-RPC support in their server. Because of this, whilst OpenSimulator does have an implementation of XML-RPC, it is incomplete and not considered sensible for use over its complete implementation of HTTP functions for both server and client operations, where these are sufficient. (See Appendix F for a

log of discussion on the official OpenSim development IRC channel about this matter.)

Thus adopting XML-RPC as the communication technology for forwarding sensor readings from the Control Panel to the virtual world would not be a sensible decision, as it would rule out compatibility of the toolkit with Second Life and jeopardise stability with OpenSim, which together constitute the two potentially most popular virtual world platforms that the toolkit may actually be used with. There may be other virtual worlds that do provide usable implementations of XML-RPC, however it is not worthwhile to investigate this possibility when the two largest players have already ruled out of the usage of XML-RPC.

So instead of using XML-RPC for the communication of reported sensor readings from the Control Panel to the virtual world, the advice from Linden Lab is heeded and HTTP is used instead, making use of much of an aforementioned prototype that used HTTP based upon how easily it could be deployed. HTTP server and client functionality is included in a complete and stable form in both Second Life and OpenSim and because it is such a basic and fundamental protocol it is safe to expect most other virtual worlds to either also provide support for it, or to be capable of supporting it with minimal modification.

7.4.4 Control Panel to Virtual World Procedure Declaration

The format of the procedure definition for HTTP communications from the Control Panel to the virtual world is partially defined by the manner in which both Second Life and OpenSim, previously identified as the two largest potential virtual world platforms for usage of this project, handle HTTP interactions. Designing the toolkit's transport to maintain compatibility with these two platforms is important for its success.

Neither Second Life nor OpenSim are designed to predominantly operate as Web servers with well known URIs; this is simply not the point of a virtual world server! So the way in which they (both) handle support for HTTP client and server activity is by running a HTTP server on the same TCP port that the virtual world server itself listens on for viewer connections (port 9000 is the default for OpenSim) with `lslhttp` as the root folder, for example;

```
http://myvirtualworldserver.com:9000/lslhttp/
```

Individual objects ('primitives' in Second Life/OpenSim lingo) can request a unique address under this directory via a call to `llRequestURL()` in a script attached to the primitive, which is returned as a seemingly pseudorandomly generated string such as;

```
http://myvirtualworldserver.com:9000/lslhttp/
    8e30b7dd-4714-4836-885c-bc8e9181aec5/
```

By generating these strings randomly each time a primitive executes a call to `llRequestURL()`, the virtual world ensures that it cannot be easily used as a public forward facing Web server with well known URIs, as addresses like these are both difficult to remember and are subject to change each time the primitive's script is reloaded or the region containing the primitive is reloaded.

The important takeaway message here is that primitives in Second Life and OpenSim are *individually* addressable externally from the virtual world server. Thus the Control Panel can send sensor readings directly to the particular primitive within the virtual world that they are destined to affect. If primitives were not individually addressable externally from the virtual world server, such as if the virtual world server implemented HTTP functionality by providing a single HTTP gateway behind which communication to individual objects is abstracted and may not be HTTP at all, it would be necessary to implement some sort of functionality within the virtual world to receive HTTP requests at this single gateway and then disseminate them to the applicable objects.

Because of this the design for the procedure definition for forwarding reported sensor readings from the Control Panel to the virtual world is a HTTP POST with a body as such;

```
id=<id>&type=<type>&value=<value>
```

For Second Life and OpenSim, where primitives are individually addressable externally to the virtual world server, the `id` field is surplus to requirement as the request is delivered directly to the correct primitive, however for other virtual world servers which implement the alternative arrangement described above of a single HTTP gateway behind which access to individual object's is abstracted, the `id` field is required to uniquely identify which object the HTTP request is destined for. Upon receiving the request, functionality at the gateway can parse the body to extract the id and from this determine where to send the reading on to. This allows the toolkit to support a wider range of virtual world servers.

7.4.5 Data Flow Diagram of WSN-to-VW Interaction

The flow of data through the system when a sensor reading is taken and reported to the virtual world through the WSN-to-VW interface is shown in figure 7.11. This diagram assumes that the Control Panel decides to forward the reading on to the virtual world; there are situations where upon receiving a sensor reading report the Control Panel won't forward the reading, such as if the user has explicitly requested that readings from this particular node are not forwarded.

7.5 Virtual World to Actuator Interface

Similar to how the WSN-to-VW interface is identified in 7.4 as being split into two parts, the Virtual World to Actuator (VW-to-Actuator) interface similarly

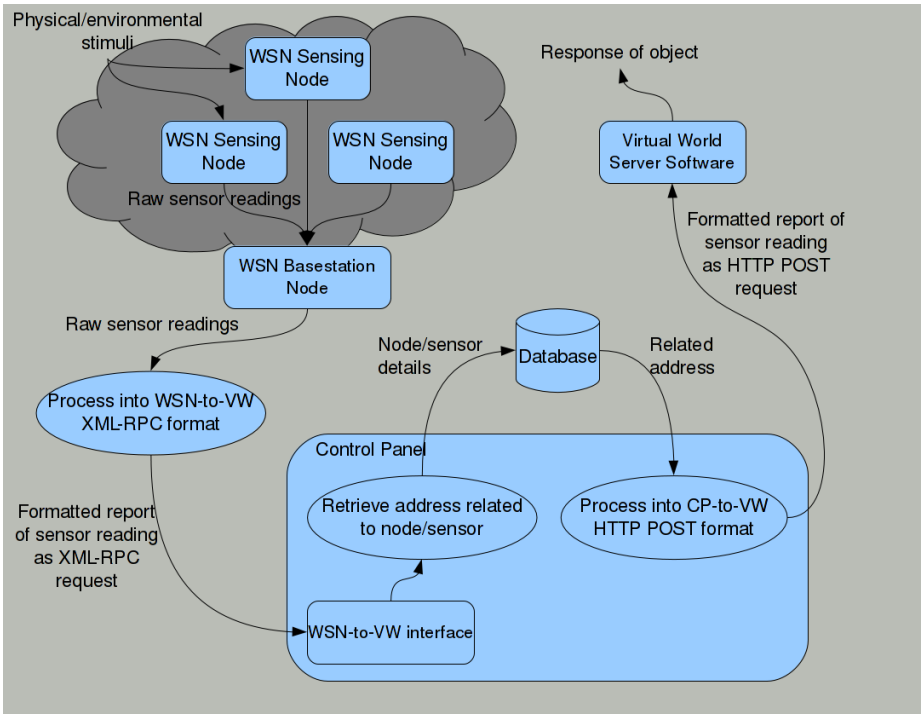

Figure 7.11: Data flow diagram of WSN-to-VW interaction.

presents the virtual world with a method of sending actuator commands to the Control Panel and the Control Panel then decides how to handle these reports, forwarding them on to the appropriate actuator controller, possibly in a different format or by a different method, where it sees fit to do so.

7.5.1 VW-to-Control Panel Protocol

The decision of what protocol to use for communication of actuator commands from the virtual world to the Control Panel is made much easier from what was learnt in 7.4.3 about the availability of protocols and supporting libraries and implementations when considering the virtual world server software. Similarly to how HTTP POST is used over XML-RPC for forwarding sensor readings from the Control Panel to the virtual world in 7.4.3, HTTP POST is likewise used to send actuator commands from objects in the virtual world to the Control Panel, as the status of the XML-RPC client provision of Second Life and OpenSim is no better than their XML-RPC server provision.

With regards to HTTP, in addition to facilitating a primitive with the ability to obtain an externally accessible URI and to easily handle incoming requests to this URI, Second Life and OpenSim also allow a primitive to easily send HTTP requests to URIs external to the virtual world. Thus a primitive that is

to control an actuator in the real world sends the relevant command information in the body of a HTTP POST request to the Control Panel.

7.5.2 VW-to-Control Panel Procedure Declaration

As has been previously discussed, the concept of an actuator is quite vague (see 7.1) so deciding upon the format of the HTTP POST body requires some thought if wide compatibility with various different actuator controllers is to be maintained.

One approach would be to try and define every possible field that a developer may wish to be contained within an actuator command and to include these fields in the POST body. Because of the vast array of possible types of actuator this would require fields for every conceivable type of movement (rotation, extension, etc.), speed and direction, temperature, switching (for relays), etc. and even if hundreds of such fields and data types are defined there would almost undoubtedly be some overlooked which would make the system less useful to any developer wishing to use actuators that require these particular fields. Also by restricting a developer to a fixed set of fields, extensibility of the project would be hindered as new unforeseen actuator types are released.

As such the best approach is to leave the POST body's contents as abstracted and high level as possible, adopting the simple form;

`id=<id>&action=<action>`

where `id` is a unique identifier of the actuator that this particular command is to affect and `action` is a string that can contain all manner of contents. This allows developers to deploy the system with their virtual world and actuators, making use of the VW-to-Actuator interface to abstract over the communication between the virtual world and the actuators, but at the same time allowing them to define their own control protocol to be included within the `action` field to control their particular actuator setup. The simplest example involves actuators that use simple relays to switch on &/or off electrical devices, where the contents of `<action>` could conceivably be as little as `on` or `off`, which combined with the `id` will be enough to switch electrical devices on and off.

7.5.3 Control Panel-to-Actuator Interface Protocol

Just as the decision of protocol in 7.5.1 was aided greatly by the lessons learned in 7.4.3, the decision of what protocol to use for forwarding actuator commands received by the Control Panel from the virtual world to the appropriate actuator controller is helped by those in 7.4.1, as these are essentially the same communications channels except sending in the opposite directions.

The arguments for selecting XML-RPC as the protocol to deliver readings from the wireless sensor network to the Control Panel stand just as well for justifying its use to forward actuator commands received by the Control Panel from the virtual world to the appropriate actuator controller. If the assumption is already made of the availability of XML-RPC libraries great enough that most

languages that a potential user may use to interface with their WSN basestation node will already have an XML-RPC client implementation, then making a similar assumption that most languages a user may use to interface with their actuator controller will already have an XML-RPC server implementation seems fair and logical.

7.5.4 Control Panel-to-Actuator Procedure Declaration

The format of the XML-RPC declaration is similar to that in 7.4.2, except that there is no equivalent for 'type' included, as it is presumed that this information is included within the `action` field.

```
Boolean control(String id, String action)
```

The `Boolean` return type is used to indicate to the client (the Control Panel) whether or not the request was successful.

7.5.5 Data Flow Diagram of VW-to-Actuator Interaction

The flow of data through the system when an avatar-induced action in the virtual world triggers an actuator command to be sent from the virtual world to the actuator through the VW-to-Actuator interface is shown in figure 7.12.

7.6 Control Panel Architecture Diagram with Transport

Figure 7.13 is an updated version of Figure 7.10 that shows the distribution and hierarchy of XML-RPC/HTTP clients and servers between the different components of the Control Panel and the virtual world server, WSN and actuators.

7.7 Database Design

A database is required to keep track of the relationships and state between the nodes (and sensors that these nodes host) as well as the actuators. If the Control Panel stores these relationships in memory instead, then they will either be lost when the Control Panel is exited, forcing the user to recreate all of these relationships the next time they execute the Control Panel even if it was in exactly the same situation as the last execution with the same nodes, sensors and actuators, or they will have to be written to file on exit and loaded again on the next execution.

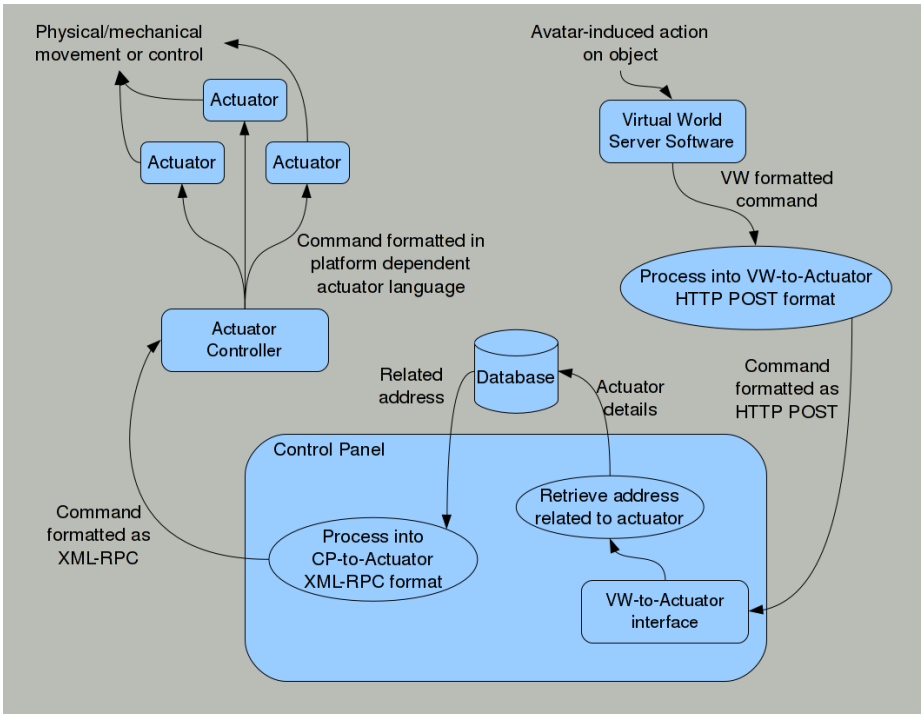

Figure 7.12: Data flow diagram of VW-to-Actuator interaction.

Whilst writing the relationships to file will maintain them between subsequent executions of the Control Panel, storing them in a database has benefits that a file-based approach does not.

For example, as discussed in 7.4.4 the URIs of primitives in Second Life and OpenSim are randomly generated each time the script that requests the URI is executed and are non human memorable. A developer could conceivably type these URIs into the Control Panel against the nodes that she wishes to affect the primitives, or maybe copy and paste them to save time, however a much better approach is to allow the virtual world access to the same database as the Control Panel uses to maintain relationships between nodes and virtual world objects. This interaction could not be achieved if the relationships were stored in a file on the computer running the Control Panel, as the virtual world server software would not have read or write access to this file.

Using a database also means that multiple different cross-reality situations can be created that make use of the same wireless sensor network and actuator hardware and the same virtual world server software, by using different databases to store different relationship sets. A developer can use the same physical arrangement to prototype many different systems, simply by changing which database the Control Panel talks to.

A final advantage worth highlighting is the increased portability that the

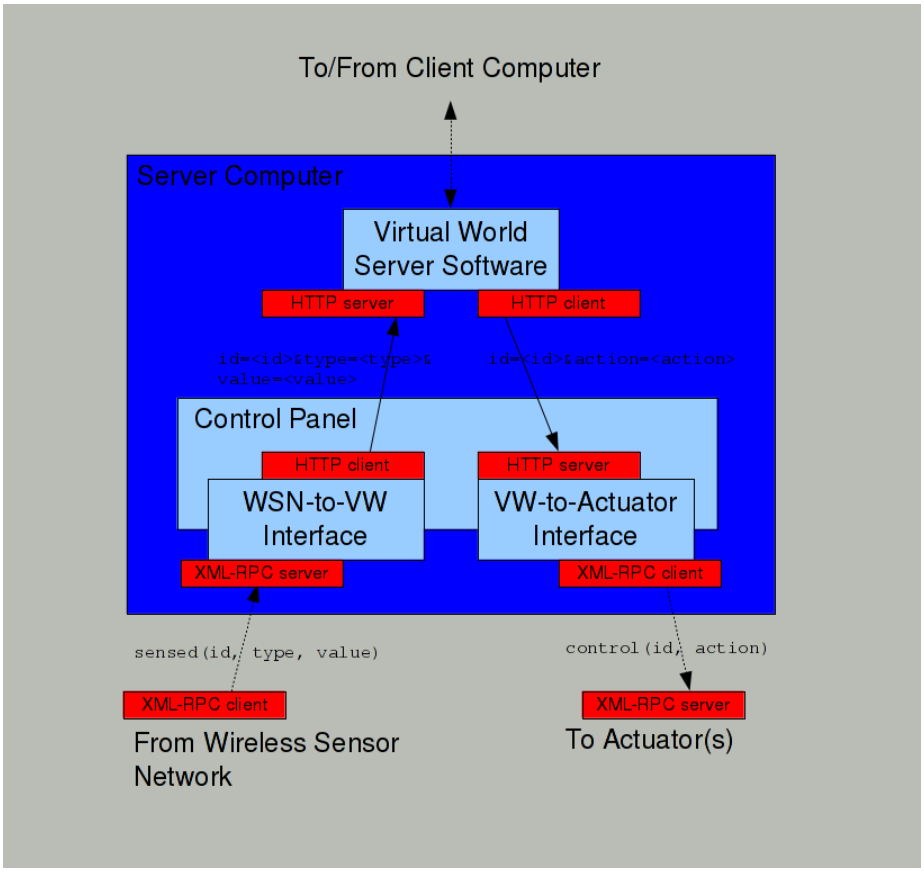

Figure 7.13: Architecture of Control Panel in relation to the interfaces and the transport technologies.

database approach allows; the Control Panel executable can be freely moved to different machines, as long as the database server is accessible from the new machine; execution can continue as though there were no changes.

7.7.1 Database Tables

Each deployment of the toolkit requires a single database with 3 tables; one to store information about WSN **nodes**, a second to store information about the **sensors** that each of these nodes hosts and a third to store information about **actuators**.

Nodes Table

This table holds;

- The unique `id` of each node.
- The address of the virtual world object that this node's sensor readings affect.

Due to the nature of WSN node hardware, it is sensible that each node should only relate to a single object in the virtual world. Even if a single node hosts multiple different sensors, such as light and temperature, the physical separation of these sensors is confined to the very small platform of the node, so the visualisation of these readings in the virtual world should likewise be confined to a single object of relative size and location to the sensors' range. As such the id field is unique, which prevents a single id being related to more than one address.

Sensors Table

This table holds;

- The `id` of the node that hosts this sensor.
- The type of sensor (eg. light, temperature) so that when a reading is delivered to an object in the virtual world the correct action can be taken determined on what type of reading has been reported. This is especially important for nodes that host multiple different types of sensor.
- The maximum value read from this sensor, or expected to be read from this sensor.
- The minimum value read from this sensor, or expected to be read from this sensor.

No single attribute in the sensors table is unique, particularly `id` because if a node hosts multiple sensors then there will be multiple records in the sensors table with this id, however the combination of id and type is unique. It is conceivable that a single node may host more than one of the same type of sensor, but even assuming this eventuality no developer would wish to identify them with identical type fields as there would then be no manner of discerning between them.

The maximum and minimum attributes keep track of the highest and lowest values that have been received by the Control Panel from this sensor, to enable basic calibration to be performed. The raw values that are sent from the WSN basestation to the Control Panel may not be of a suitable form to be passed directly to the virtual world. For example a light sensor on one type of node may report a value between 0 and 254, whilst a light sensor on a different type of node may report a value between 0 and 65535. Obviously these two readings cannot both be directly used to control lights in a virtual world without some sort of scaling such that 254 from the first sensor equates to the same intensity of light from a virtual world object as 65535 from the second sensor.

In practice even different samples of the same sensor can report markedly different values, due to variances inherent within manufacturing tolerances and by drift in hardware/firmware calibration, so software calibration is required even when only one type of sensor is being dealt with.

It is sensible to scale all sensor readings, no matter what type, to a range of 0 to 1, so that any virtual world receiving WSN readings will know what range they are dealing with. If different sensors types were to be allowed to report on different scales, this could lead to all manner of confusion when putting these readings to use.

Actuators Table

This table holds;

- The unique `id` of this actuator. An actuator id has no relation to a node id.
- The address of the actuator controller that the actuator is attached to and controlled by.

Many actuators may share the same address, as it may be the case that a single actuator controller hosts the XML-RPC gateway for many individual actuators. At the same time it may also be the case that each actuator is individually accessible by its own unique address upon which its own XML-RPC server is running; this database design allows for both possible situations.

7.8 Control Panel GUI Design

With the functionality of the Control Panel decided, the design of the GUI can be considered. As mentioned in 5.2 software prototyping can be useful for GUI design and is used extensively here.

There are three main actions that the GUI presents the user with interfaces to;

- Administration of nodes
- Administration of sensors
- Administration of actuators

It is therefore logical to split the GUI into 3 discrete sections, each focussing on one of the above areas, by using tabs. As far as possible a similar layout is used between the tabs to provide a better sense of consistency and to make it easier for a user to locate what they are looking for − for example the terminal that provides textual output about the actions that the toolkit is performing is always located in the same position at the bottom of all 3 tabs. Note that this terminal is non-editable and is used only for output, it does not allow the user to enter textual commands.

7.8.1 Administration of nodes tab

Figure 7.14 shows the design for the tab for administration of nodes, which presents the user with interfaces and controls for managing the WSN nodes that the toolkit is interacting with.

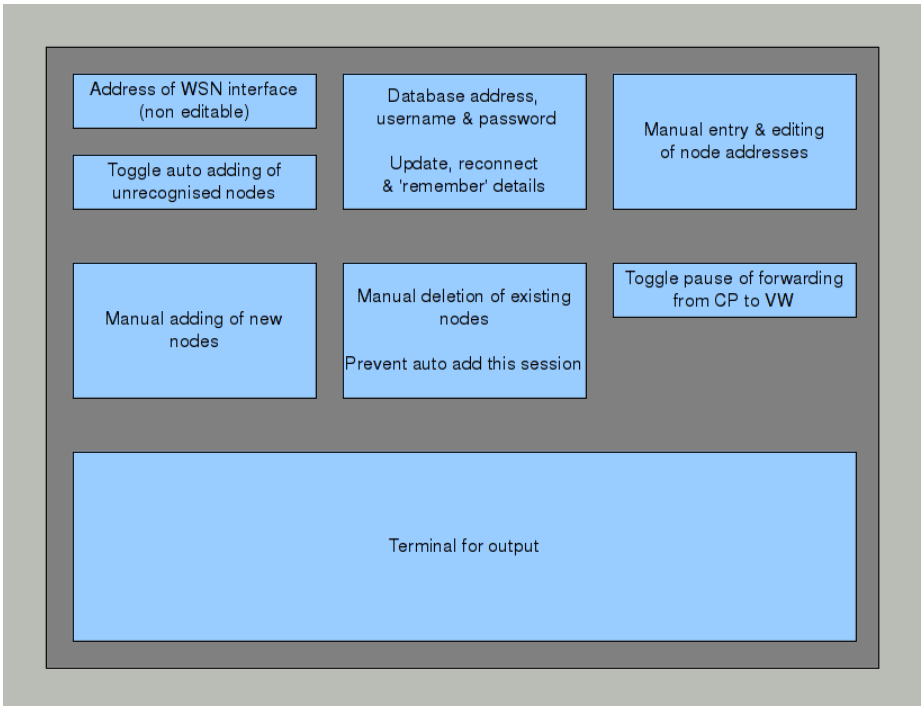

Figure 7.14: Design of node administration GUI tab.

The address that the WSN-to-VW interface is running upon is determined automatically and displayed for the user. This tab also presents the user with an interface to define what database to use to store the necessary relationships about nodes, sensors and actuators along with the ability to 'remember' these settings so that they don't have to be entered every time the Control Panel is executed.

Otherwise the main functionality of this tab is of adding, editing and removing nodes. Prototyping this functionality showed that manually adding every single node that a toolkit has to interact with is an extremely tedious activity, so the option to automatically add nodes is given – when checked, if the Control Panel receives a reading from a node that is not in its database it automatically adds it to the database. In this fashion, if a user has a WSN consisting of 100 nodes they don't have to enter the details for all 100 but just have to ensure

that they all send at least 1 message to the Control Panel and the registration will be taken care of for them. It should be noted that this automatic adding necessitates an option when manually removing a node to prevent the node from being automatically added later in the same session, which can be overridden by manually adding it.

Prototyping also revealed that a 'pause' option to stop the Control Panel from forwarding sensor readings on to the virtual world is useful. This prevents the virtual world from being bombarded with sensor readings during the initial setup of the WSN when the readings aren't actually of any particular use.

7.8.2 Administration of sensors tab

This tab, as depicted in figure 7.15, presents the user with control over the sensors associated with a particular node. By choosing from the available nodes (those that are registered with the Control Panel) sensors can be added and removed. As with the automatic adding of unrecognised nodes, a similar option to automatically add unrecognised sensors is presented.

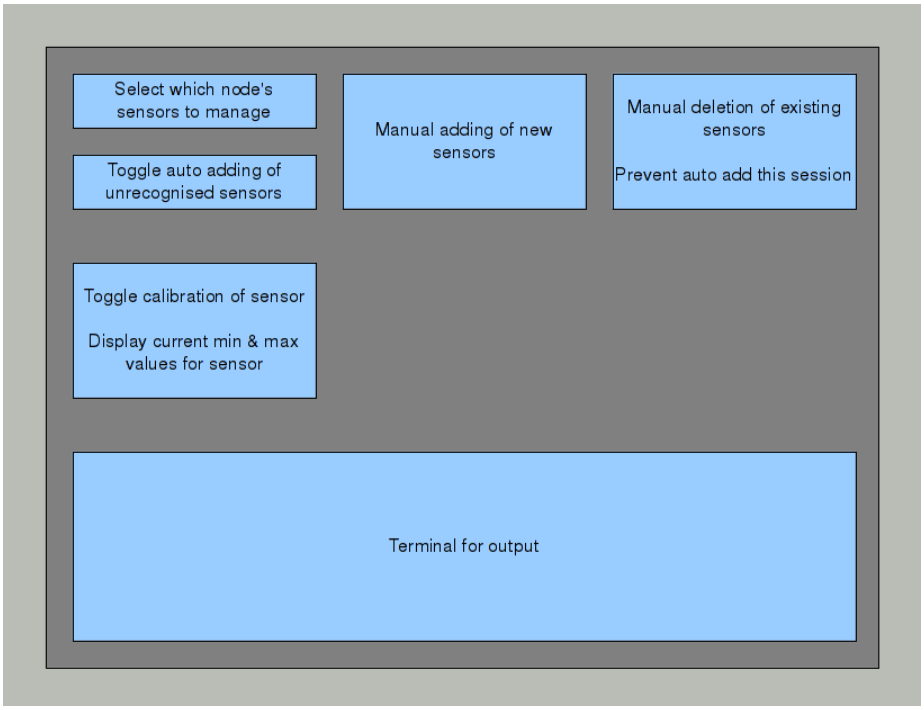

Figure 7.15: Design sensors administration GUI tab.

A further option for calibrating a node is provided; when activated any readings received from that sensor will be used to calibrate it, by updating the maximum and minimum values stored in the database for this sensor, which are then used to map the reading onto a 0 to 1 scale. It is possible that a user may wish for readings not to be scaled however, so calibration is not automatically performed and requires the user to check the tick box. There is also a reset option to set both the maximum and minimum values back to null (the same as the sensor not having been calibrated at all).

7.8.3 Administration of actuators tab

The design of the actuators administration tab is shown by figure 7.16. In a similar fashion to the nodes tab, the address of the CP-to-Actuators interface is automatically determined and presented for the user. Likewise an option to pause the forwarding of commands from the Control Panel to the actuators is provided – this is especially important for actuators as they may be in control of large, expensive &/or dangerous equipment so undesired activation during administration or testing and development with a system must be guarded against.

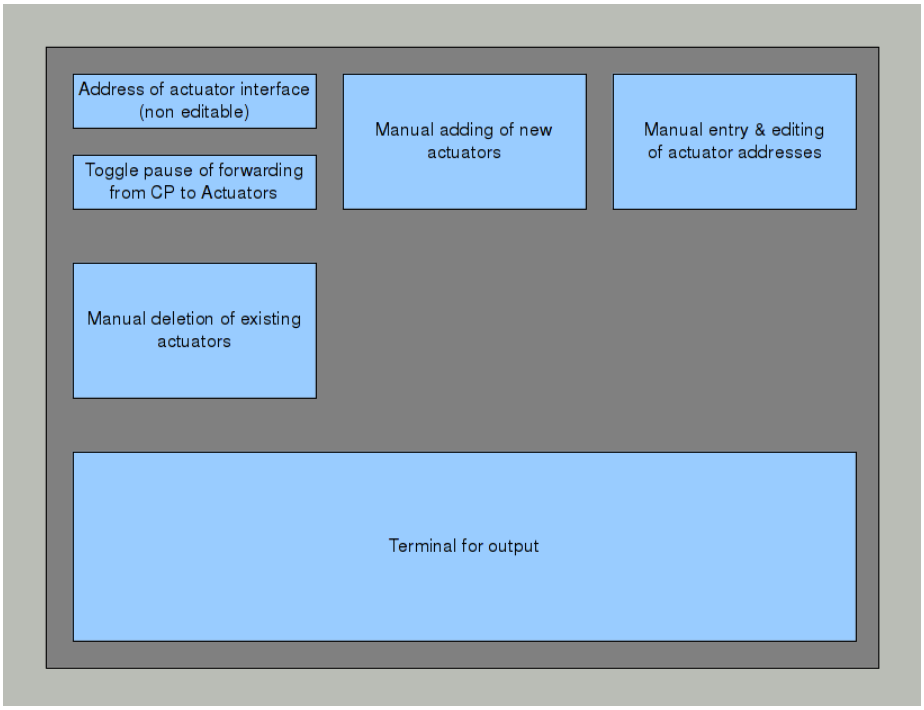

Figure 7.16: Design of actuators administrationGUI tab.

Because the VW-to-Actuator protocol is intentionally very abstract to allow for maximum compatibility and extensibility with a wide range of heterogeneous actuator controllers, it is not possible to implement a feature for automatic adding of previously unrecognised actuators. The standard used for the content of the `action` field is not known to the Control Panel, as it is specific to the actuator platform being used, so the Control Panel cannot parse it to extract information such as the address of the controller for this actuator, or the type of the actuator.

7.9 Summary

The toolkit consists of a single central application referred to as the Control Panel, which encapsulates two remotely invokable interfaces. The WSN-to-VW interface presents a WSN with the ability to report sensor readings to the Control Panel, which then parses and processes them before forwarding them to the appropriate virtual world object. The VW-to-Actuator interface presents the virtual world with a means of sending actuator commands to the Control Panel, which then forwards them on to the correct actuator controller.

Communication between the WSN, actuators and the Control Panel is achieved through XML-RPC, whilst communication between the virtual world and the Control Panel is accomplished using HTTP POST. This maintains maximum compatibility with the widest range of virtual world, WSN and actuator platforms, as well as creating an amiable environment for development and debugging that consists of freely available and openly accessible technologies.

A database is used to store information and relationships between nodes, sensors and actuators, allowing the Control Panel to be closed and reopened, or even moved between different computers, without losing any state. It also allows rapid switching between multiple different cross-reality situations that make use of the same WSN, actuator and virtual world deployments, simply by changing which database the Control Panel accesses.

The Control Panel presents the user with a GUI that allows them to easily perform administration over the nodes, sensors and actuators of the system, as well as defining the details for access to the database and providing diagnostic output in terminals about the events of the system as information is flowing between the different components.

Chapter 8

Implementation

This chapter discusses the implementation of the toolkit as well as certain aspects of the implementation of the example use case, which is used both to aid in the design and development of the final solution and also to allow testing to be performed. There is not enough space in this document to discuss every feature of the implementation at a low level, so particular aspects are chosen and highlighted where importance is associated.

8.1 Selection of Example Use Case

It is necessary at this point to identify which of the example use cases, listed in 1.3, is implemented to demonstrate the toolkit's functionality and success. This scenario is not only employed for testing purposes after implementation is complete but is also used to guide implementation through development of prototypes from an early stage.

With the current media focus and hype over the importance of energy efficiency and 'carbon footprints' it seems appropriate that 1.3.3 is the example use case implemented, as it provides a good representation of a topical real world scenario where the toolkit can be used. An ulterior motive for this decision is an allusion to the long berated automatic lighting system that features in one of the department's buildings and keeps lights switched on even when nobody is present for many hours, wasting electricity.

8.1.1 Details of Use Case

Full details of the use case implementation are found in Appendix A as it serves as the primary testing platform, however some knowledge of it is required now to fully appreciate and comprehend this chapter's content.

The demonstration combines the Tmote Invent WSN platform [71], running nesC [72] applications upon the TinyOS operating system [73], with the virtual world server software OpenSim [74] and the Phidgets USB interfaces. [75]

A simulation of the department's John Honey building was created in OpenSim and using the toolkit, readings from the Tmote's light sensors control the illumination of several distinct lights in the simulation, whilst other lights in the simulation can be commanded by an avatar to send actuator commands to a custom built lighting fixture in the real John Honey building where two lightbulbs are controlled by the Phidget interface.

Extensive software and hardware prototyping is performed upon these technologies and forms an integral component of the implementation stage of the project, as new technologies are learned and new techniques experimented with to gauge their ability and suitability to achieve the goals ahead.

8.2 Initial Prototyping

Learning enough about WSNs, actuators and virtual worlds through software and hardware prototyping to become proficient enough at developing with all of them is the first stage in implementation of the solution. Although most of these software prototypes aren't ultimately included in the solution and don't directly contribute to its codebase, in-depth knowledge and understanding of all of them has to be achieved before a suitable implementation can be written. This steep learning curve and familiarisation is required, because without fully comprehending the capabilities of each technology, what restrictions they are subject to, what provisions they have for communicating and interacting with the other technologies and even just how they behave during development and deployment, it is impossible to meet the project's requirements.

Also, because the toolkit is intended for use by developers who work with these technologies, it would be short-sighted to design and develop it without first spending considerable time 'in their shoes' to experience what their development environments and practices are like. By experiencing these first hand it is possible to design the toolkit in such a way that it merges as well as possible into the practices and approaches that these developers are already akin to.

8.2.1 WSN Prototyping

The first prototypes were concerned with becoming familiar with WSNs as this was the area of least prior experience and which posed the greatest challenge. Even at a simple scales, such as the use case where sensing nodes take readings at set intervals with their light sensors, send these over short-range radio to the basestation node, which then presents them to the connecting computer via a serial forwarding service, many weeks of reading documentation and tutorials is required to gain enough experience to successfully extract the sensor readings from the basestation node.

Learning the basics of WSN development, such as the modular construction and execution model of nesC applications is frustratingly complex as it represents a development paradigm so different to any previously encountered traditional programming. Once these basics have been mastered and simple

applications compiled and deployed, further prototypes were created to investigate the communication techniques that were available, between nodes but also between the basestation node and host computer as this is where the interface to the Control Panel is addressed.

Figure 8.1 shows an extract from one of these early nesC prototypes, which involves a single node connected directly to a computer, taking readings from its light sensor and sending them to the computer over serial.

```
implementation {
  bool sbusy = TRUE;
  message_t spacket;

  event void Boot.booted() {
    call Leds.set(0);
    call SerialControl.start();
    call PhotoControl.start();
  }

  event void TestTimer.fired() {
    call Photo.read();
  }

  event void Photo.readDone(error_t result, uint16_t data
      ) {
    TestMsg* msg;

    if(!sbusy) {
      sbusy = TRUE;

      msg = (TestMsg*)(call SerialPacket.getPayload(&
          spacket, NULL));
      msg->data = data;

      if(call SerialSend.send(AM_BROADCAST_ADDR, &spacket
          , sizeof(TestMsg)) != SUCCESS) {
        sbusy = FALSE;
      }
    }
  }
```

Figure 8.1: Extract of nesC prototype that sends light readings to a computer directly over a serial connection.

The computer then uses a Java program, an extract shown in figure 8.2, to retrieve the reading from the message in its hex format and prints it to the

terminal.

```
public class Test implements MessageListener {
  public static final String PORT = "/dev/ttyUSB0";
  public static final String PLATFORM = "tmote";
  private static boolean stest = false;

  public static void main(String[] args) {
    String comm = "serial@" + PORT + ":" + PLATFORM;
    PhoenixSource ps = BuildSource.makePhoenix(comm,
        PrintStreamMessenger.err);
    MoteIF mif = new MoteIF(ps);
    Test lt = new Test(mif, null);
  }

  public static String hexVal(int val) {
    String str = "00" + Integer.toHexString(val).
        toUpperCase();
    return str.substring(str.length() - 2);
  }

  public void messageReceived(int to, Message message) {
    TestMsg tm = (TestMsg)message;
    System.out.println(">> " + tm.get_data());
  }
```

Figure 8.2: Extract of Java prototype that retrieves sensor readings from hex messages received from nodes.

8.3 Interim Demonstration

By the date of the interim demo a substantial prototype with fully operational WSN-to-VW functionality had been developed. In this prototype sensing nodes take light readings and send these via radio to a basestation node. From here they are passed to a Java application that extracts the necessary information and inserts it into a HTTP POST. The URI of the appropriate primitive in OpenSim is obtained directly from a database via JDBC and the POST is sent to this address and the simulated lights' intensities change accordingly. The architecture of this prototype is shown by figure 8.3.

At this stage the implementation of the interfaces and of the Control Panel itself are not yet formally achieved; this prototype is intended as a proof-of-concept and a platform for experimentation, bespoke in the manner that it

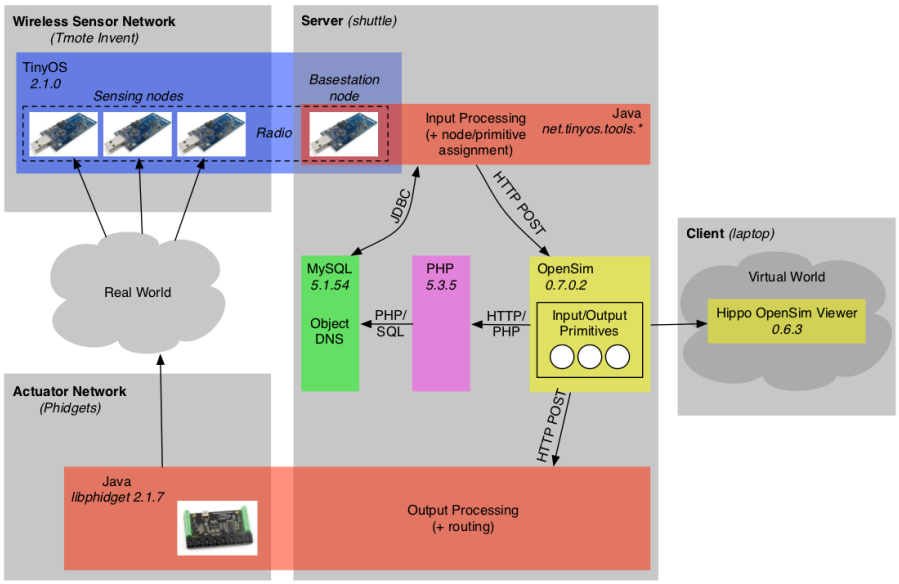

Figure 8.3: Architecture of the prototype system shown at the interim demonstration.

does not implement concrete standards and protocols for the communication between different components, but demonstrates the ability of the underlying technologies to achieve these communications.

The presence of PHP is due to the use of a modified version of ObjectDNS, a simple Domain Name System (DNS) for Second Life and OpenSim primitives, written by a member of the Second Life community [76] and modified for the demonstration. It sends an HTTP POST from each primitive that registers a URI to a PHP script running on a webserver, which updates the URI of this primitive in the database. The Java application that processes the sensor readings queries this database to determine the current URI of the primitive that a particular reading relates to. This interaction is discussed as a desirable feature in 7.7.

A prototype lighting fixture, controllable by computer via a Phidget interface had been constructed (see figure 8.4), however it was not at this point controllable from OpenSim so the demo only achieved the state of augmented virtuality. Further development soon after the demo succeeded in controlling the fixture and achieving cross-reality.

This prototype confirmed the ability of HTTP to act as transport between WSN, actuators and virtual world. It also highlighted the importance of a GUI, which had not been considered a high priority until experience interacting with this real world deployment.

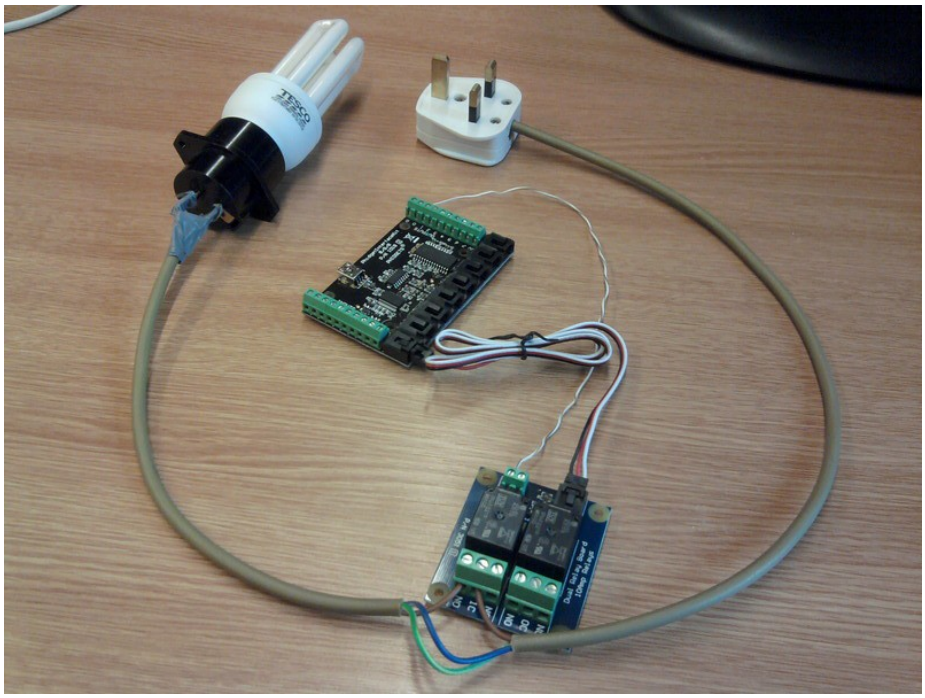

Figure 8.4: The prototype lighting fixture controllable from a computer via a Phidget USB interface and Single Pole Double Throw (SPDT) relay.

8.4 Ratification of Protocols

After the success of the interim demo prototype in achieving cross-reality and proving the suitability of HTTP, the standards and protocols for the information interchange between components could be ratified.

HTTP was used for the interim demo without investigating more complex alternatives, due to the relative ease of its integration – beneficial for fast prototyping. XML-RPC was subsequently chosen to replace HTTP as transport between the Control Panel and WSN/actuators (see 7.4.1), but upon discovering the sad state of XML-RPC support in Second Life and OpenSim the HTTP functionality from the prototype is adopted by the final implementation for communication between the Control Panel and the virtual world.

A standard was also decided upon for the node-to-node radio communications. Whilst this is not within the scope of the toolkit, it makes the implementation of further prototypes and the example use case much easier, by allowing more abstracted access to the contents of a message. The Message Interface Generator (mig) is used to automatically generate 'getter' methods in a Java class (see figure 8.5) for a message syntax provided in a header file, so that a Java program that receives messages from the basestation node can access the

```
public class PhotoToRadioMsg extends net.tinyos.message.
    Message {

/**
  * Return the value (as a int) of the field 'nodeid'
  */
   public int get_nodeid() {
      return (int)getUIntBEElement(offsetBits_nodeid(),
          16);
   }

/**
  * Return the value (as a int) of the field '
     photoreading'
  */
    public int get_photoreading() {
        return (int)getUIntBEElement(
            offsetBits_photoreading(), 16);
    }

}
```

Figure 8.5: Extract of MIG-generated Java code that allows easy access to node radio messages by providing 'getters'.

```
public void messageReceived(int to, Message message) {
  try {
    int photoreading = 0;
    int nodeid = 0;
    if (message instanceof PhotoToRadioMsg) {
    photoreading = ((PhotoToRadioMsg)message).
        get_photoreading();
    nodeid = ((PhotoToRadioMsg)message).get_nodeid();
  }
```

Figure 8.6: Example of Java code that uses MIG generated methods in figure 8.5 to access data from a node's radio transmission.

content of these messages by using these getters (see figure 8.6), rather than having to access specific indexes in the message and then convert from hex.

8.5 GUI

GUI development used the functional interim demo prototype as a testbed to ensure that every required feature was present and that they were presented in a logical and efficient manner. Designing a GUI on paper is simple enough, but often it is not possible to ascertain how well the design will actually function without applying and testing it upon a prototype or production system and subsequently modifying it accordingly. Figures 8.7, 8.8 and 8.9 are screenshots of the 3 GUI tabs created.

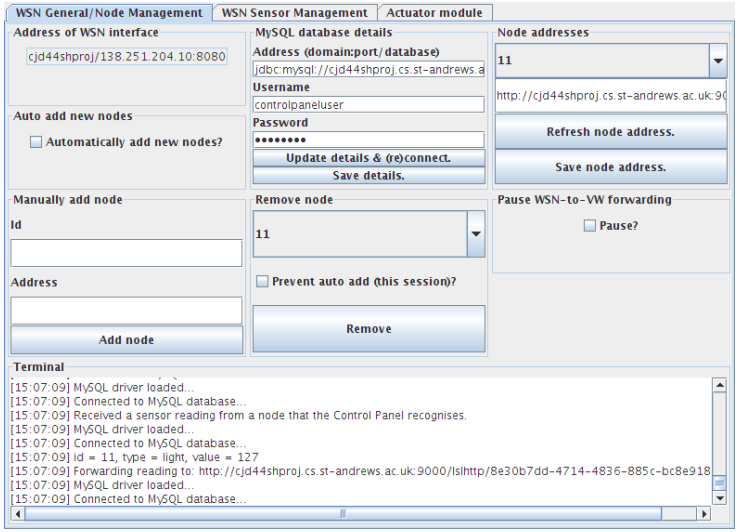

Figure 8.7: Screenshot of node administration GUI tab.

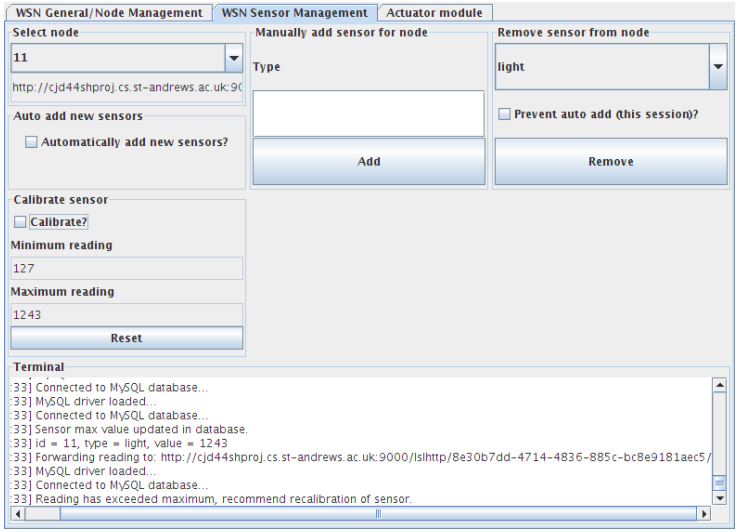

Figure 8.8: Screenshot of sensor administration GUI tab.

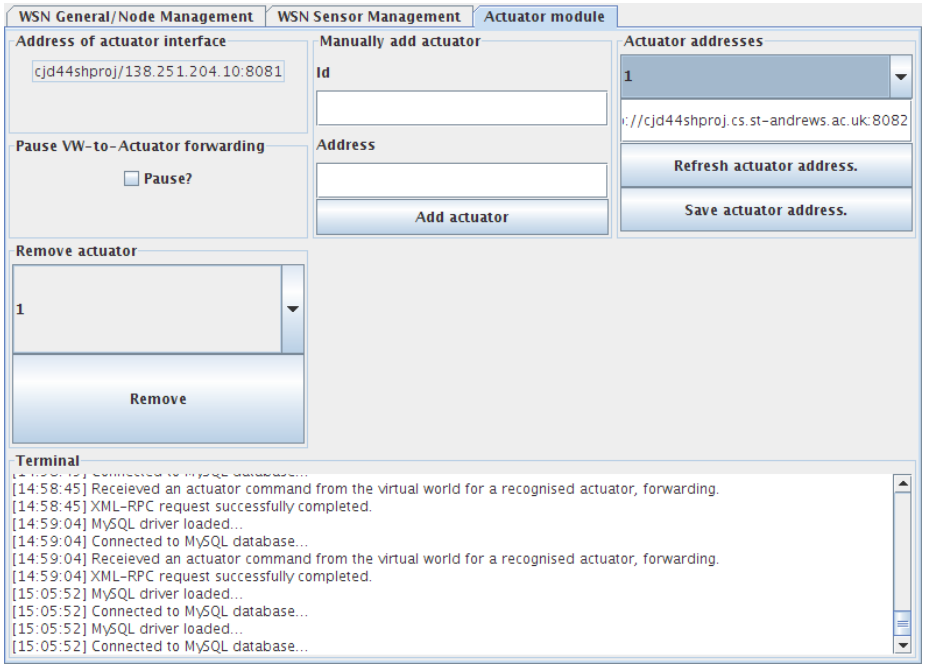

Figure 8.9: Screenshot of actuator administration GUI tab.

8.6 Maintaining State

One of the toolkit's main responsibilities is to maintain records of nodes, sensors, actuators and addresses. As discussed in Chapter 7 this is achieved by a database, however simply defining the structure of the tables and the types and restrictions on their attributes is not enough; the ordering and dependencies of database activity have to be carefully considered in order to maintain integrity.

Imagine the scenario where a user adds a node to the Control Panel with id `node1`, then adds 3 sensors to this node; `light`, `heat` and `pressure`. Now presume the user removes `node1` from the Control Panel. Later if the user adds a new node and happens to reuse id `node1`, they should not expect there to be `light`, `heat` and `pressure` sensors associated with it – these should have been removed automatically when the original `node1` was removed.

Subtle dependencies like this, if not identified and guarded against, threaten integrity of the database. Such an oversight could have disastrous effects – a command delivered to the wrong actuator could cause damage to expensive equipment and even threaten human life.

Flowcharts 8.10 and 8.11 show the processes involved when the Control Panel receives a sensor reading. Figure 8.10 continues into figure 8.11, connected by

the boxes with dashed outlines. Flowchart 8.12 shows the process involved when the Control Panel receives an actuator command from a virtual world. All of these checks and validations are performed each time the Control Panel attempts to modify the database.

See Appendix C.1.1 for the syntax used to set up the database.

8.7 Summary

Achieving proficiency with the constituent technologies involved in creating cross-reality systems was the first and most important part of the implementation process. Without devoting the considerable time required to overcome the challenge of learning these disparate development practices the final solution would not have been a success.

Substantial prototyping, both software and hardware, was performed to investigate the capabilities and restrictions of each technology, and to ascertain the best methods and approaches for bringing them together in harmony. Although the vast majority of this prototype code does not feature in the final solution, it directly contributes to its success. After attaining the required level of proficiency, implementation of the final solution continued smoothly.

Considerable theoretical work was also performed in ratification and testing of protocols and in consideration of data integrity with regards to the supporting database. To ensure that the solution maintains correct operation, careful extrapolation and investigation of every possible use case scenario of adding, removing and editing nodes, sensors and actuators from the Control Panel had to be performed, to identify the checks and dependencies that have to be addressed when each operation is performed.

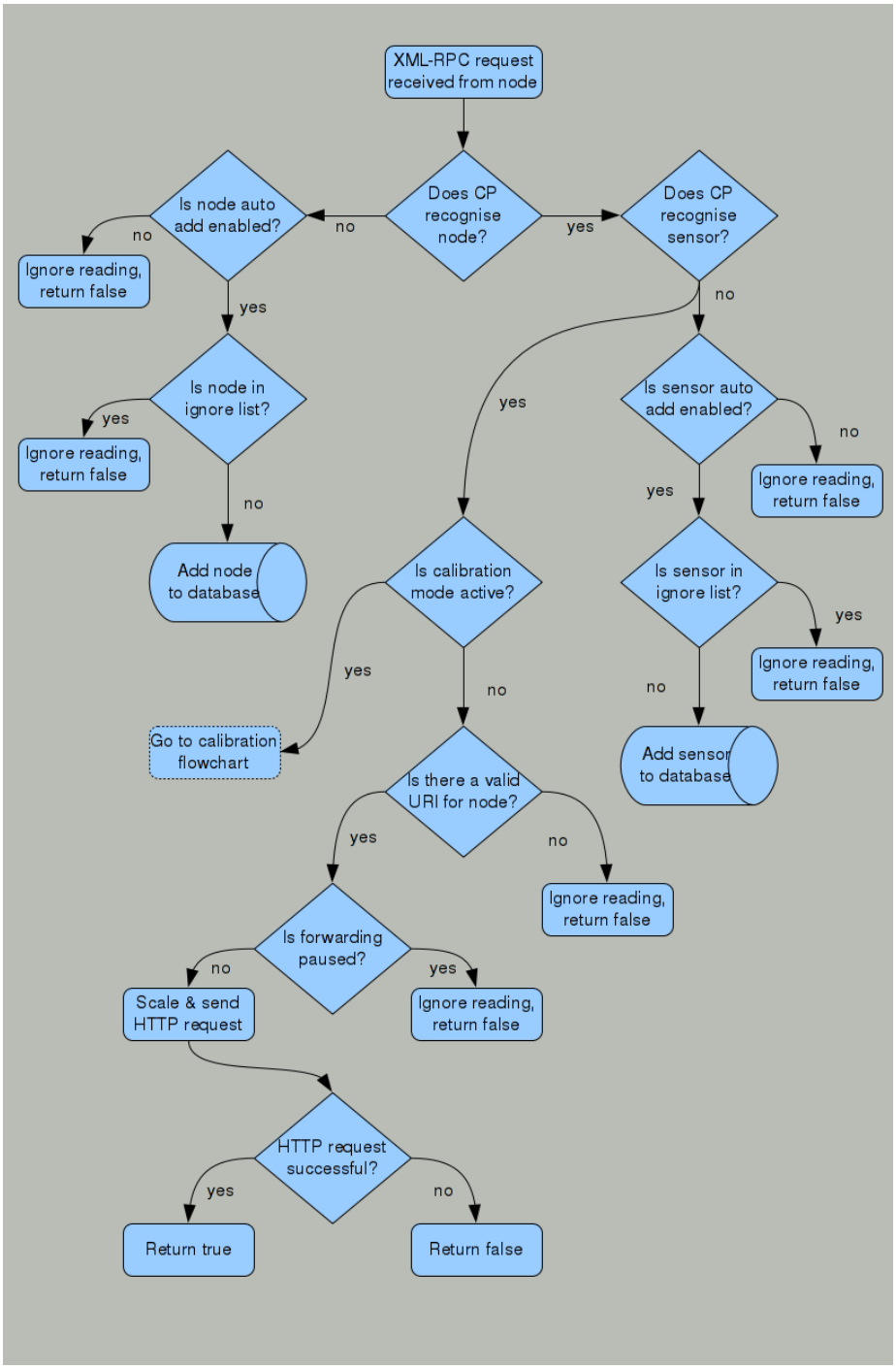

Figure 8.10: Flowchart of activity when the Control Panel receives a sensor reading from a node.

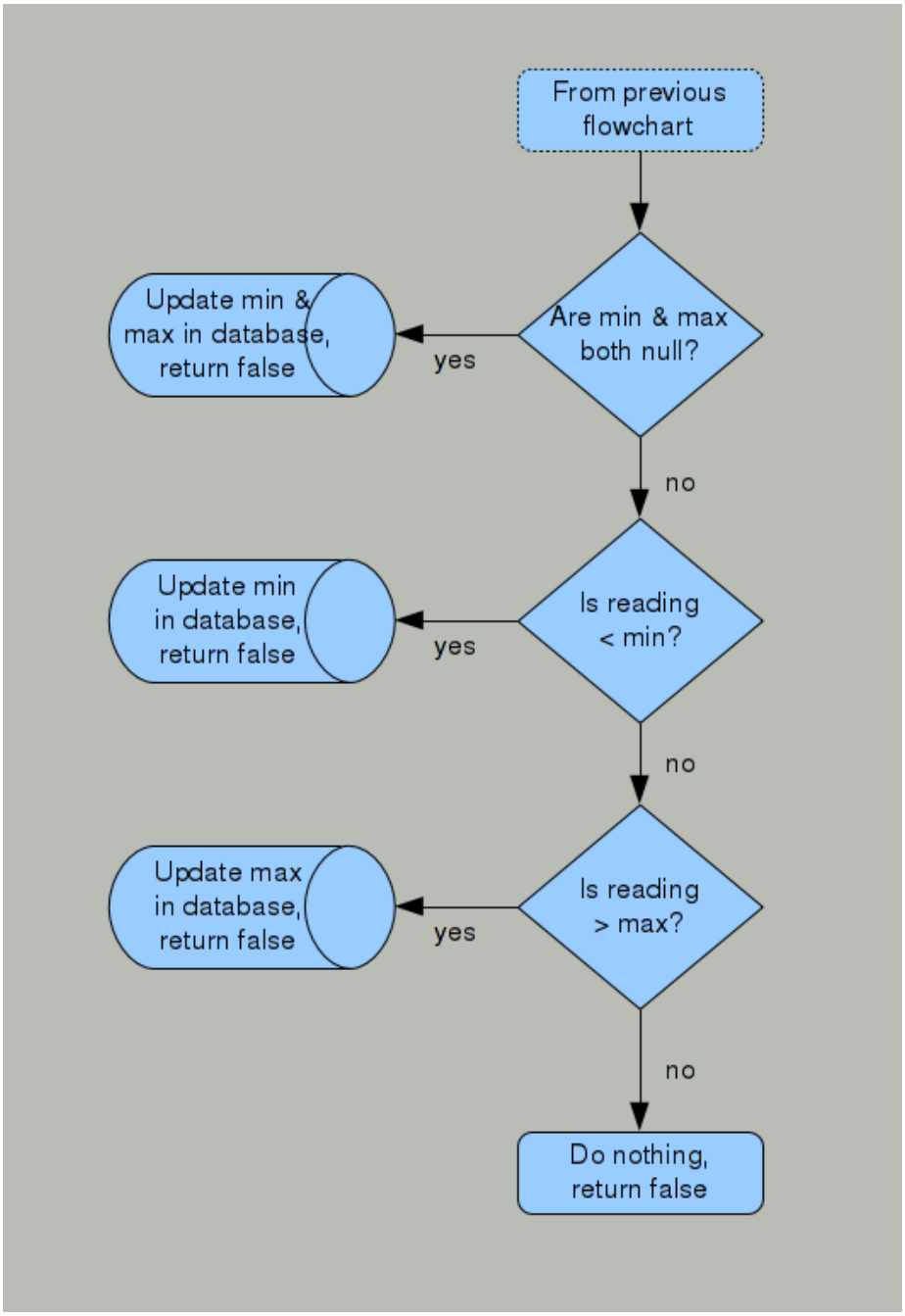

Figure 8.11: Continuation of flowchart in figure 8.10, for calibrating a sensor.

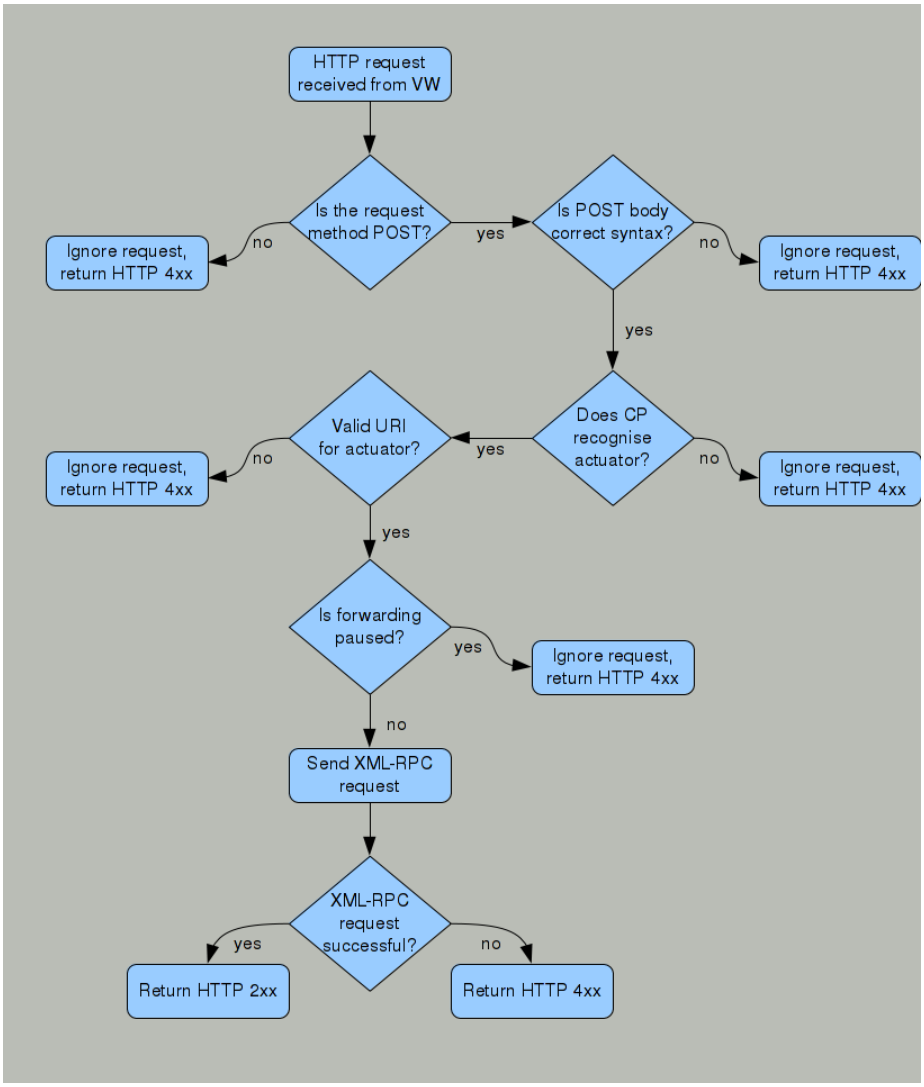

Figure 8.12: Flowchart of activity when the Control Panel receives an actuator command from a virtual world.

Chapter 9

Evaluation and Critical Appraisal

In this chapter the fruit of the project is evaluated and appraised against the original objectives and requirements as laid out earlier in this document. It is also compared and contrasted against similar work in the public domain, such as that published in papers and distributed as software.

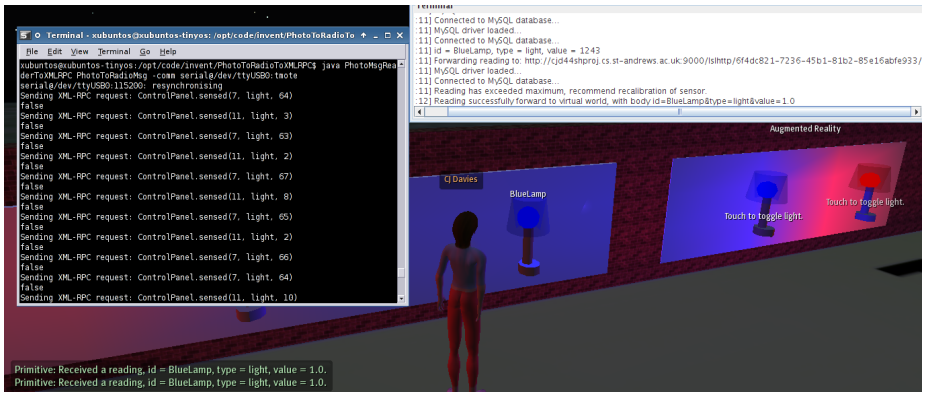

Figure 9.1: Using the example use-case implementation to test how well the solution meets its objectives and requirements.

9.1 With Respect to Original Objectives

The project has completely achieved all of its original primary and secondary objectives (see 2.1). In hindsight these objectives were poorly organised between primary and secondary, however they have been met regardless.

9.1.1 Confusion of Original Objective Classification

The overall intent of the project can be broadly divided into 2 main tasks;

1. Develop a software toolkit that facilitates bidirectional informational or media flow between WSNs, actuators and virtual worlds.

2. Deploy this toolkit to an example use case scenario to test and prove its functionality, suitability and success.

As such it would have made sense for these two tasks to serve respectively as the basis for the original primary and secondary objective categories; the primary objectives to create the toolkit, the secondary to deploy it. However the second task, of deploying the toolkit to an example use case, actually had its constituent subtasks split across the primary and secondary objective categories.

The reason for this is that when the objectives were written, not enough was known about the technologies concerned. It was naïvely conceived that the toolkit would include applications that ran on the WSN component and within the virtual world itself. As further research into these technologies was performed and software and hardware prototyping began, it became apparent that the concepts of WSNs, actuators and virtual worlds were too vague and heterogeneous for a single application written for any of them to be capable of running on more than a single, or at most a handful, of platforms. For example, if some form of interface to the central application had been developed for usage within a Second Life simulation, this interface would not be compatible with the Active Worlds platform. Likewise a WSN application written for the Contiki platform would not be compatible with the TinyOS plaftorm.

Thus whereas original primary objectives 1a and 1b were correctly classified, 2 and 3 should have been classified as secondary objectives.

9.1.2 Further Discussion of Original Objectives

Original primary objectives 1a and 1b have been met entirely by the software toolkit that has been produced. This presents a central Control Panel application that hosts two interfaces; one to facilitate a WSN to send information to a virtual world and the other to facilitate a virtual world to send commands to actuators. These can be used simultaneously.

Original primary objectives 2 and 3 have been met as part of the example use case implementation (full details with screenshots and photographs in Appendix A);

- A WSN application for the TinyOS environment was written in the nesC language for deployment onto the Tmote Invent platform. This consists of one program to be deployed to sensing nodes, which takes readings from the Invent's light sensor at set intervals and sends these via short-range radio to a second program to be deployed to the basestation node, which listens for incoming radio communications and forwards them via serial to

the computer it is connected to and on to OpenSim via the appropriate interface.

- Desk lamp primitives were created and scripted in OpenSim such that when an avatar touches them they send commands to the actuators via the appropriate interface.

Original secondary objectives 1 through 6 have also been met by the example use case implementation. A simulation of the John Honey building was created in OpenSim, Tmote Invents were installed and deployed with the sensing and basestation applications and the actuator controlled lighting fixture was deployed and programmed with Java to receive commands from the desk lamp primitives in OpenSim. This setup then succeeded in visualising real time physical &/or environmental changes in the real John Honey building in the simulation and in effecting physical &/or environmental changes in the real John Honey building in response to avatar actions in the simulation. These actions were performed simultaneously and a cross-reality environment was successfully created and maintained.

Original tertiary objective 1 was poorly conceived but has been partially met. The way in which the toolkit was designed and developed means that it does not need to be expanded to support more physical and environmental stimuli; a single node can have essentially unlimited different sensors associated with it, bounded only by the capabilities of the MySQL database. If a developer wishes to use an extremely unusual sensor type, as long as its readings can be represented and transported in a numerical format, which was previously determined a fair assumption to make, then the toolkit is capable of supporting it.

The second part of original tertiary objective 1, expanding the WSN application to support more physical &/or environmental stimuli, was conceived when it was expected that the toolkit would include a WSN application. As this is not actually the case, it renders this part of the objective redundant. However, this objective could have been adopted by the example use case implementation for which a WSN application was developed but which only uses light sensors, to better demonstrate the capability of the toolkit to support a heterogeneous range of different sensor types by making use of some of the other sensors present on the Tmote platform.

Original tertiary objective 2 has not been met, but again this would only have expanded the scale of the demonstration to better show how the toolkit handles higher loads − it would not have increased the functionality of the toolkit itself.

9.2 With Respect to Emergent Objectives

Emergent primary objective 1 has been met in full, thus allowing the successful implementation of the toolkit, and this represents a significant achievement. If this objective had not been met, then it would not have been possible to meet

the other objectives as designing and developing the toolkit relied upon the knowledge and experience gained from meeting this objective.

Emergent secondary objective 1 has been met by the design and fabrication of a lighting fixture that is capable of independently switching 2 standard light-bulb sockets using relays connected to a Phidget interface kit that allows control from a computer connected by USB. This fixture was used in the example use case implementation and further details can be found in Appendix A.

Emergent tertiary objectives 1 and 2 have not been achieved. Similar to how it evolved that original primary objectives 2 and 3 were naïvely conceived, as a single application could not be made to support cross platform deployment to heterogeneous virtual worlds and WSNs, emergent tertiary objectives 1 and 2 were also discounted as possible components of the toolkit.

9.2.1 Virtual Sensors

The virtual sensor paradigm of emergent tertiary objective 1 could not be implemented in a manner independent of virtual world platform but was also considered to be at too high a level of abstraction to constitute part of the toolkit.

The motivation for the toolkit was to allow developers to more easily integrate WSNs and actuators with virtual worlds to create cross-reality systems. It achieves this by abstracting over the lower level organisation, connectivity and communication between these different components. Once a sensor reading has been delivered to the virtual world, or an actuator command has been delivered to the actuator controller, the toolkit's responsibility comes to an end. How these readings and commands are subsequently used is up to the individual user and is of no concern to the toolkit − this information is above the level of abstraction that the toolkit operates at.

9.2.2 Greater Control of Toolkit from Virtual World

Emergent tertiary objective 2 proposed an extension to original primary objective 3. It has already been identified that this primary objective was naïvely conceived as a single application cannot be written that supports deployment to any virtual world. Thus achieving this emergent objective would only have affected the example use case implementation − it would not have had any bearing on the implementation nor functionality of the toolkit itself, as it would not have been able to form part of the cross-platform solution.

9.3 With Respect to Requirements Document

The system is now evaluated with respect to the functional and non-functional requirements laid out in the Requirements Document (see 4.6). These requirements were the result of an analysis of the original objectives, expressed more

formally, so the discussion here is brief as it covers much of the same as already covered in 9.1.

9.3.1 Functional Requirements

All of the functional requirements (see 4.6.1) have been met. The toolkit facilitates a WSN to communicate with a virtual world via an interface, a virtual world to communicate with actuators via an interface, and allows both of these actions to take place simultaneously.

9.3.2 Non-Functional Requirements

Whilst harder to quantify success with regards to non-functional requirements, they have all been met. Numbering of the beneath explanations matches that of the non-functional requirements as they are listed in 4.6.2.

1. Whilst the toolkit has only been demonstrated with a single example each of WSN, virtual world server and actuators, the interfaces that it presents were designed and developed to be compatible with a wide range of different platforms. Whilst it would be short-sighted to claim compatibility with all platforms, the vast majority should be able to implement the interfaces.

2. HTTP and XML-RPC are adopted as the transport between the WSN, actuators and virtual world server. These are extremely common, well-established and stable standards, accessible to a great many languages and it would not be untoward to expect most developers who might make use of the toolkit to already have had experience working with at least HTTP, if not XML-RPC as well.

3. The interfaces (see 7.4.2 and 7.5.2) are presented in a concise and simple manner, where the purpose of each field is both intuitive and well explained by this report.

4. The use of HTTP and XML-RPC ensures that the communications protocols are simple, efficient and easy not only to use but also to debug.

5. The toolkit can be rapidly deployed upon a single computer, but also makes it just as easy to deploy it to a distributed system involving multiple different computers, where each hosts a different component of the overall system (see Appendix C for more details on deployment).

6. Low message complexity and computational complexity ensures that the system is very responsive in both directions of communication.

9.4 With Respect to Related Work

Examples of related work are discussed in 3.5. The conclusions drawn from this discussion are that whilst there have been other projects, both commercial and academic, that have created and hosted cross-reality systems, these have all represented bespoke, one-off implementations that did not attempt to define architectures, standards or protocols to aid the development of similar systems in the future by other researchers and developers, particularly those outside the particular institution that bore the existing systems.

This project differs from these existing implementations in that it is not foremost concerned with actually creating a cross-reality environment. Instead it focusses on researching and analysing the constituent technologies involved in cross-reality systems to gain enough information and knowledge about their capabilities and restrictions, and also the style of development associated with them, to be able to design a toolkit that allows any WSN platform, virtual world server and actuators to be easily combined to create a cross-reality environment. The fruit of this project is not a cross-reality system as others have bore before, but a tool that allows for easier creation of cross-reality systems in the future.

9.4.1 DoppelLab

There may be similarities between this project and DoppelLab (see 3.5.2) as DoppelLab alludes to provide both *"an open-ended platform for building visual applications atop... ...data"* captured by a WSN and that *"applications in turn become sensor-driven interfaces to physical world actuation and control"*. However the project is still in its early stages, has yet to publish and in particular it is not obvious from the short video and brief explanation on the project's homepage whether or not it directly interfaces with actuators. The emphasis from the video's voiceover is toward providing a platform for the visualisation of WSN readings, rather than upon attempting to support creation of cross-reality situations.

9.4.2 MPEG-V

The MPEG-V standard (see 3.4.1) is the most similar of the related works. Its aims are broadly similar to this project's, in that it *"intends to provide a standardized global framework and associated interfaces, intermediate formats definitions and the like, to enable the interoperability between virtual worlds... ...and between virtual worlds and the real world (sensors, actuators..."*. The distinction between MPEG-V and the project of this report is that MPEG-V is still under development and has yet to be ratified; the project of this report presents a complete, functional and tested implementation of such a framework. The success of this project supports the justification behind the proposal and development of MPEG-V.

9.5 Summary

This project has met all of its significant objectives and all of its requirements. Where objectives have not been met, or have only been partially met, this is either due to these objectives being ill-conceived at a time when not enough was known about the technologies and thus rendered redundant, or their sole purpose was for extending the scope of the implemented use case example used to demonstrate and test the toolkit.

With respect to related work, this project represents a fairly unique achievement; existing cross-reality implementations have not attempted to define the necessary architectures, standards or protocols to enable future development of new cross-reality systems by others; it is too early in the development of the DoppelLab project to ascertain exactly what it intends to accomplish; and with MPEG-V still in the development stages this project instead provides the only completed framework with an example implementation to support it.

Chapter 10

Conclusions

In summary the toolkit developed by this project allows a user to easily combine WSNs, actuators and virtual worlds to create cross-reality systems where there is simultaneous bidirectional informational flow from WSN to virtual world and from virtual world to actuators.

The toolkit comprises an easily deployable central application that presents a GUI for administration and management of nodes, sensors and actuators, and provides the WSN and the VW with well defined interfaces for sending information from the real world to the virtual world and vice-versa. By making these interfaces remotely addressable over a network via common and popular protocols the possible deployment architectures supported, in terms of the distribution of components between different logical computers, are numerous and by insightful design of the interfaces themselves there is no reliance on any particular WSN, actuator or virtual world platform.

10.1 Key Achievements

Whilst the example use case implementation of a cross-reality environment is immediately intriguing, visually impressive and a substantial achievement in its own right, it is only a demonstration of the real success of the project – the toolkit that has been developed and which allows such cross-reality environments to be so easily created.

10.1.1 Attaining Development Proficiency

The most substantial achievement during this project was starting with no experience about any of the technologies involved and subsequently overcoming the challenge of learning enough about them and becoming proficient enough with their development environments and methodologies within a short space of time to design and develop the toolkit in such a successful manner.

Whilst the toolkit could have been implemented with lesser knowledge about

these technologies and tested through software simulations of WSNs and actuators, the key to how well it operates and integrates with each technology is a direct product of the time and effort devoted to becoming so experienced with these technologies.

10.1.2 Heterogeneity and Accessibility of Deployments

From the outset of the project it was obvious that for true success the toolkit would have to support as many different technologies and deployment models as possible. It achieves this fully by using common protocols (XML-RPC and HTTP) for communication between WSN, actuators and virtual world and by developing the Control Panel application in the Java language, so that it can be deployed upon any computer with any operating system as long as it has a running JVM.

By making the interfaces remotely invokable, the WSN basestation node and the actuator controller can be in completely different geographical locations to the computer running the Control Panel, which in turn can be in a completely different location to the computer running the virtual world server software. The WSN basestation and actuator controller could be at the top of a mountain, invoke the interfaces using a satellite connection to the Internet, with the Control Panel running on a researcher's laptop in their field office and the virtual world server running in a machine room back at their department on the other side of the world.

10.1.3 GUI

It was not conceived at the stage of the requirements elicitation that a standalone GUI would be required as part of the toolkit; only after software and hardware prototyping reached the stage of a full deployment involving WSN, actuators and virtual world did it become apparent that a GUI would be required for easy administration and control over the cross-reality environment created.

The successful implementation of the GUI is proof of the success of throwaway software (and hardware) prototyping and represents good project management by successfully reassigning the available development time between the different parts of the system so that there was enough time to develop and test the GUI in addition to the other requirements.

10.2 Significant Drawbacks

Whilst the time required to become proficient with the technologies involved could be seen as a drawback, because of how it resulted in fairly slow progress on the final solution early in development, this was an expected price and the quality of the solution by far supports the decision to spend so much time learning about the technologies.

10.2.1 Choice of Wireless Sensor Network

The one technology that dominated the familiarisation process for the wrong reasons was the WSN and this is directly attributed to the poor choice of WSN platform − the Tmote Invent. The Invent was chosen because of the availability of nodes already owned by the school, however in hindsight the project would have benefited from first doing background research into the existing platforms and subsequently choosing one more appropriate, such as the mica-family, rather than settling for what was convenient.

The Invent is a well designed and capable hardware platform, however after a corporate takeover its new owners opted to drop all support of not only the hardware but also the drivers and software utilities required to develop for it. The result is that the current version of TinyOS (2.1) does not ship with the required utilities to build applications for the platform. TinyOS 2 is *"a clean slate redesign and re-implementation of TinyOS"* as the developers thought *"that many aspects of 1.x strain to meet requirements and uses that were not foreseen when it was designed and implemented"* and that it had *"fundamental limitations"* [77] so using TinyOS 1 to overcome the lack of support in 2 was not an enticing prospect. Instead help was elicited from the tinyos-help mailing list and it was discovered that Urs Hunkeler [78] at l'École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland has created a partial port of the Invent's drivers to TinyOS 2, to the extent that applications can be built and make use of the LEDs and the light sensor.

Developing for the Invent also required the use of the XubunTOS virtual machine [79] due to the otherwise draconian specificity of particular *versions* of packages required by the development environment. This arrangement was hardly ideal and substantially hampered progress by logically differencing the codebase between 2 machines.

Managing to overcome these infractions and build a functioning example use case implementation is proof of what perseverance and an active developer community can achieve in the face of adversity and corporate abandonment.

10.3 Future Directions

Cross-reality is an area of substantial interest, currently more so academically than commercially but as the concept and technology evolve there will be greater commercial adoption where the field promises interesting prospects. The existence of the MPEG-V project is proof of this interest and this project is a proof-of-concept of MPEG-V as it has designed, developed and tested a simpler version of the framework that MPEG-V proposes.

Future directions for this project would almost certainly be in relation to MPEG-V, possibly to use this project as a basis for development of an MPEG-V compliant framework when the standard becomes ratified.

10.4 Concluding Remarks

Cross-reality is an interesting and extremely promising area of innovation that has potential applications in many fields. The toolkit created by this project proves the ability of a standardised framework to allow developers to easily and quickly create cross-reality environments by providing an easy way to integrate wireless sensor networks, actuators and virtual worlds by abstracting over the underlying communication and mapping required.

The success of this project reinforces the validity of the proposed MPEG-V standard and future advancements will surely involve the ratification of the standard and subsequent implementations of it.

Appendix A

Testing Summary

Testing represents a substantial part of the project because it is impossible to extensively test the toolkit without going to the lengths of actually deploying it to an example use case scenario involving real WSN and actuator hardware and a real virtual world simulation, which is no small undertaking.

Whilst the hardware components that are an integral part of a cross-reality system could in theory be simulated for testing purposes, and in fact applications are provided with this submission to do just this to a certain extent, no software simulation can completely faithfully behave in the same manner as the real hardware. In particular one of the aims of the toolkit is to use knowledge of and experience with the development environments and methodologies associated with the hardware to better tailor the toolkit's interactions with them. Without having hands-on experience working with the hardware through prototypes and an example use case implementation it would be impossible to achieve this.

As such testing was performed throughout the entire design and development process, essentially starting as soon as software and hardware prototyping began, rather than only toward the end of the project once the development of the final solution had begun or was complete as is common of the traditional waterfall model of software development.

A.1 Overview of Testing

Two Java programs were developed for performing simple tests of the WSN-to-VW and VW-to-Actuator functionality of the toolkit without having to make use of any WSN nodes, actuators or virtual world server software. All that is required is for the Control Panel application to be running, with a connection to a database, and for the interfaces to be visible and accessible from the computer running the test programs. The design and usage of these is discussed in A.2.

In addition to these software utilities, the majority of the testing was performed using an implementation of the example use case. Selection of this use case is discussed in 8.1 and it is explained in detail in A.3.

A.2 Software Testing

Two Java programs were created for testing the toolkit; SensorSim and ActuatorSim. In addition the TinyOS simulator TOSSIM was used for some testing as an alternative to deploying the application being developed onto physical Invent nodes.

A.2.1 SensorSim

This is a Java program that simulates a reading being reported from a WSN to the Control Panel via XML-RPC and the WSN-to-VW interface. Its usage is as follows;

```
java SensorSim <address of interface> <node id> <sensor type>
                                <value>
```

where `<address of interface>` is the address of the WSN-to-VW interface (as displayed in the node management tab of the Control Panel GUI), `<node id>` is the id of the node that this simulated reading is to originate from, `<sensor type>` is the type of sensor that this simulated reading is to have been read from and `<value>` is the (unscaled) reading to have come from the sensor. So for example, if one wanted to simulate a node with id of 'node7' sending a reading to the Control Panel via the WSN-to-VW interface from its temperature sensor with a value of 42, the syntax for using SensorSim to achieve this would be (assuming that the Control Panel XML-RPC server is running on port 8080);

```
java SensorSim http://controlpanelserver.com:8080 node7
                        temperature 42
```

The outcome of the XML-RPC request to the Control Panel is printed to the terminal; true if the request was successful and false otherwise. Just as if this was a real reading being sent from a real WSN, output is also generated and displayed to the Control Panel GUI terminal.

SensorSim Example Usage

An example use of SensorSim is depicted by the following figures A1 through A.3. Figure A1 shows the command line syntax used to invoke SensorSim, figure A.2 shows the response to this request in the GUI's output terminal and figure A.3 shows the response in the OpenSim simulation (more details in A.3.5) upon receiving this request, forwarded through the Control Panel. In this example the Control Panel already knew about the node and the sensor, and the sensor had been calibrated, so a scaled reading was forwarded to OpenSim.

```
[root@cjd44shproj bin]# java SensorSim http://cjd44shproj
    .cs.st-andrews.ac.uk:8080 RedLamp light 254
true
```

Figure A.1: Syntax used to invoke SensorSim example.

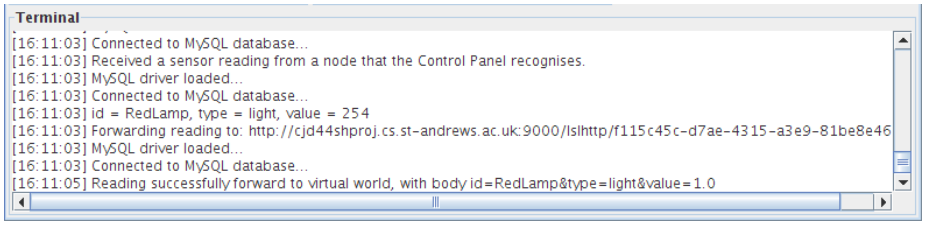

Figure A.2: Output from GUI terminal in response to SensorSim example.

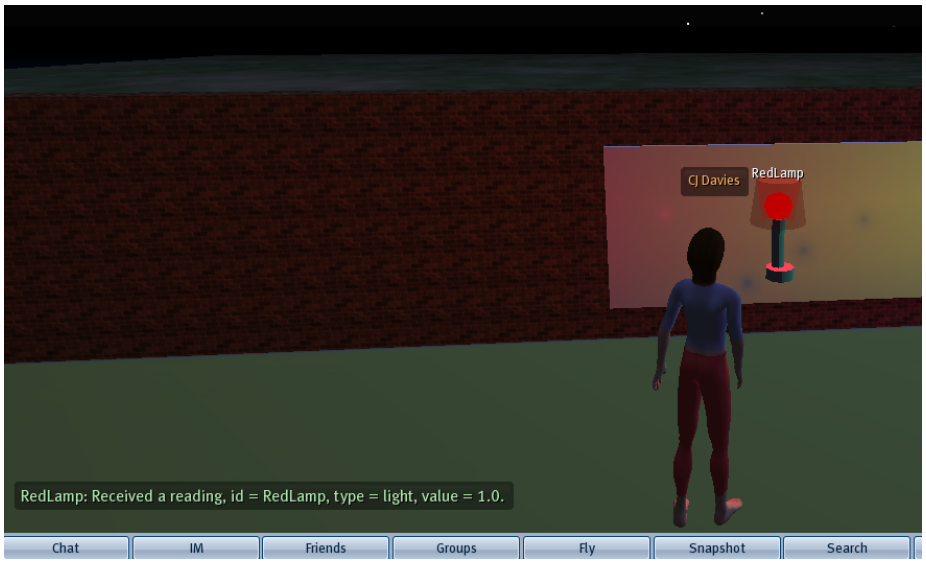

Figure A.3: Reaction in OpenSim of SensorSim usage.

A.2.2 ActuatorSim

This is another Java program, similar to SensorSim, but which simulates an actuator command being sent from an object in a virtual world to the Control Panel via HTTP POST and the VW-to-Actuator interface. Its usage is as follows;

```
java ActuatorSim <address of interface> <actuator id> <action>
```

where similar to SensorSim, `<address of interface>` is the address of the VW-to-Actuator interface (as displayed in the actuator management tab of the Control Panel GUI), `<actuator id>` is the id of the actuator that the command is destined for and `<action>` is the action string that contains details of what the actuator is to do, in the platform-specific format defined by the particular developer.

ActuatorSim Example Usage

An example use of ActuatorSim is depicted by the following figures A4 through A6. Figure A4 shows the command line syntax used to invoke the call of ActuatorSim, figure A5 shows the output from the Control Panel's GUI terminal upon receiving this request and figure A6 shows the response from the example actuator server (more details in A.3.6) upon receiving the request, forwarded through the Control Panel.

```
[root@cjd44shproj bin]# java ActuatorSim http://
    cjd44shproj.cs.st-andrews.ac.uk:8081 2 on
id=2&action=on
Content-type = [application/x-www-form-urlencoded]
Host = [cjd44shproj.cs.st-andrews.ac.uk:8081]
Content-length = [14]
Connection = [keep-alive]
User-agent = [Java/1.6.0_24]
Accept = [text/html, image/gif, image/jpeg, *; q=.2, */*;
    q=.2]
```

Figure A.4: Syntax used to invoke ActuatorSim example usage.

A.2.3 TOSSIM

TOSSIM is a TinyOS node simulator, which compiles directly with TinyOS code, simulates the network stack at the bit level and scales to thousands of nodes [80]. It can be used to test TinyOS applications without any physical

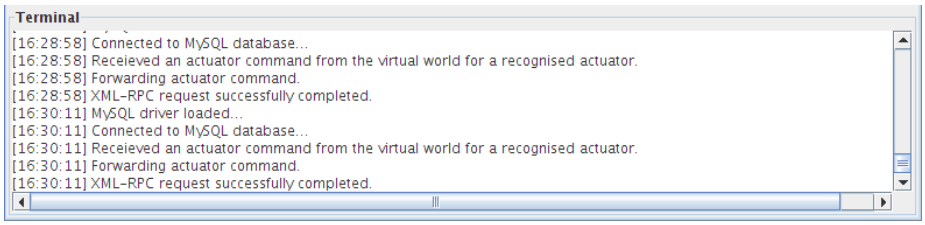

Figure A.5: Output from GUI terminal in response to ActuatorSim usage.

```
[root@cjd44shproj bin]# java ActuatorRPCServer 8083
Added ActuatorRPCServer to the mapping at
    ActuatorRPCServer.
MethodServer running on 8083...
XML-RPC server running on cjd44shproj/138.251.204.10:8083
id: 2
action: on
Red light switched on (or already on).
```

Figure A.6: Reaction in ActuatorRPCServer of ActuatorSim usage.

nodes and allows for easier development and debugging as it is considerably faster to redeploy an application to TOSSIM than it is to redeploy it to thousands of physical nodes every time a minor change is made to the source code.

TOSSIM was used for some of the early software prototypes created for this project. When first embarking upon the learning process of TinyOS development it made sense to first develop in TOSSIM as this removed all of the difficulties and nuances introduced by working with a physical platform, particularly one like the Tmote Invent which only has sketchy third-party support for the current version of TinyOS.

A.3 Testing with Example Use Case Implementation

The use case 1.3.3 chosen for implementation is concerned with using a cross-reality environment to visualise the lighting levels in a building so that a human operator can see where lights have been left on and turn them off from within the virtual world simulation. This is proposed as being useful for large buildings where walking around the entire building may take prohibitive amounts of time, or require crossing multiple different security or other organisational boundaries, as navigating an avatar within a virtual world simulation avoids both of these issues. In addition, an avatar can zoom out of a simulation to see much more of the building in a single view than somebody walking about the real building can

– the simulation can even intentionally omit walls or ceilings, or render them as translucent to allow easier visualisation of where a light is on.

A.3.1 Platforms Used by Use Case

- The Tmote Invent platform from Moteiv [71] was chosen to play the role of the WSN. These motes run the TinyOS operating system [73] for which applications are programmed in the nesC language [72]. See figure A.7 for an example deployment of Tmote Invent nodes, with one basestation node connected directly to the laptop computer and 2 sensing nodes that communicate with each other and the basestation node using short range node-to-node radio.

Figure A.7: Tmote Invent nodes in deployment

As discussed in 10.2.1 this was in hindsight a poor choice of WSN platform, as it no longer has any official hardware or software support and can only be used in a limited capacity by use of an incomplete and unofficial port of the necessary drivers by Urs Hunkeler at l'École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland [78].

Development of the nesC applications was conducted within an instance of the XubunTOS virtual machine [79], due to the difficulty of setting up the required development environment manually in the host distribution.

- The Phidgets USB interface platform was chosen to fill the position of actuators. This decision was largely due to previous experience with the platform and the presence of available hardware already within the department. This platform allows a great range of devices to be connected and controlled from a computer's USB port with well written interfaces available for numerous programming languages, including Java.
- OpenSim was chosen as the virtual world server software. OpenSim is discussed in more detail, including why it is a good choice for this implementation, in 3.1.

A.3.2 Overview of Use Case Architecture

The overall architecture of the use case implementation is shown by figure A.8.

Two computers are used, one desktop and one laptop. The desktop runs the Control Panel program, hosts the MySQL database, runs the OpenSim server and is directly connected to the Phidgets interface kit. The laptop computer runs the virtual world viewer software (Hippo OpenSim Viewer) and is directly connected to the Tmote Invent basestation node.

The construction of this use case required several components to be separately implemented and to interact with the toolkit/Control Panel;

1. A WSN of Tmote Invent nodes, comprising;
    - several sensing nodes which are not connected physically to a computer, installed with a program that periodically takes readings from the Invents' light sensors and sends this over short range node-to-node radio to the basestation node
    - exactly one basestation node which is connected physically to a computer, installed with a basestation program that listens for incoming messages on node-to-node radio and forwards any messages that it receives to the computer that it is connected to by a serial forwarding service.
2. A Java program running on the same computer that the WSN basestation node is connected to, to listen for messages coming from the basestation node over the serial forwarder, extract the important information from them (eg. the id of the node, the type of reading and the value of the reading) and send this information via XML-RPC to the Control Panel, using the WSN-to-VW interface.
3. A simulation of the John Honey building in OpenSim, including;
    (a) primitives representing lighting fixtures that are scripted in the Linden Scripting Language (LSL) to respond appropriately (eg. changing the amount of light that they emit) according to messages from the WSN

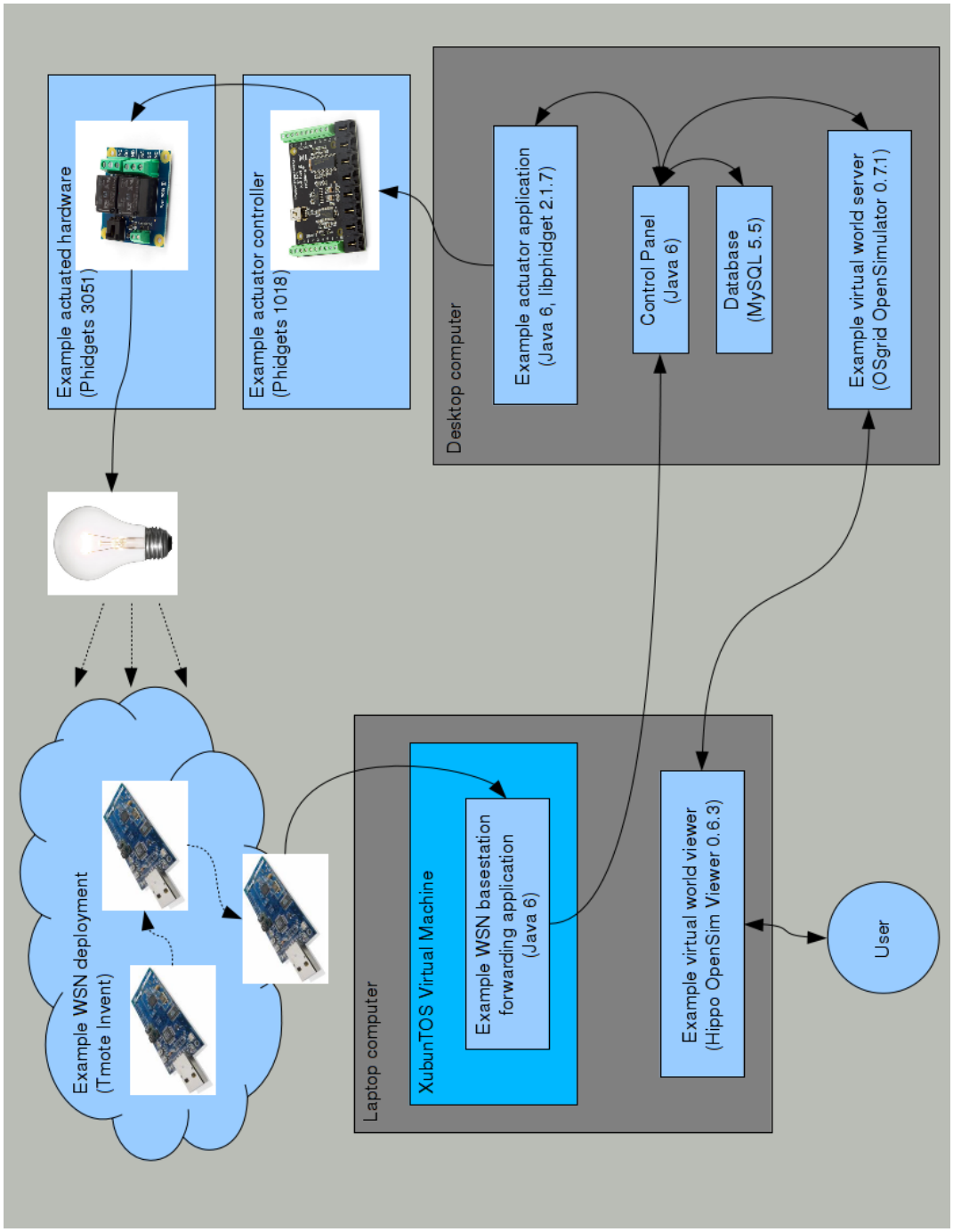

Figure A.8: Overview of example use case implementation architecture.

(b) primitives representing lighting fixtures or switches that are scripted to respond appropriately (eg. sending commands to actuators) when interacted with by an avatar.

4. Apparatus capable of switching one or more lighting fixtures on and off by being controlled by the Phidgets connected to a computer.

5. A Java program connected to the same computer as the lighting apparatus to receive actuator commands from the Control Panel, extract the important information and forward this to the Phidgets and apparatus.

A.3.3 WSN Programs

Two programs are used by the WSN; one for the sensing nodes and another for the basestation node.

Sensing Node Program

The sensing nodes use a program called `PhotoToRadio` which comprises the files `PhotoToRadio.h`, `PhotoToRadioC.nc` and `PhotoToRadioAppC.nc` as per the TinyOS naming convention.

As expected `PhotoToRadio.h` is simply a header file. `PhotoToRadioC.nc` is the implementation of the program itself and `PhotoToRadioAppC.nc` is the 'configuration' of the program.

This program takes readings from the Invent's light sensor at periodic intervals, then sends it using the short range mote-to-mote radio (see figure A10) using a message structure defined in the header file (see figure A9).

```
typedef nx_struct PhotoToRadioMsg {
        nx_uint16_t nodeid;
        nx_uint16_t photoreading;
} PhotoToRadioMsg;
```

Figure A.9: Message structure defined for node transmission of light sensor readings.

Basestation Node Program

The basestation program used is provided by the TinyOS distribution source tree and is found at `$TOSROOT/tinyos-2.1.0/apps/BaseStation`. This is a simple program that just listens on the node's radio interface for incoming messages and whenever it receives one it forwards it to the computer that it is connected to by USB using an onboard serial to USB converter.

```
event void Photo.readDone(error_t result, uint16_t data)
    {
        counter++;
        call Leds.set(counter);

        if (!busy) {
            PhotoToRadioMsg* ptrpkt = (
                PhotoToRadioMsg*)(call Packet.
                getPayload(&pkt, NULL));
            ptrpkt->nodeid = TOS_NODE_ID;
            ptrpkt->photoreading = data;

            if (((lastTransmit + ((lastTransmit/100)
                *50)) > data) || ((lastTransmit - ((
                lastTransmit/100)*50)) < data)) {
                lastTransmit = data;
                if (call AMSend.send(
                    AM_BROADCAST_ADDR, &pkt,
                    sizeof(PhotoToRadioMsg)) ==
                    SUCCESS) {
                        busy = TRUE;
                }
            }
        }
}
```

Figure A.10: nesC code responsible for transmitting light readings over node-to-node radio.

A.3.4 WSN Java Program

The Java program `PhotoMsgReaderToXMLRPC.java` listens on the USB port that the basestation node is connected to for incoming messages. When it receives one it parses the message using an implementation of the Message Interface Generator (mig) which automatically generates the file `PhotoToRadioMsg.java` which allows easy access to the structure of the message from within Java (see 8.4). It extracts the source node's id and the reading then sends these, along with the sensor type (light) by an XML-RPC request to the URI that the WSN-to-VW interface is running upon. This program makes use of the Apache XML-RPC library for facilitating XML-RPC client actions.

A.3.5 OpenSim Simulation

A simulation of the entire John Honey building was built in OpenSim (see figure A.11). Within this simulated building are a number of desk lamp primitives

arranged beneath headings of 'augmented virtuality' and 'augmented reality' (see figure A.12).

The augmented virtuality lamps are scripted with a modified version of ObjectDNS (as discussed in 8.3) so that they obtain an externally accessible HTTP URI from the OpenSim HTTP server and report this URI to the related ObjectDNS PHP script (also running on the desktop computer via an instance of Apache) which updates these URIs in the Control Panel database; this saves copying the long pseudorandom strings into the Control Panel's manual node add feature by hand. When these augmented virtuality lamps receive a HTTP POST from the Control Panel, they extract the `<value>` and set their light level to this value (see figure A13).

The augmented reality lamps are scripted (see figure A.14) so that when an avatar 'touches' them, they send an HTTP POST request to the Control Panel, where the `action` is either `on` or `off`, depending upon the current state of the lamp in the simulation. The lamps also change their light level in the simulation between 100% and 0% to visualise their state with the corresponding state in the real world. Because the action sent is explicitly 'on' or 'off', rather than 'switch' or 'toggle', the state of illumination of the augmented reality lamps can never become the opposite of the real lamps.

A.3.6 Actuator Hardware and Java Program

A PhidgetInterfaceKit 8/8/8 (p/n 1018) [81] is used to interface the desktop computer with a Dual Relay Board (p/n 3051) [82]. The Dual Relay Board is mounted inside an ABS project box, which has 2 standard bayonet light bulb sockets on top and a standard C14 switched mains socket ('IEC'/'kettle') on the side. A heat-shrinked cable connects the Dual Relay Board inside the box to the PhidgetInterfaceKit outside the box and subsequently to the desktop computer via USB. This allows the desktop computer to independently switch each light bulb socket on and off at will.

The Java program `ActuatorRPCServer` provides an example implementation of an actuator controller that implements the VW-to-Actuator interface, by running an XML-RPC server with a method `control(String id, String action)`. With this example the id field differentiates between the 2 lightbulbs and the action field contains either 'on' or 'off' to control the lightbulbs.

A.3.7 Summary of Components

Figure A.15 is a visual representation of the different components A.3.3 to A.3.6 and how they interact with each other. Omitted from this diagram is the MySQL database that the Control Panel accesses and the virtual world viewer software used by the user.

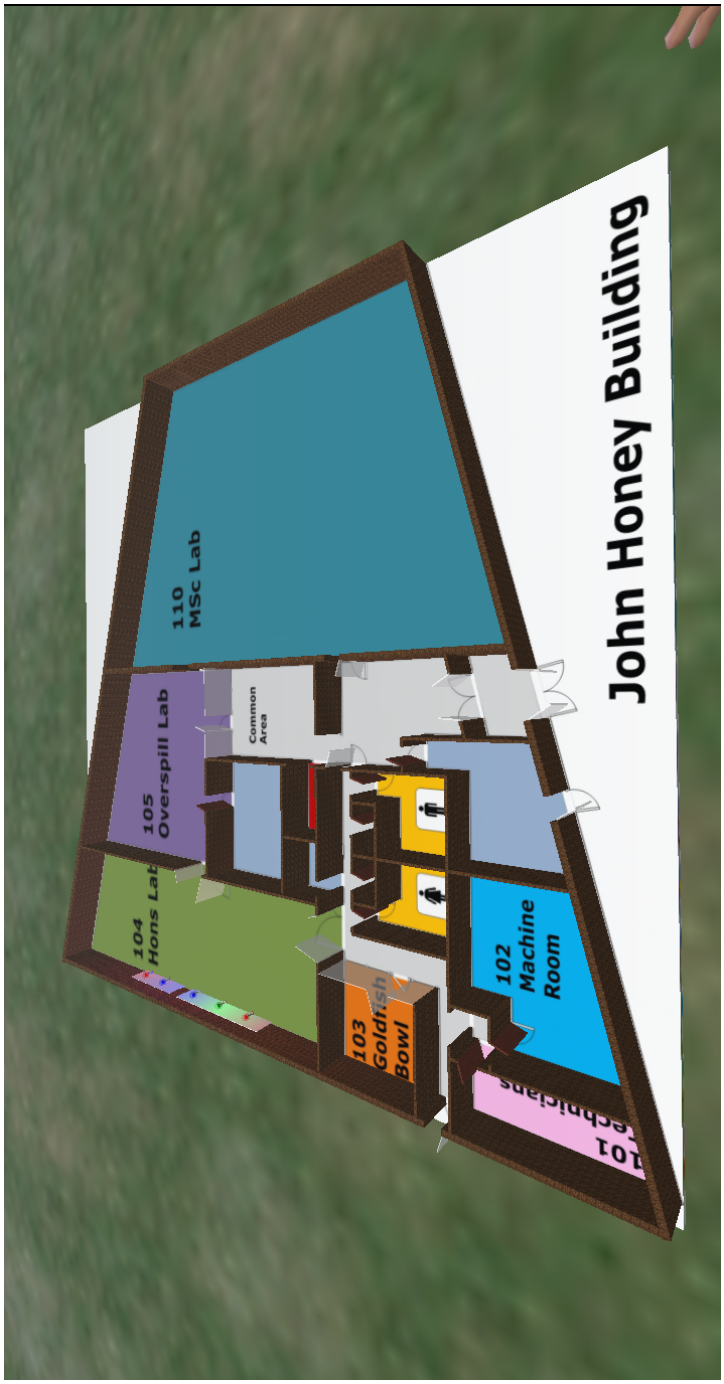

Figure A.11: Simulation of the John Honey building in OpenSim

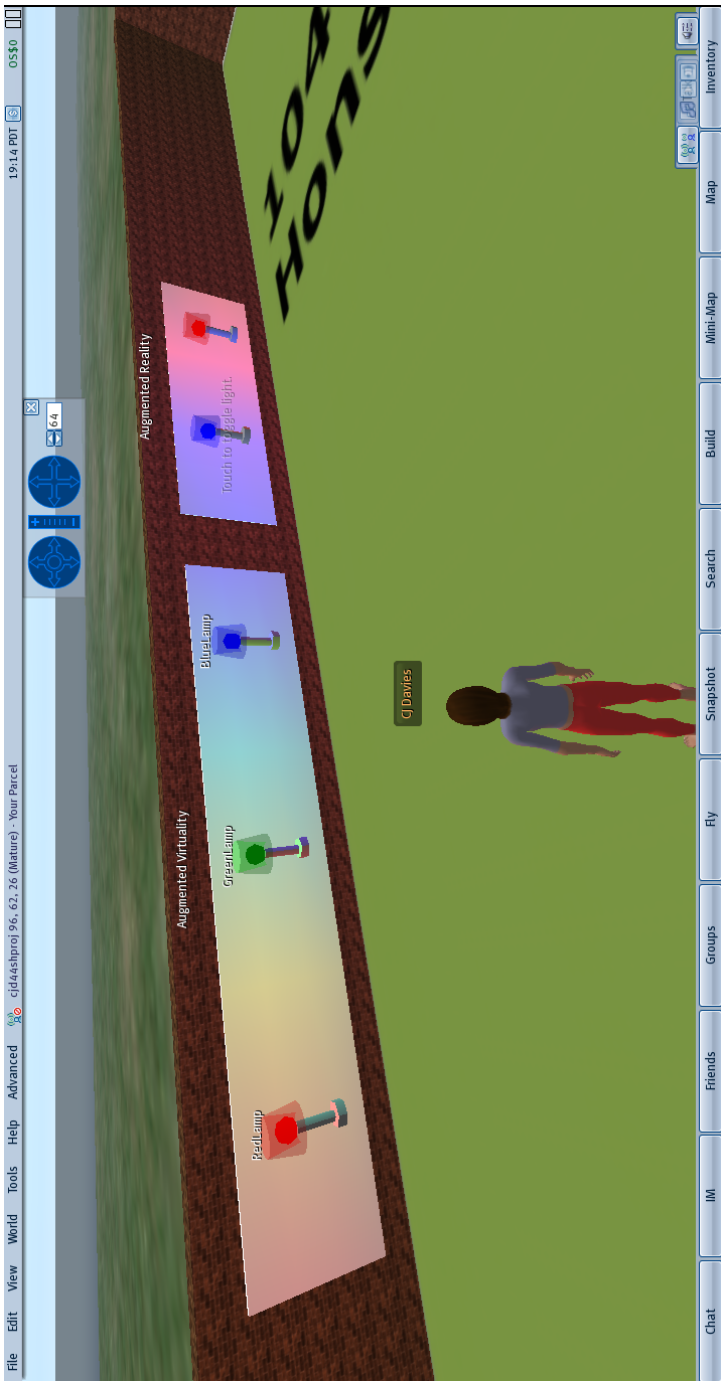

Figure A.12: Augmented virtuality and augmented reality desk lamp primitives in OpenSim

```
//if the primitive receives a HTTP POST...
else if (method == "POST") {

  //POST is in the form "id=21&type=light&value=127"

  //first split up the POST body into the 3 different name/
      value pairs
  list ampersandSeparator = ["&"];
  list spacers = [];
  list nameValuePairs = llParseString2List(body,
      ampersandSeparator, spacers);

  //now split the "id=21" name/value pair into it's two
      parts
  list equalsSeparator = ["="];
  list idNameValuePairs = llParseString2List(llList2String(
      nameValuePairs,0), equalsSeparator, spacers);

  //now do the same for the "type=light" name/value pair
  list typeNameValuePairs = llParseString2List(llList2String
      (nameValuePairs,1), equalsSeparator, spacers);

  //and finally the "value=127" name/value pair
  list valueNameValuePairs = llParseString2List(
      llList2String(nameValuePairs,2), equalsSeparator,
      spacers);

  llSay(0, "Received a reading, id = " + llList2String(
      idNameValuePairs,1) + ", type = "
      + llList2String(typeNameValuePairs,1) + ", value = "
      + llList2String(valueNameValuePairs,1) + "." );

  llSetPrimitiveParams([PRIM_POINT_LIGHT,TRUE,
                        <1.0,0.0,0.0>,  // light color vector
                            range: 0.0-1.0 *3
                        llList2Float(valueNameValuePairs, 1),
                        10.0,           // radius
                            (.1-10.0)
                        0.6 ]);         // falloff
                            (.01-1.0)

  llHTTPResponse(id,200,"Data received.");
}
```

Figure A.13: Extract of LSL of augmented virtuality lamps, which changes the intensity of the light emitted by the lamp in accordance with the new value received from the Control Panel via HTTP POST.

```
touch_start(integer total_number) {

  if (light_s == TRUE) {
    llHTTPRequest("http://cjd44shproj.cs.st-andrews.ac.uk
        :8081/"[HTTP_METHOD,"POST",HTTP_MIMETYPE,"application
        /x-www-form-urlencoded"], "id=1&action=off");
    llSetPrimitiveParams([PRIM_POINT_LIGHT,TRUE,
                          <0.0,0.0,1.0>,  // light color
                             vector range: 0.0-1.0 *3
                           0.0,            // intensity
                              (0.0-1.0)
                           10.0,           // radius
                              (.1-10.0)
                           0.6 ]);         // falloff
                              (.01-1.0)
    light_s = FALSE;
    }

    else if (light_s == FALSE) {
      llHTTPRequest("http://cjd44shproj.cs.st-andrews.ac.uk
          :8081/"[HTTP_METHOD,"POST",HTTP_MIMETYPE,"
          application/x-www-form-urlencoded"], "id=1&action=
          on");
      llSetPrimitiveParams([PRIM_POINT_LIGHT,TRUE,
                            <0.0,0.0,1.0>,  // light color
                               vector range: 0.0-1.0 *3
                             1.0,            // intensity
                                (0.0-1.0)
                             10.0,           // radius
                                (.1-10.0)
                             0.6 ]);         // falloff
                                (.01-1.0)
      light_s = TRUE;
    }
}
```

Figure A.14: Extract of LSL of augmented reality lamps, which send actuator commands to the Control Panel using HTTP POST and toggle their own light levels in accordance.

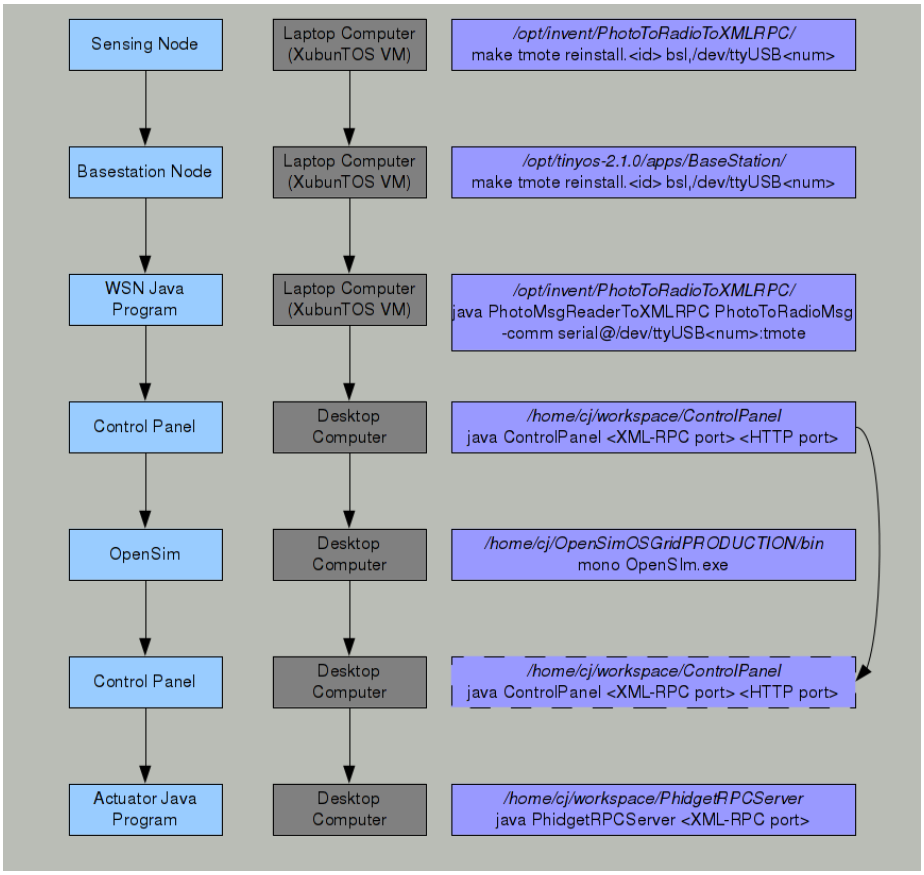

Figure A.15: Detailed component breakdown of example use case implementation architecture

Figure A.16 shows this apparatus with 2 standard energy-saving lightbulbs connected, figure A.17 shows the apparatus with different coloured gels placed over the lightbulbs so as to more easily determine which action in the simulation affects which lightbulb in the real world, and A.18 shows some of the internal wiring of the apparatus.

The Java program `PhidgetRPCServer.java` runs an XML-RPC server on the port passed to it as a command line argument. When the Control Panel sends an XML-RPC request to this server, it is parsed and the appropriate relay is switched on or off, causing the appropriate lightbulb to turn on or off.

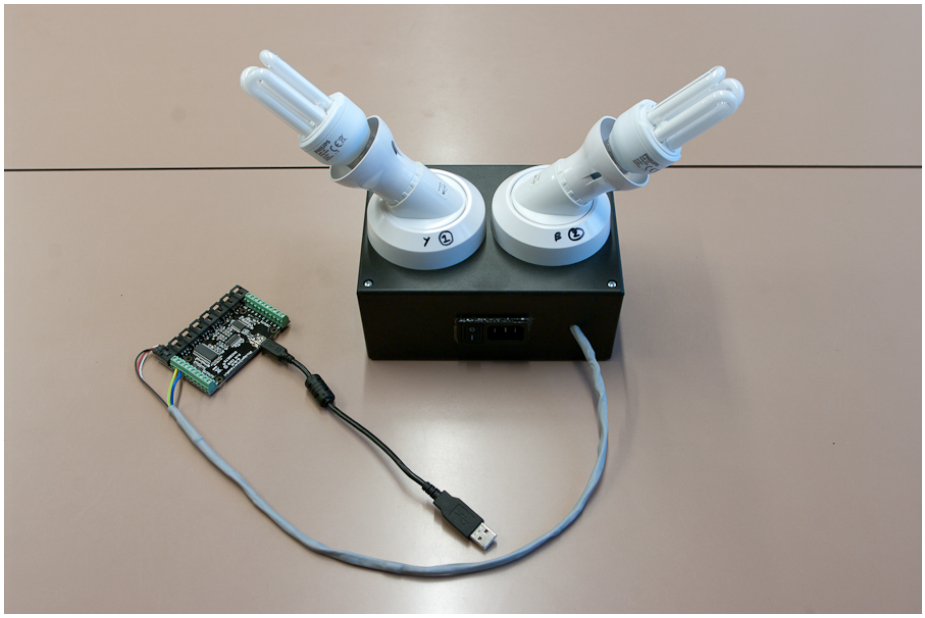

Figure A.16: Actuator controlled lighting appliance.

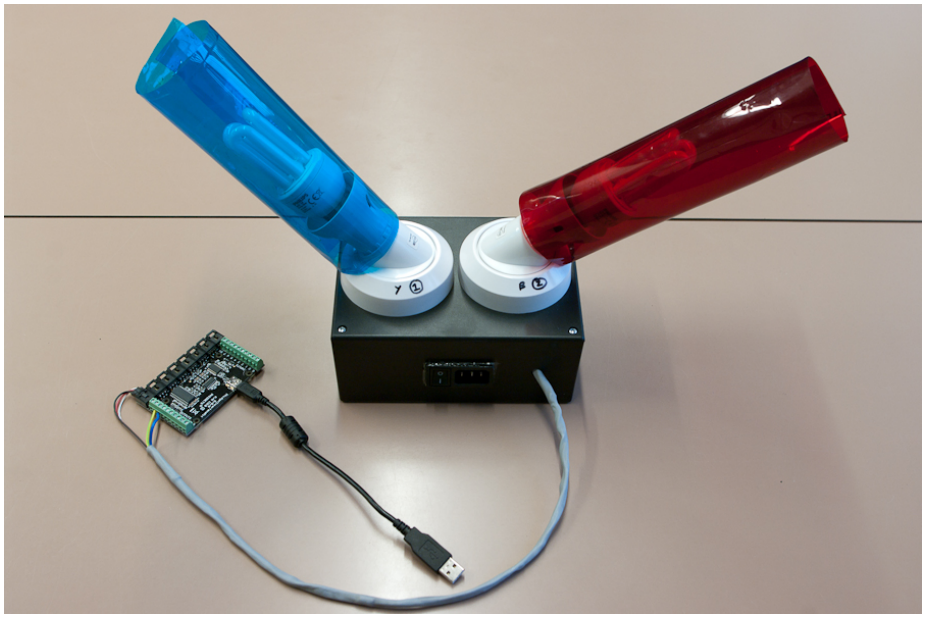

Figure A.17: Actuator controlled lighting appliance with coloured gels.

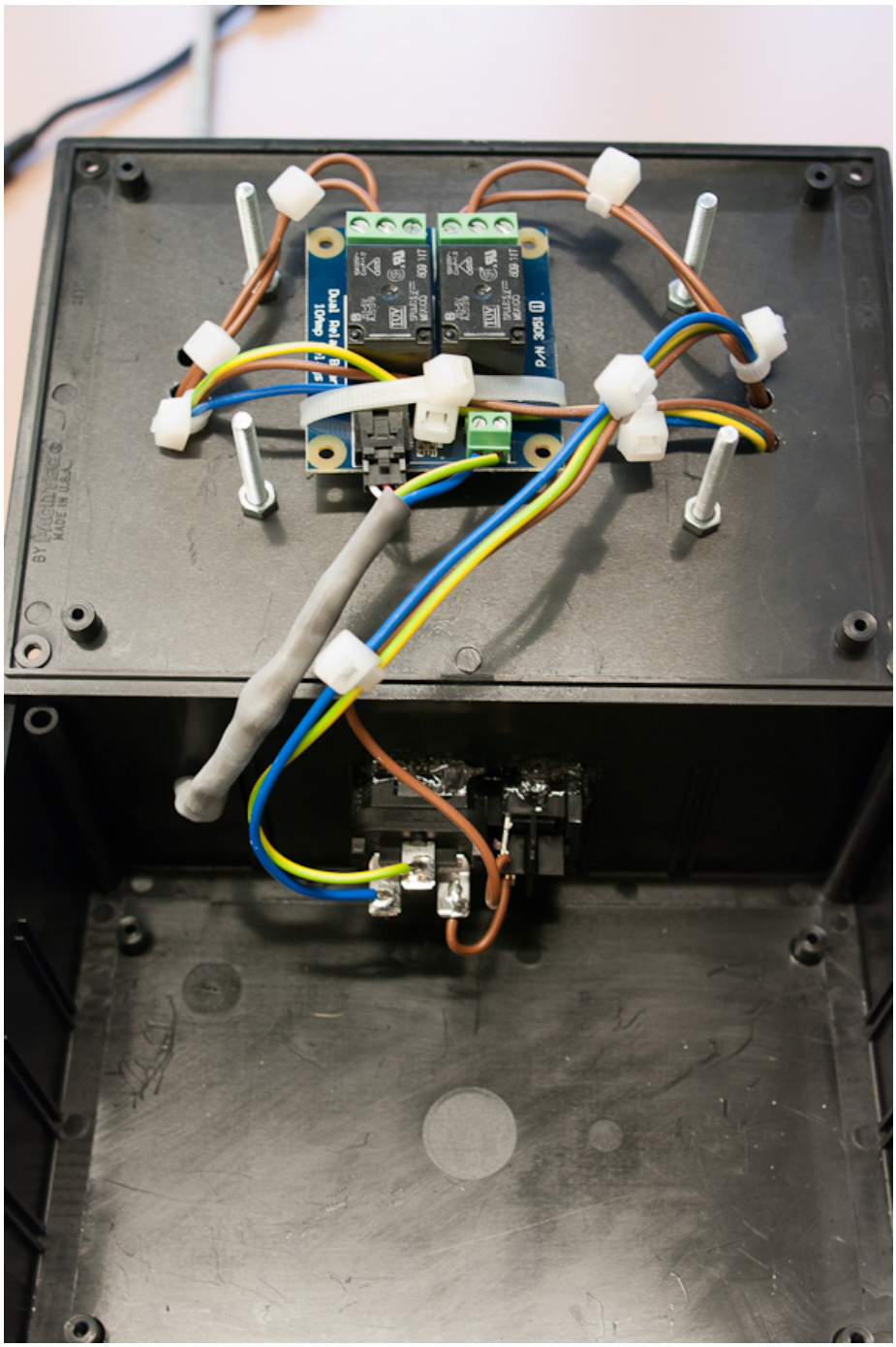

Figure A.18: Some of the internal wiring of the actuator controlled lighting appliance, showing the Phidgets 3051 Dual Relay Board at its heart.

A.3.8 Operation of Example Use Case Implementation

When all components of the implementation are running, the readings from the Tmote Invents' light sensors directly controls the light intensity of the augmented virtuality desk lamp primitives in the OpenSim simulation. An avatar touching the augmented reality desk lamps in the simulation directly controls the lightbulbs on the light box. By bringing one or more of the Tmotes close to the lightbulbs of the lightbox, the control of one or both of the lights from the augmented reality desk lamps will cause the augmented virtuality desk lamps to increase/decrease their intensity as well.

Throughout all of this, the Control Panel can be used to add/remove nodes/sensors/actuators, pause and unpause forwarding and provide diagnostic output to its terminals explaining what is happening as each communication passes through it on its way to/from the virtual world.

Appendix B

Status Report

The toolkit is complete with regards to its original primary and secondary objectives (see 9.1 for discussion) and with regards to the requirements document (see 9.3). Objectives which were not met were limited to some tertiary and emergent objectives (see 9.1.2 and 9.2) and were all either deemed redundant &/or would have served only for testing purposes.

The toolkit is available as the `CrossingReality` package, see Appendix E for full details and Appendix C for usage instructions.

The most prominent contribution of this project is the toolkit that allows the discrete technologies of wireless sensor networks, actuators and virtual worlds to be quickly and easily brought together to form a cross-reality environment. Highlight should be given to the achievement of attaining the required development proficiency with these different technologies, which allowed the toolkit to be implemented in such a successful manner. Further highlight should be given to the range of different deployment architectures that are supported, in terms of the number and arrangement of physical computers used in the system, which computers the different components are running on and which computers the WSN basestation node and actuators controller are connected to. Likewise highlight should be placed on the platform-independent manner in which the toolkit communicates with the WSN, actuators and virtual world server software, which ensures that the maximum number of different platforms are supported.

The example use case implementation also deserves attention, as it proves the suitability and success of the toolkit in a real world deployment scenario and represents a substantial development commitment.

There are no examples, either commercial or academic, of a toolkit such as this having been created before, however the development of the MPEG-V standard indicates that there is considerable interest in wider development of such toolkits. The success of this project proves the possibility and worth of implementations of the MPEG-V standard once it is ratified (see 10.3 for further discussion on future directions for this work).

Appendix C

User Manual

This appendix provides instructions on how to make use of the toolkit. For listings of where particular files are located, please see Appendix E.

C.1 Toolkit Usage

To make use of the toolkit, the following steps must be completed;

1. Setup database
2. Deploy Control Panel application
3. Implement interfaces

C.1.1 Setup Database

The Control Panel uses a MySQL database to store details about nodes, sensors, actuators and relationships between them. This database is not setup automatically by the Control Panel program so it must be setup manually, by issuing the following commands into MySQL to create the database with the 3 tables that it uses.

```
CREATE DATABASE controlpaneldb;
```

(The database does not have to be called `controlpaneldb`*, this is just an example; it can be called anything as long as this name is then used when inputting the database details into the relevant section of the Control Panel GUI.)*

```
CREATE TABLE nodes (
        id VARCHAR(50) NOT NULL,
        address VARCHAR(100),
        UNIQUE (id)
);
```

```
CREATE TABLE sensors (
        id VARCHAR (50) NOT NULL,
        type VARCHAR (50) NOT NULL,
        max INTEGER,
        min INTEGER,
        UNIQUE INDEX idtype (id, type)
);
```

```
CREATE TABLE actuators (
        id VARCHAR (50) NOT NULL,
        address VARCHAR (100),
        UNIQUE (id(
);
```

C.1.2 Deploying Control Panel Application

The central Control Panel application is provided as `ControlPanel.class` (see Appendix E for full software listings) and is run by issuing the following command;

```
java ControlPanel <XML-RPC server port> <HTTP server port
    >
```

`<XML-RPC server port>` specifies the port that the XML-RPC server will run on, to accept sensor readings from a WSN implementing the WSN-to-VW interface. `HTTP server port` specifies the port that the HTTP server will run on, to accept actuator commands from a virtual world implementing the VW-to-Actuator interface. If the ports specified are already in use or otherwise not available, the Control Panel terminal will produce output to this effect recommending to close the application and try again, after checking that the ports are available or choosing different ports.

Once the Control Panel application is running, enter the database address, username and password into the relevant section on the `WSN General/Node Management` tab. The database address must be in the format;

`domain:port/controlpanel`

For example;

`cjd44shproj.cs.st-andrews.ac.uk:3306/controlpaneldb`

(assuming the name of the database is `controlpaneldb` as above).

Then click the `[Update details & (re)connect.]` button. To save the database details so that they are not lost each time the Control Panel is closed, click the `[Save details.]` button.

Once the Control Panel is up and running with an active database connection its operation is self-explanatory thanks to the well labelled interfaces.

C.1.3 Implementing Interfaces

Once the Control Panel application is running and a database connection has been set up, the interfaces can be implemented by WSN and virtual world software wishing to make use of them. Further details about the interfaces, protocols, declarations, etc. are contained in Chapter 7.

WSN-to-VW Interface, WSN side

The address and port that the WSN-to-VW interface is running upon is displayed in the top left of the `WSN General/Node Management` tab in the section labelled `Address of WSN interface`. Any application wishing to implement this interface to send readings from a WSN to a virtual world needs only to send XML-RPC requests to this address where the handler is;

```
ControlPanel
```

and the method is;

```
sensed(String id, String type, int value)
```

where `id` is the unique id of the node sending the reading, `type` is the type of sensor that the reading originates from (eg. light, temperature, etc.) and `value` is the (unscaled) value of the reading.

WSN-to-VW Interface, VW side

At the virtual world itself, objects/primitives or the HTTP gateway must be scripted/programmed ready to receive HTTP POST requests with a body of the form;

```
id=<id>&type=<type>&value=<value>
```

Upon receiving these requests, it is up to each individual user to decide how to have the virtual world respond.

VW-to-Actuators Interface, VW side

The address and port that the VW-to-Actuators interface is running upon is displayed in the top left of the `Actuator Management` tab in the section labelled `Address of actuators interface`. Any virtual world application wishing to implement this interface to send actuator commands from a virtual world to actuators needs only to send HTTP POST requests to this address with a body that matches the syntax;

```
id=<id>&action=<action>
```

where `<id>` is the id of the actuator that this command is destined for and `<action>` is the platform-specific command to be performed by this actuator.

VW-to-Actuators Interface, Actuator side

Any application wishing to receive actuator commands from a virtual world via the VW-to-Actuator interface such that it can control actuators in response to these commands, must run an XML-RPC server where the handler is;

```
ActuatorRPCServer
```

and the method is;

```
control(String id, String action)
```

where `id` is the id of the actuator that this command is destined for and `action` is the platform-specific command to be performed by this actuator. Upon receiving these requests, it is up to each individual user to decide how to have the actuators respond.

C.1.4 Libraries

Libraries used by the Java applications are included in the `Libraries` folder and it is assumed that these are added to the classpath before attempting to make use of the toolkit. Please see Appendix E for a listing of these libraries.

Appendix D

Maintenance Document

Due to the clear definition of the interfaces in 7.4 and 7.5 it should be possible to make changes, even substantial ones, to the underlying functionality of the toolkit without it affecting any other programs that make use of the toolkit, as long as the interface definitions remain the same and achieve the same end functionality.

Because the communications between the WSN, actuators, virtual world server and Control Panel all make use of common and accessible transport technologies (HTTP and XML-RPC) it is even possible to completely change the internal logic of the Control Panel, as far as completely chancing what language it is written in, without having to change any programs that make use of it.

`SensorSim` (A.2.1) and `ActuatorSim` (A.2.2) can be used for testing the compatibility of any modified version of the Control Panel and TOSSIM (A.2.3) can be used to simulate higher WSN loads than `SensorSim` is capable of if this is required.

Appendix E

Software Listings

Software is included in the `code` directory of the submission. There are 3 top-level directories within the `code` directory;

- `toolkit` − contains the toolkit software itself.
- `usecase` − contains the example use case implementation software.
- `libraries` − contains libraries used by the toolkit software and the use-case software.

E.1 `code/toolkit`

- `bin` − compiled binary class files of the `CrossingReality` package.
- `doc` − Javadoc for all classes in the `CrossingReality` package.
  - `index.html` − the root page of the Javadoc.
- `src` − Java source files of the `CrossingReality` package.
  - `ActuatorSim.java`
  - `ControlPanel.java`
  - `MethodServer.java`
  - `SensorIgnoreIdTypeObject.java`
  - `SensorSim.java`

A UML diagram of the `CrossingReality` package is included as figure E.1.

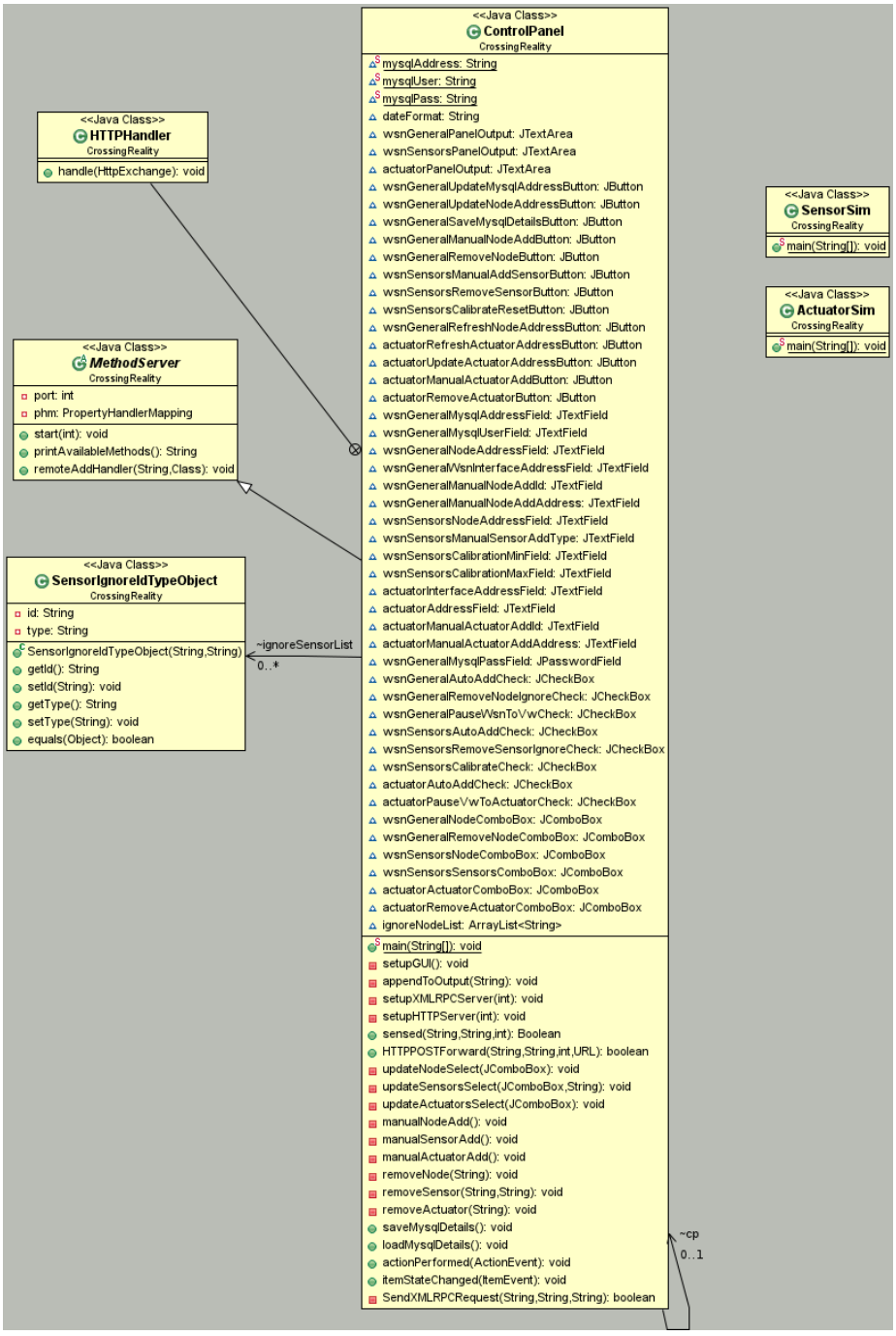

Figure E.1: UML diagram of the `CrossingReality` package.

E.2 code/usecase

This directory contains all of the software used for the example use-case implementation, which was used for the demonstration. This is provided for interest only and the following is only a brief overview of the contents; it is not intended to provide enough information to be able to successfully recreate the example use-case implementation, as this requires considerable development knowledge of WSNs, actuators and virtual worlds.

- `ActuatorRPCServer` – Example implementation of the VW-to-Actuators interface, Actuator side. This program is an XML-RPC server that accepts XML-RPC requests from the Control Panel and uses them to control 2 lightbulbs via the Phidgets hardware.
- `invent` – Tmote Invent source tree. This contains the WSN application that was deployed to the sensing nodes, which periodically takes readings from their light sensors and sends these over node-to-node radio to the basestation node. Also contains the Java program that receives these messages from the basestation node and subsequently forwards them on to the Control Panel via XML-RPC.
- `tinyos-2.1.0` – TinyOS source tree. This contains the basestation application that was deployed to the basestation node, which receives messages from the sensing nodes and passes these to the computer it is connected to.
- `opensimarchive.oar` – The simulation in OpenSim of the John Honey building, with the scripted desk lamp primitives for augmented virtuality and augmented reality. Doors are also scripetd to open/close when touched, which can easily be modified to open/close upon receipt of WSN readings, presumably from an accelerometer or a light break sensor.

E.3 code/libraries

- `apache-xmlrpc-3.1.3` – Used by the Control Panel, the Java program that forwards sensor readings from the basestation node to the Control Panel, and the Java program that controls the Phidgets in response to commands from the Control Panel.
  - `commons-logging-1.1.jar`
  - `ws-commons-util-1.0.2.jar`
  - `xmlrpc-client-3.1.3.jar`
  - `xmlrpc-common-3.1.3.jar`
  - `xmlrpc-server-3.1.3.jar`
- `mysql-connector-java-3.0.17-ga-bin.jar` – Used by the Control Panel.

- `phidget21.jar` – Used by the Java program that controls the Phidgets in response to commands from the Control Panel.

Appendix F

IRC Log

The following is a log of discussion on the official `#opensim-dev` IRC channel on the Freenode IRC server, on the topic of XML-RPC support in OpenSim.

```
11:30 -!- Irssi: Join to #opensim-dev was synced in 1
    secs
11:31 < cj> I have primitives in OpenSim with scripts
    that are controled from outside
                  OpenSim via HTTP POST - can I use XML-
                     RPC instead?
11:31 < cj> I've been searching for a while & keep
    finding conflicting or old
                  information
11:32 < cj> essentially I want to either be able to send
    XML-RPC requests to
                  individual objects, or to a single
                     interface which can then pass them
                     on
                  to the correct object
11:33 < cj> I've found RemoteAdmin, which appears to
    start an XML-RPC server, however
                  it seems to only support specific
                     administrative commands?
11:34 < surrealiz3> cj:
11:34 < surrealiz3> you post to those primitives ?
11:34 < surrealiz3> how can that be ?
11:34 < cj> atm I send HTTP POST to individual primitives
    , using the ObjectDNS scripts
                  on the wiki to keep track of the URL
11:35 < cj> each object calls llrequesturl, registers
    that url in a mysql database,
                  the out-world program then looks up
```

```
                        the url in the database & sends a
                        HTTP
                    POST to the primitive
11:36 < cj> I want to achieve something similar but
    sending XML-RPC requests rather
                    than HTTP POST
11:36 < surrealiz3> i was not aware of that
11:36 < surrealiz3> so opensim registers an http handle
    for each prim ?
11:37 < cj> yup, it runs an HTTP server on the same port
    as OpenSim itself & assigns a
                    randomly generated HTTP URI to any
                        prim that requests it
11:39 -!- RomainFG [~RomainFG@2a01:e35:1393:6ca0:6ef0:49
    ff:fedf:722] has joined #opensim-dev
11:41 < surrealiz3> waht can be invoked in the prim
    through post ?
11:41 -!- Katerchen [~Katerchen@f053090049.adsl.alicedsl.
    de] has joined #opensim-dev
11:42 < surrealiz3> i assume if it supports http xml-rpc
    is possible maybe not implemented
                     atm
11:43 < surrealiz3> seems nice feature gonna check it
    later :)
11:43 -!- Taoki|away is now known as Taoki
11:54 < cj> anything can be invoked on the prim through
    post, it's limited only by
                    your creativity
11:55 < cj> the prim gets the entire POST body, so you
    can process the contents anyway
                    you wish
11:56 < cj> what I'm actually doing is taking sensor
    readings from a Wireless Sensor
                                    Network  & sending
                                       them to prims via
                                       HTTP POST to make
                                       them light up
                    dependig on how much light there is in
                        the real world
11:57 < surrealiz3> seems kool
11:58 < cj> it works okay as it is, but I'd much rather
    have XML-RPC than POST :(
11:58 < cj> found an old mailing list entry that might
    help, going to give it a read
11:58 < surrealiz3> so the body gets parsed by lsl script
     inside prim
```

```
11:58 < Melanie_T> xmlrpc is possible
11:58 < Melanie_T> just badly broken
11:58 < cj> Melanie_T: any info on just how broken?
11:58 < surrealiz3> you would like to expose lsl script
    methods through xml-rpc ?
11:59 < Melanie_T> it would need to be reworked along the
     lines of http-in
11:59 < Melanie_T> currently, xmlrpc will take up one
    http server thread for each request
11:59 < Melanie_T> for it's entire duration
12:00 < Melanie_T> we saw that leads to starvation and
    hangle
12:00 < Melanie_T> hangs
12:00 < cj> I think the idealsituation would be a single
    XML-RPC interface to which I
                    could send requests, from which they
                        would be directed to the applicable
                    prim
12:00 < Melanie_T> http-in uses another approach
12:00 < Melanie_T> and xmlrpc would need to be changed to
     support it
12:00 < Melanie_T> also, grids need to implement an
    xmlrpc receiver prim directory
12:01 < Melanie_T> to track what sim they are in
12:01 < cj> so essentially, it would be a lot of work to
    change it from the current
                    POST setup to an XML-RPC one & even
                        then it wouldn't be as
                    efficient/stable?
12:01 < Melanie_T> and maintain a separsate xmlrpc
    gateway
12:01 < Melanie_T> just like LL does
12:01 < Melanie_T> we at avination maintain such a
    registry and gateway for email()
12:02 < Melanie_T> we didn't do it for xmlrpc because it'
    s very little used in sl
12:02 -!- ter_touch [~tertouch@75-120-148-69.dyn.
    centurytel.net] has quit [Read error:
           Connection reset by peer]
12:02 < Melanie_T> it's very slow, requests can take a
    minute or two and often time out in sl
12:02 < Melanie_T> people stopped using it
12:02 -!- ter_touch [~tertouch@75-120-148-69.dyn.
    centurytel.net] has joined #opensim-dev
12:02 < Melanie_T> so we felt safe to say that with http-
    in available, no one would need
```

```
                    xmlrpc
12:02 < cj> ah, just found the entry on the SL wiki
    saying that XML-RPC is considered
                    'deprecated'
12:03 < Melanie_T> it's deprecated in sl
12:03 -!- core [~core@unaffiliated/core] has joined #
    opensim-dev
12:03 < cj> right, staying with POST is best then
12:03 < Melanie_T> and incomplete in opensim
12:03 < cj> what I'm writing is meant to in theory be
    mostly compatible with sl
12:03 < Melanie_T> then use http-in
12:03 < Melanie_T> and run an objdns server
12:04 < Melanie_T> search the wiki for "object dns"
12:04 < cj> yep, just before you joined the conversation
    I mentioned that I've already
                    implemented the feature using
                        ObjectDNS & HTTP POST
12:04 < Melanie_T> objdns allows you to find a prim's url
12:04 < cj> I was simply investigating the possibility of
     replacing it with RPC
12:04 < cj> but now it seems it's a much better idea to
    leave it as it is!
12:04 < Melanie_T> now you know it's not advisable
12:05 < cj> thankyou very much for all your knowledge :)
12:05 < surrealiz3> ejeje Melanie_T rulez :P
12:06 < Melanie_T> you're welcome
```

Bibliography

[1] Jonathan Bishop. Enhancing the understanding of genres of web-based communities; the role of the ecological cognition framework. *Int. J. Web Based Communities*, 5:4–17, November 2009.

[2] H. Tamura, H. Yamamoto, and A. Katayama. Mixed reality: future dreams seen at the border between real and virtual worlds. *Computer Graphics and Applications, IEEE*, 21(6):64 –70, nov. 2001.

[3] Mark Wright, Henrik Ekeus, Richard Coyne, James Stewart, Penny Travlou, and Robin Williams. Augmented duality: overlapping a metaverse with the real world. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 263–266, New York, NY, USA, 2008. ACM.

[4] Beth Coleman. Using sensor inputs to affect virtual and real environments. *IEEE Pervasive Computing*, 8(3):16–23, 2009.

[5] Joshua Lifton, Mathew Laibowitz, Drew Harry, Nan wei Gong, Manas Mittal, and Joseph A. Paradiso. Metaphor and Manifestation - Cross-Reality with Ubiquitous Sensor/Actuator Networks. *IEEE Pervasive Computing: Mobile and Ubiquitous Systems*, 8(3):24–33, July-September 2009.

[6] Chip Morningstar and F. Randall Farmer. *The lessons of Lucasfilm's habitat*, pages 273–302. MIT Press, Cambridge, MA, USA, 1991.

[7] Mats Deutschmann, Luisa Panichi, and Judith Molka-danielsen. Designing oral participation in second life – a comparative study of two language proficiency courses. *ReCALL*, 21:206–226, May 2009.

[8] Andrea De Lucia, Rita Francese, Ignazio Passero, and Genoveffa Tortora. Development and evaluation of a system enhancing second life to support synchronous role-based collaborative learning. *Softw. Pract. Exper.*, 39:1025–1054, August 2009.

[9] Andrea De Lucia, Rita Francese, Ignazio Passero, and Genoveffa Tortora. Development and evaluation of a virtual campus on second life: The case of seconddmi. *Comput. Educ.*, 52:220–233, January 2009.

[10] Azilawati Jamaludin, Yam San Chee, and Caroline Mei Lin Ho. Fostering argumentative knowledge construction through enactive role play in second life. *Comput. Educ.*, 53:317–329, September 2009.

[11] Richard Stephen Clavering and Andrew Robert Nicols. Lessons learned implementing an educational system in second life. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 2*, BCS-HCI '07, pages 19–22, Swinton, UK, UK, 2007. British Computer Society.

[12] Joe Geigel. Teaching animation in second life. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, pages 64:1–64:1, New York, NY, USA, 2009. ACM.

[13] Konstantinidis Andreas, Thrasyvoulos Tsiatsos, Theodouli Terzidou, and Andreas Pomportsis. Fostering collaborative learning in second life: Metaphors and affordances. *Comput. Educ.*, 55:603–615, September 2010.

[14] Spencer J. Lee, Edward A. Fox, Gary Marchionini, Javier Velacso, Gonçalo Antunes, and José Borbinha. Virtual dl poster sessions in second life. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '09, pages 473–474, New York, NY, USA, 2009. ACM.

[15] Jennifer A. Polack-Wahl. Seeing data in second life. *J. Comput. Small Coll.*, 24:103–109, June 2009.

[16] Helmut Prendinger, Boris Brandherm, and Sebastian Ullrich. A simulation framework for sensor-based systems in second life. *Presence: Teleoper. Virtual Environ.*, 18:468–477, December 2009.

[17] Boris Brandherm, Sebastian Ullrich, and Helmut Prendinger. Simulation of sensor-based tracking in second life. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, AAMAS '08, pages 1689–1690, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[18] Maurizio de Pascale, Sara Mulatto, and Domenico Prattichizzo. Bringing haptics to second life. In *Proceedings of the 2008 Ambi-Sys workshop on Haptic user interfaces in ambient media systems*, HAS '08, pages 6:1–6:6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[19] Matteo Varvello and Geoffrey M. Voelker. Second life: a social network of humans and bots. In *Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '10, pages 9–14, New York, NY, USA, 2010. ACM.

[20] James Kinicki and Mark Claypool. Traffic analysis of avatars in second life. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '08, pages 69–74, New York, NY, USA, 2008. ACM.

[21] Andrew Lang and Jean-Claude Bradley. Chemistry in second life. *Chemistry Central Journal*, 3(1):14, 2009.

[22] SciLands. Scilands virtual continent. `http://www.scilands.org/`.

[23] ElucianIslands Team. The elucian islands village: An introduction. `http://blogs.nature.com/ub51cd45e/2010/02/03/the-elucian-islands-village-an-introduction`.

[24] Kriti Sen Sharma. Sbes advanced multiscale ct facility. `http://samct.imaging.sbes.vt.edu/news/slate`.

[25] Paul R. Messinger, Eleni Stroulia, Kelly Lyons, Michael Bone, Run H. Niu, Kristen Smirnov, and Stephen Perelgut. Virtual worlds - past, present, and future: New directions in social computing. *Decis. Support Syst.*, 47(3):204–228, 2009.

[26] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Dmitriy Portnov, and Roy Want. Bridging physical and virtual worlds with tagged documents, objects and locations. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 29–30, New York, NY, USA, 1999. ACM.

[27] Paolo Baronti, Prashant Pillai, Vince W. C. Chook, Stefano Chessa, Alberto Gotta, and Y. Fun Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Comput. Commun.*, 30(7):1655–1695, 2007.

[28] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Comput. Netw.*, 38:393–422, March 2002.

[29] Ian Akyildiz. Grand challenges for wireless sensor networks. In *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '05, pages 142–142, New York, NY, USA, 2005. ACM.

[30] Ian Akyildiz and Mehmet Can Vuran. *Wireless Sensor Networks*. John Wiley & Sons, Inc., New York, NY, USA, 2010.

[31] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: a survey. *Comput. Netw.*, 50:2127–2159, September 2006.

[32] Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdhury. A survey on wireless multimedia sensor networks. *Comput. Netw.*, 51:921–960, March 2007.

[33] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Comput. Netw.*, 47:445–487, March 2005.

[34] Ian Akyildiz, Terry Todd, Hossein Mouftah, and Jean-Louis Gauvreau. Wireless mesh networking. In *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '05, pages 223–223, New York, NY, USA, 2005. ACM.

[35] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, 2008.

[36] Conjecture Corporation. What is an actuator? `http://www.wisegeek.com/what-is-an-actuator.htm`.

[37] Feng Xia, Yu-Chu Tian, Yanjun Li, and Youxian Sung. Wireless sensor/actuator network design for mobile control applications. *Sensors*, 7(10):2157–2173, 2007.

[38] Rosula S. J. Reyes, Jose Claro Monje, Marc Ericson C. Santos, Lorlynn A. Mateo, Roma Lynne G. Espiritu, John Vianney Isiderio, Carlos Miguel M. Lacson, and Ray Edwin T. Ocfemia. Implementation of zigbee-based and ism-based wireless sensor and actuator network with throughput, power and cost comparisons. *WTOC*, 9:395–405, July 2010.

[39] Edith Ngai, Yangfan Zhou, Michael R. Lyu, and Jiangchuan Liu. A delay-aware reliable event reporting framework for wireless sensor-actuator networks. *Ad Hoc Netw.*, 8:694–707, September 2010.

[40] Xianghui Cao, Jiming Chen, Chuanhou Gao, and Youxian Sun. An optimal control method for applications using wireless sensor/actuator networks. *Comput. Electr. Eng.*, 35:748–756, September 2009.

[41] Daniel-Ioan Curiac and Constantin Volosencu. Hierarchical lighting control in urban environments based on wireless sensor-actuator networks. In *Proceedings of the 6th WSEAS international conference on Dynamical systems and control*, CONTROL'10, pages 163–166, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[42] Virtual Worlds News. Ibm launches 3-d data center in opensim. `http://www.virtualworldsnews.com/2008/02/ibm-launches-3.html`, February 2008.

[43] VRcontext. Vrcontext-vrcontext: Revolutionizing operations efficiency. `http://www.vrcontext.com`, 2010.

[44] E. Medina, R. Fruland, and S. Weghorst. Virtusphere: Walking in a human size vr hamster ball. *Proceedings of the Human Factors and Ergonomics Society 52nd Annual Meeting, Santa Monica, CA: HFES.*, 2008.

[45] R. Grasset, P. Lamb, and Billinghurst M. Evaluation of mixed-space collaboration. *In Proceedings of ISMAR 2005, pp. 90-99.*, 2005.

[46] J.W. Lin. Enhancement of user-experiences in immersive virtual environments that employ wide-field displays. *Doctoral Thesis, University of Washington, Seattle.*, 2004.

[47] R.A. May. Toward directly mediated interaction in computer supported environments. *Ph.D. thesis, University of Washington.*, 2004.

[48] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. In *IEICE Trans. Information Systems*, volume E77-D, pages 1321–1329, 1994.

[49] Boriana Koleva, Holger Schnädelbach, Steve Benford, and Chris Greenhalgh. Traversable interfaces between real and virtual worlds. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 233–240, New York, NY, USA, 2000. ACM.

[50] Céline Coutrix and Laurence Nigay. Mixed reality: a model of mixed interaction. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 43–50, New York, NY, USA, 2006. ACM.

[51] Marie Kim, Hwang Jae Gak, and Cheol Sig Pyo. Practical rfid + sensor convergence toward context-aware x-reality. In *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences*, pages 1049–1055, New York, NY, USA, 2009. ACM.

[52] Information exchanve with virtual worlds 'context, objectives and use cases'. ISO/IEC JTC1/SC29/WG11 w9424 issued at the 82nd MPEG Meeting in Shenzhen.

[53] Jean H.A. Gelissen (Philips). Summary of mpeg-v. ISO/IEC JTC 1/SC 29/WG 11/N9901, May 2008.

[54] Information technology - media context and control - part2: Control information. ISO/IEC FCD 23005-2, July 2009.

[55] Information technology - media context and control - part3: Sensory information. ISO/IEC FCD 23005-3, December 2009.

[56] C. Timmerer, J. Gelissen, M. Waltl, and H. Hellwagner. Interfacing with virtual worlds. 2009.

[57] Requirements for mpeg-v version 3.2. ISO/IEC JTC 1/SC 29/WG 11/N10498, February 2009.

[58] N. Yankelovich, J. Slott, A. Hill, M. Bonner, J. Schiefer, B. MacIntyre, E. Mugellini, O.A. Khaled, F. Barras, J. Bapst, M. Back, E. Aviles-Lopez, and J.A. Garcia-Macias. Building and employing cross-reality. *Pervasive Computing, IEEE*, 8(3):55 –57, 2009.

[59] Joshua Harlan Lifton. *Dual Reality: An Emerging Medium.* PhD thesis, Massachusetts Institute of Technology, September 2007.

[60] MIT Media Lab. Dual reality lab. http://www.media.mit.edu/resenv/dual_reality_lab/.

[61] Joshua Lifton, Mark Feldmeier, Yasuhiro Ono, Cameron Lewis, and Joseph A. Paradiso. A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 119–127, New York, NY, USA, 2007. ACM.

[62] Yasuhiro Ono, Joshua Lifton, Mark Feldmeier, and Joseph A. Paradiso. Distributed acoustic conversation shielding: an application of a smart transducer network. In *Proceedings of the First ACM workshop on Sensor and actor networks*, SANET '07, pages 27–34, New York, NY, USA, 2007. ACM.

[63] Mathew Laibowitz, Nan-Wei Gong, and Joseph A. Paradiso. Wearable sensing for dynamic management of dense ubiquitous media. *Wearable and Implantable Body Sensor Networks, International Workshop on*, 0:3–8, 2009.

[64] MIT Media Lab. Ubiquitous sensor portals. http://www.media.mit.edu/resenv/portals/.

[65] MIT Media Lab. Doppellab, exploring dense sensor network data through a game engine. http://www.media.mit.edu/resenv/doppellab/.

[66] Developing ar applications with artoolkit. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, pages 305–305, Washington, DC, USA, 2004. IEEE Computer Society.

[67] G. Kurillo, R. Vasudevan, E. Lobaton, and R. Bajcsy. A framework for collaborative real-time 3d teleimmersion in a geographically distributed environment. In *Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on*, pages 111 –118, 2008.

[68] Ian Sommerville. *Software engineering (5th ed.)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[69] Inc. Linden Research. Category:lsl xml-rpc - second life wiki. http://wiki.secondlife.com/wiki/Category:LSL_XML-RPC.

[70] D Moore, M Thome, and K Z Haigh. scripting your world, the official guide to second life scripting.

[71] Moteiv. Tmote invent user guide. http://sentilla.com/files/pdf/eol/tmote-invent-user-guide.pdf.

[72] UC Berkeley WEBS Project. nesc: A programming language for deeply networked systems. http://nescc.sourceforge.net/.

[73] TinyOS. Tinyos home page. http://www.tinyos.net/.

[74] OpenSim. Main page - opensim. http://opensimulator.org/wiki/Main_Page.

[75] Inc. Phidgets. Phidgets inc. - unique and easy to use usb interfaces. http://www.phidgets.com/.

[76] Zetaphor. Object dns - zetaphor's stuff. http://zetaphor.wikidot.com/object-dns.

[77] Philip Levis. Tinyos 2.0 overview. http://www.tinyos.net/tinyos-2.x/doc/html/overview.html.

[78] Urs Hunkeler. Urs hunkeler. http://people.epfl.ch/urs.hunkeler.

[79] TinyOS. Running a xubuntos virtual machine image in vmware player - tinyos documentation wiki. http://docs.tinyos.net/index.php/Running_a_XubunTOS_Virtual_Machine_Image_in_VMware_Player.

[80] Berkeley University of California. Simulating tinyos networks. http://www.cs.berkeley.edu/~pal/research/tossim.html.

[81] Phidgets Inc. Phidgets inc. - unique and easy to use usb interfaces. http://www.phidgets.com/products.php?category=0&product_id=1018.

[82] Phidgets Inc. Phidgets inc. - unique and easy to use usb interfaces. http://www.phidgets.com/products.php?category=9&product_id=3051.