The Visualisation of Adaptive Behaviour in Ubiquitous Computing Experiments

Cormac Hampson

A dissertation submitted to the University of Dublin, Trinity College

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

May 2006

DECLARATION

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Name:	Cormac Hampson
Date:	31st May, 2006

PERMISSION TO LEND AND/OR COPY

Name: Cormac Hampson

Date: 31st May 2006

ACKNOWLEDGEMENTS

Many thanks are due to my supervisor, Dr. David Lewis, for the considerable time spent assisting me on this project, and for all the valuable advice and guidance offered. Likewise, I would like to thank Austin Kenny, Eleanor O'Neill and Kris McGlinn, for all their work and helpful feedback.

This project's inception was assisted greatly by the comments and suggestions of Peter Williams and Dr. Owen Conlan, for which I offer my sincerest gratitude. To all my family and friends, and especially my fellow ubiquitous computing classmates, I would like to thank you for all your support and encouragement throughout the course of this dissertation.

Cormac Hampson
University of Dublin, Trinity College
May 2006

SUMMARY

The creation of new ubiquitous computing systems has been hindered by the considerable economic and logistical factors associated with their development. Hence, 3D simulators (complete with virtual environments and sensors) have been successfully employed so that rapid prototyping and evaluation of experiments can take place in a reliable and efficient manner, before any real world implementation need occur. However, a significant feature lacking in these simulators is an overview visualisation tool that would enable researchers, at a glance, to understand what is happening in their experiment at any moment in time.

The ubiquitous computing simulator developed by the Knowledge and Data Engineering Group (KDEG) allows researchers to integrate their prototype services into a virtual environment, providing a test bed for the evaluation of their context aware applications. The complex nature of the interactions taking place on this platform has meant that debugging new experiments and eliciting salient information from the data can be a slow and difficult process. This dissertation describes the development of a prototype SVG (Scalable Vector Graphics) visualisation tool that responds directly to this need.

The SVG tool visualises a short experiment involving the interaction between a Location-aware Instant Messenger and the 3D simulator, and a detailed evaluation is undertaken. The results presented show that this application provides significant benefits to developers of such experiments, and that future work based on this prototype can lead to the creation of a robust and extensible tool, central to the development of new ubiquitous computing environments.

.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: Introduction

This chapter introduces the dissertation topic and explores the motivation behind the work. It is followed by an examination of the objectives hoped to be achieved by the project, and concludes with a summary of the document structure.

1.1 Motivation

Ubiquitous Computing is an emerging field of research that concerns itself with methods of *"enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user"* [1]. This burgeoning research area has been to the forefront of developing *smart spaces*; environments that have been equipped with sensors that can perceive and react to individuals, without necessitating them to wear any special equipment.

Typically the development process for a new ubiquitous computing system involves equipping a test environment with sensors, evaluating a variety of configurations, and then rolling the system out to its intended location. There are numerous challenges associated with this process, including the difficulty in creating reproducible experiments to evaluate, and the significant issue of turning a small-scale laboratory prototype into a real world implementation. Thus, the complex nature of ubiquitous computing systems has meant that the development of such environments is a very time consuming and expensive process.

Other factors that have exacerbated the slow development of smart spaces include the logistics of finding suitable test locations to equip with sensors, and the relative expense of such equipment. Hence there has been great interest in the development of 3D computer simulations of smart-space environments, so that rapid prototyping and evaluation of ubiquitous computing experiments can take place in a reliable and efficient manner, before any physical implementation need occur. These simulators generally contain virtual versions of real world environments complete with various sensor types. Thus, they have had success in speeding up the development process, reducing costs, and easing the significant logistical issues associated with the evaluation of new ubiquitous computing environments.

This dissertation describes an SVG (Scalable Vector Graphics) [2] visualisation tool that has been developed in order to aid the evaluation and development of experiments in 3D ubiquitous computing simulators. Sensor network events and user activity within the simulator can be correlated with reference to pre-defined goals, and thus this dissertation will establish whether post-hoc 2D visualisation of the experiment can increase the efficiency of developing scenarios, collating user feedback and correlating results to design goals. Furthermore, the visualiser will allow the XML output from the simulator to be viewed in an accessible and useful manner, enabling more knowledge to be extracted from the data generated. This is vital as the current presentation of data in large XML databases is not conducive to effective debugging or collation of information. The need for intelligent visualisation of sensor data will be exacerbated by sensors' increasing prevalence in our daily lives.

> *Sensors will result in an explosive increase in data flows as networks become more ubiquitous… new data mining tools will need to be developed in order to effectively extract value from the data… new approaches for the visualization and representation of information will be necessary so that people can clearly understand and communicate the data's value* [3]

Other features of this application will be the ability for developers to replay previous experiments at a later date, allowing for general comparisons, as well as more forensic analysis of events. Moreover, this visualisation allows for spatiotemporal relationships within the simulator to be more easily discerned, and opens up the possibility for data to be quantified in graphs and charts. Currently the XML log exported from the KDEG Ubiquitous Computing Simulator (see Section 2.1.1) is an underused resource, and this paper highlights how SVG visualisation is a practical way of deriving extra benefits from it.

Finally, this project will show how it is possible to implement different SUTs (Systems Under Test) into the visualiser, so that external systems can have their readings integrated with those of the simulator itself. The evaluation of these systems in a virtual environment will be central to their successful implementation in the real world. Hence, this dissertation will highlight the benefits visualisation can bring to the development of such services, by juxtaposing simulator data with the interactions that the SUT is having with the overall framework.

1.2 Objectives:

In general, the objective of this project is to examine information visualisation in the context of ubiquitous computing simulators, and to examine its usefulness as an aid to developing and evaluating domain specific experiments. Specifically the aims for this project are:

1. Design and develop software that can take XML output from the KDEG Ubiquitous Computing Simulator and a System Under Test, and visualise this data in an elegant and useful manner.

2. Access the level of benefit that the post hoc analysis of 2D visualisations can bring to designers and developers of simulator experiments.

3. Investigate whether SVG is an appropriate technology in which to develop such a visualisation tool in.

4. Elaborate on the potential that a more sophisticated version of this software would have in enhancing the integration of various Systems Under Test into the ubiquitous computing simulator.

1.3 Document Structure:

This chapter is followed by a survey of the State of the Art in the areas of ubiquitous computing simulators and SVG data visualisation. Chapter 3 examines the design of this visualisation tool, highlighting choices made regarding the technical implementation of the software, and the visual representation of data. A complete examination of the project’s implementation and problems encountered follows this chapter, which is in turn followed by an evaluation of the project and discussion on its merits and failures. The paper ends in Chapter 6 with some concluding comments and a look at potential future work that could extend the development of this software.

Chapter 2: State of the Art

This chapter presents work currently being undertaken in the two main topics discussed in this paper, namely ubiquitous computing simulators and SVG information visualisation. An analysis of these projects concludes this chapter.

2.1 Ubiquitous Computing Simulators

Because of the expense and practicalities associated with the design and testing of ubiquitous computing applications, there have been recent developments in simulators that allow applications to be evaluated in a more systematic and economic fashion. This section presents two such simulators who exploit 3D technologies to provide a viable alternative to real world prototyping of smart spaces.

2.1.1 KDEG Ubiquitous Computing / Context Awareness Simulator

The success of a pervasive computing application largely relies on its ability to dynamically adapt to the users' context in an appropriate manner. This requires extensive user-centred design and testing to ensure that the levels of adaptive behaviour within the application are suitable for the services it's providing. As a result, the Knowledge and Data Engineering Group (KDEG) in Trinity College Dublin have developed such a platform for the user-centred evaluation of context-aware services [4]. This platform has been implemented as there is a need for the development of systems that allow for repeatable experiments in ubiquitous computing technology, which can then be evaluated in a systematic fashion. It aims to speed up the development process, allow rapid prototyping of systems, and reduce expenses associated with the evaluation of smart space applications.

The platform contains a 3D ubiquitous computing simulator, which is sufficiently realistic to convey changes to the physical and social context of avatars. In addition to this, simulated sensors can be configured to reside inside the virtual environment, with their readings providing the basis for an electronically sensed view of this world. The simulator also provides mechanisms for researchers to integrate their prototype services (System Under Test or SUT) into it, providing a test bed for the evaluation of these context aware applications.

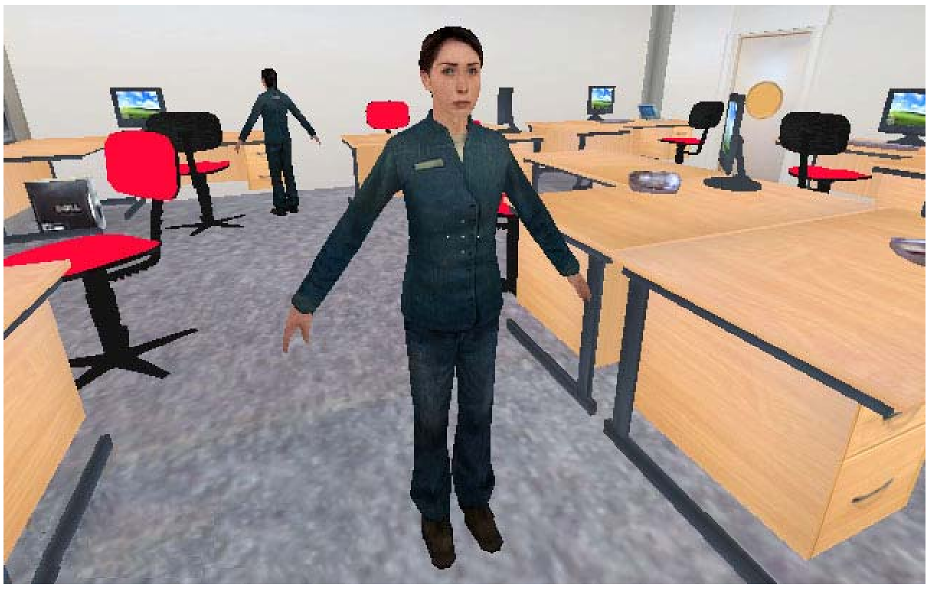


Figure 1: Avatar inside the KDEG Ubiquitous Computing Simulator

The simulator uses a modified Half-Life 2 game engine [5] to display a virtual environment (complete with models of ubiquitous computing sensors), and facilitates the exporting of relevant data in the form of XML encoded messages. It is this XML log that will provide the raw material for the visualisations implemented in this project. The power and flexibility of simulators in comparison with real world implementations, is highlighted by the ease in which a large number of heterogeneous sensors (pressure mats, wireless access points, Bluetooth beacons) can be situated throughout the environment efficiently and easily. These sensor configurations can then be reused or adapted for different environments.

Researchers use a Java application or Proxy to interface their SUT to the simulator, and can access multiple environments simultaneously. Likewise, multiple SUTs may access a single environment. When an experiment is started, messages flow from the virtual environment to the SUT at runtime, with the data leaving the simulator providing the context information in which the SUT can base its decisions. Any responses from the SUT that require changes to the 3D environment (i.e. switching lights off if no user has been detected in a room for more than two minutes), is sent back asynchronously, and actuated in the simulator.

One such SUT that has been implemented on this platform is a Location-aware Instant Messaging service [6] that displays the location of other users if permitted by the accompanying policy-based access control mechanism. The virtual environment allowed the developer to test and debug the system from his own desk, without the logistical problems or costs associated with a real world implementation. Furthermore, running multi-user experiments can be done regularly and at short notice, because the platform facilitates the manipulation of avatars' location in an easy an intuitive way for new learners. In this project, the XML output from the Location-aware Instant Messenger, coupled with the simulator XML, will provide the raw data for a unified visualisation of their interactions.

Problems associated with the KDEG simulator include the tedious and time-consuming nature of reconfiguring connections between the four machines in the framework, and the lack of a dedicated GUI to simplify this procedure. However, with regards to its aim of enabling user-centred evaluation of context-aware services, it has had some initial successes, and these can be built on to provide a more flexible and robust framework.

2.1.2 3DSim: Rapid Prototyping Ambient Intelligence

In October 2005, Ali A. Nazari Shirehjini and Felix Klar presented 3DSim [7] to the Joint sOc-EUSAI conference in Grenoble. This tool which aids the rapid prototyping of Ambient Intelligence applications has a 3D simulator as a core feature. As in the KDEG simulator, the main motivation for this work is to allow the verification of pervasive applications in a virtual world, before deploying them in a physical environment. This encourages rapid development cycles, and saves both on cost and logistics. Moreover, it makes it feasible to test the same interaction solutions in several domains (office, home, etc), not just limiting it to one focused experiment.

One major difference is that 3DSim allows both virtual sensors to be used, as well as real-world sensors, which are connected via standardized interfaces. Thus any UPnP (Universal Plug and Play) [8] control point may invoke actions on a virtual UPnP device in the simulator. These devices are dynamically inserted into the environment and can be removed at run-time. Because XML is core part of UPnP, enabling device and service descriptions, control messages, and eventing, all interaction can be stored in an XML log for later analysis. In order to detect changes in sensors and act upon them, 3DSim has an integrated *Environment Monitoring System* (EMS) which analyses the data, and if appropriate visualises the changed state in the virtual world (i.e. its location, lighting, etc).

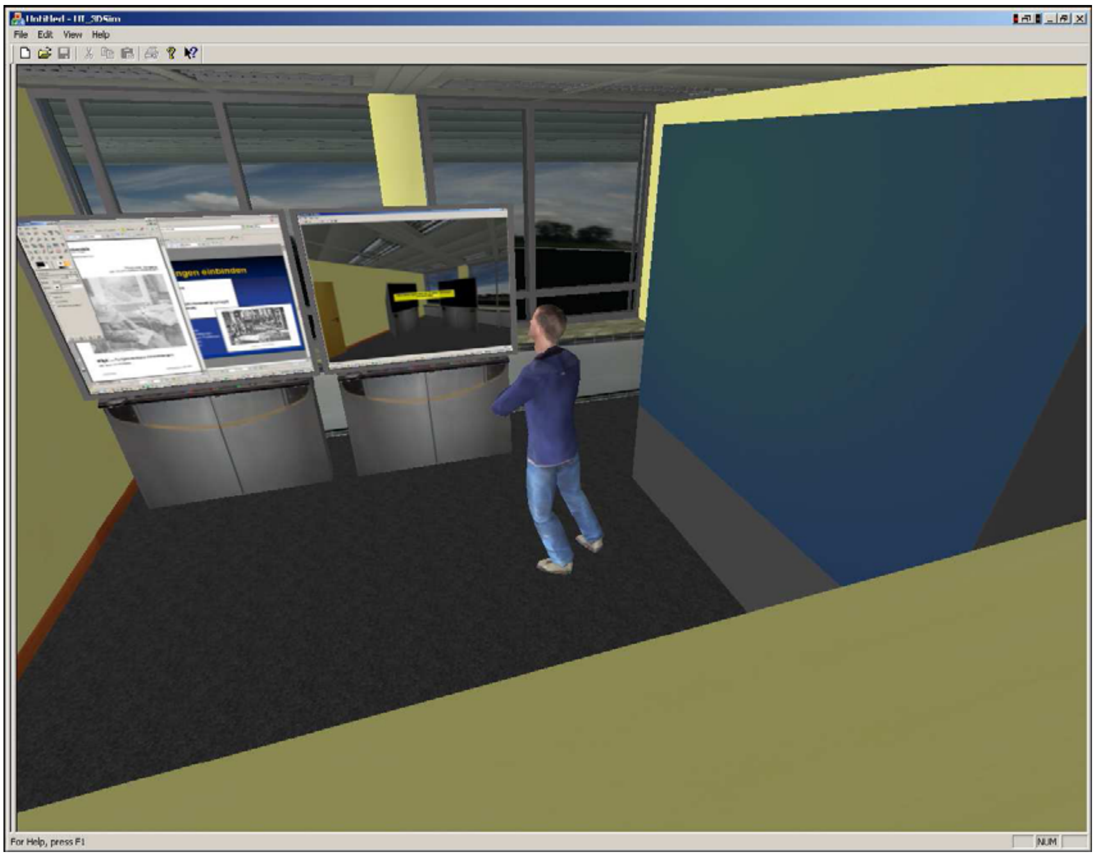


Figure 2: 3DSim display with an avatar looking at a SMART Board.

3DSim employs what it calls *Context Visualisation* to represent and animate events received from real sensor components. This can be used to test accuracy in context aware systems, such as sensors correctly recognising user activity, and objects reporting their state accurately. Thus, a designer can monitor the *awareness* of an environment he is setting up, and reconfigure accordingly until the space is reacting appropriately to the sensor input. If the EMS identifies a user activity from the sensors, an animation of an avatar doing that task (cleaning a whiteboard, presenting in the boardroom, etc) is generated in the simulator. Likewise, *environmental states*, such as light, are visualised in the virtual world by adjusting the 3D scene settings, and *device state*s, such as a door opening, are rendered in the scene with appropriate changes to light and shadow. Another important feature of 3DSim is the ability for multimedia artefacts to be displayed on screens within the virtual world.

The core of 3DSim consists of a CVE-server (Collaborative Virtual Environment) which manages the simulated world, and a 3DSim-client which visualises and allows interaction with the 3D world via a GUI. The CVE is based on HOUCOM [9] which manages object lifecycles and provides event distribution. The platform is also extensible, allowing modules containing new 3D objects to be added to, or removed from, the environment dynamically at runtime.

Overall, 3DSim has some unique features which make it a valuable application for developers of Ambient Intelligence to use. For instance, it facilitates the development of PDA based control systems, adaptive user interfaces and goal-based interaction systems. Furthermore, it renders scenes to give a photo-realistic impression, and if components designed using it are developed with UPnP, it means that they can be easily adapted to real world environments. These factors combine to make 3DSim a flexible and innovative tool.

2.2 SVG Information Visualisation

SVG is a W3C recommended language for describing two-dimensional graphics and graphical applications in XML. It has been used as a technology for numerous forms of information visualisation and as has proved to be a flexible and efficient open source method for such research. Vector formats such as SVG are small in size, can be animated, and allow the skewing and enlarging of graphics without degradation. Thus, it means that SVG or Adobe Flash's SWF format [10] are the only viable options for this project. A combination of a Java widget scheme (SWT [11], SWING [12], etc) with a raster graphic format like PNG, JPEG or GIF, would not allow the graphical flexibility required for this project.

Though SWFs are a powerful vector format who's native scripting language, ActionScript [13], has good XML support, the fact that it is a proprietary technology militates against it. Furthermore, because SVG *is* an XML language, thus allowing easy integration with other XML technologies such as XSL [14], meant that SVG was chosen as the vector format of choice for this project. In this section I will describe how this technology has been exploited to produce some innovative and useful spatiotemporal visualisations.

2.2.1 Dynamic Processes of the Gruben Glacier

In 2003 Yvonne Isakowski produced a visualisation of dynamic glacier processes using SVG animation [15]. Because geometric variations, increasing and decreasing mass, and ice flows, are significant properties of glaciers, it was an ideal subject in which to demonstrate the advantages that dynamic SVG animation could offer cartographic visualisations. Previously cartographers had relied heavily on static media to visualise glaciers, which meant they could only offer a snapshot of the current status of these highly variable, non-static entities. In her dissertation, Isakowski used the *Gruben* glacier in the Valais region of Switzerland as her test location, as near homogeneous measurements from the area has been gathered annually since 1970. This provided the ideal data to demonstrate the potential SVG animation has in enhancing the visualisation and understanding of such dynamic activity. Furthermore, this work would enable the discovery of difficulties and problems associated with the technology, as well as its many positive attributes.

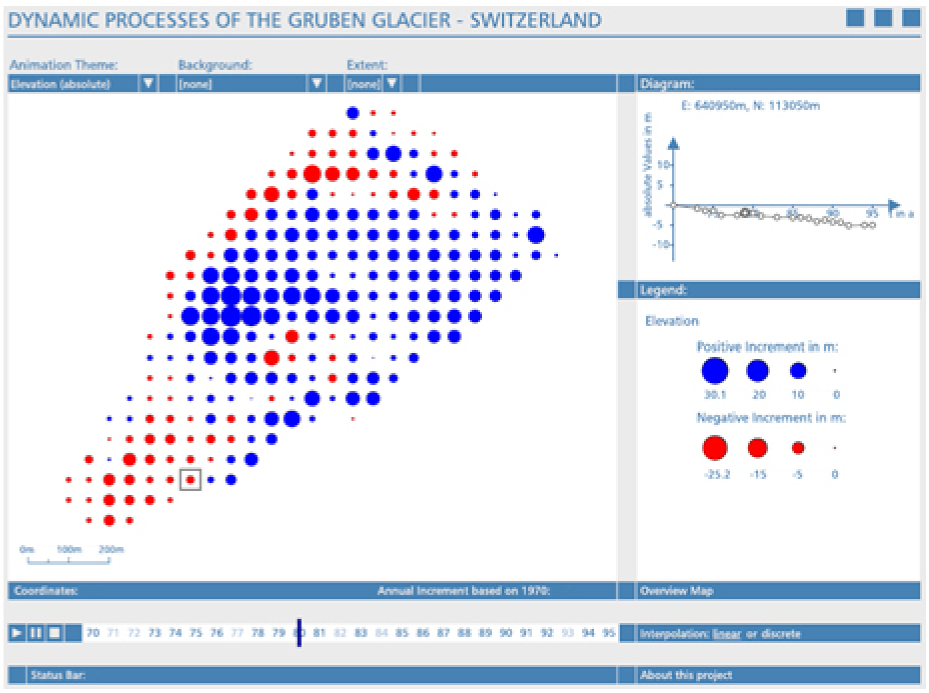


Figure 3: Screenshot of Dynamic Processes of Gruben Glacier

The initial stage in the development of the glacier visualisation system involved the conversion of the base data from raw text files to an SVG format, which was done using a PERL [16] script. Criteria for the representation of the glacier and its interface were then drawn up (scale, legend, navigation, font, and colour), and several SVG documents pertaining to each individual animation were created with these criteria in mind. As a result, four separate animations were integrated into the final piece, highlighting the absolute elevation change, relative elevation change, surface velocity, and extent of the glacier, over a twenty five year period.

A combination of SMIL (Synchronised Multimedia Integration Language) [17] and DOM (Document Object Model) [18] functions were implemented to control the animation of the SVG. This was necessitated by a lack of flexible timing controls in SMIL alone. By encoding information in SVG attributes (radius and colour), it was possible to highlight the extent of the increase or decrease in the glacier in an intuitive and effective manner. In addition, each individual point has a graph of values associated with it that can be accessed, and because these graphs are synchronised to the main animation, it allows them to be played in parallel. Drop down menus are used to switch between different animation themes, but the lack of GUI elements native to SVG were noted and showed some of its shortcomings as a tool for interface design.

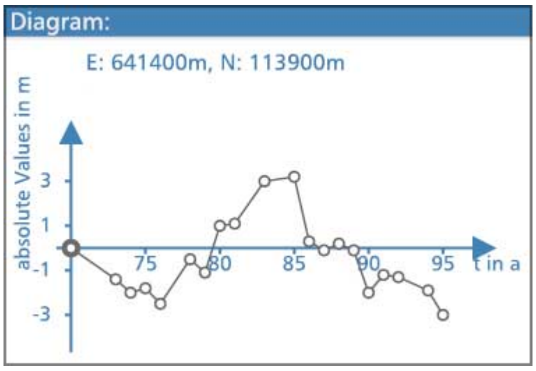


Figure 4: Graph (absolute) for a certain point in the elevation animation

Another attribute of SVG highlighted in the production of this application was that animation parameters were stored as text, and thus didn't bloat file sizes unnecessarily. However, it was noticed that when animating a large number of objects at once smooth motion was impaired, even on powerful machines, as animations are calculated and rendered in real time. Furthermore, as interpolation and rendering of animation objects were done *just in time*, allowing users to interact with graphics dynamically, there were some inconsistencies in the drawing of some complex data, especially when pausing the animation.

Notwithstanding these issues, the spatiotemporal visualisation of the Gruben glacier is one of the best examples of dynamic data visualisation to be implemented in SVG. It combined four animation topics, and integrated them with an interactive timeline and intuitive user interface. The combination of allowing users to compare different perspectives of the one event, with the ability to control the presentation of the data, has resulted in a powerful application, albeit for a niche domain.

2.2.2 GEMOS, A Building Management and Security system's SVG Interface

In 2005 Christopher B. Peto presented GEMOS [19] to the SVG Open conference in Enschelde the Netherlands. GEMOS exploits vector graphics created by architects in CAD [20], to develop an SVG interface for a building management and security system. The motivation for this was to use existing entities as the basis for a new application, practical for everyday use.

Because all building plans today are created in vector format (such as AutoCAD, DWG or DXG) they are easy to convert into SVG, which maintains the ability to zoom into them without degrading the image quality, whilst also adding a significant dynamic component to the plans. This adaptivity is the key that allows GEMOS to monitor and control building systems in real time, via a web interface. For instance, by interfacing devices such as fire sensors and intrusion detectors with the server, it allows for their readings to be dynamically displayed over the static floor plan. Thus, if a fire sensor is triggered in the building, an automatic script is called to highlight the area in question, and to print out an updated sensor map. These can be used to better inform administrators, and the fire department, as to the best plan of action.

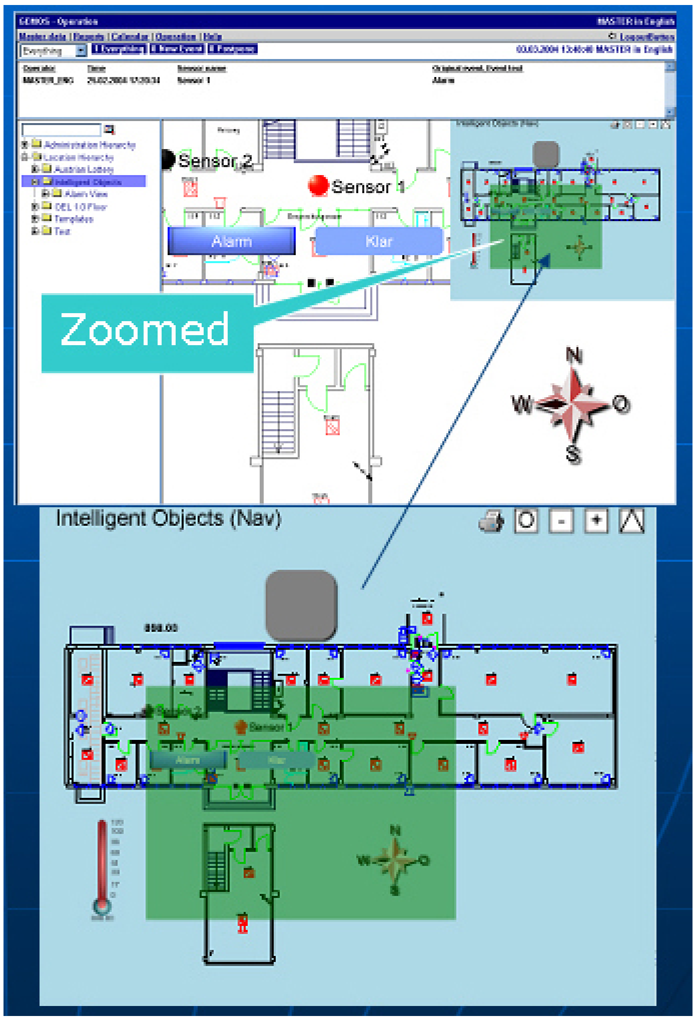


Figure 5: GEMOS, Real-time floor plan display

The entire process starts with the importing and converting of CAD data into a SVG floor plan and database. It is essential at this stage that static graphics (walls, doors, etc) are separated from the dynamic objects such as sensors. The CAD import assistant, which is used at this point, ensures that any future changes to the building plans can be updated in the SVG interface quickly and efficiently. When the base floor plan has been added to the system, layers of sensors, organized by type, can be categorized and superimposed on the map using the web interface. Once this has been completed satisfactorily settings are saved to the GEMOS web server.

In order to edit floor plans and configure intelligent objects in the system, the SVG editor is employed to interact with the GEMOS server. It consists of a HTML frameset which loads objects from the database into session parameters when instantiated. Once initialised the objects and map are loaded into the editor's SVG interface. Each type of object (Sensor, Circle, Line, etc) has an ECMAScript [15] class associated with it, containing all its attributes, properties and DOM references. All objects are then sorted into arrays, as this optimizes speed by limiting access to the DOM, except when a change of position or style has been performed on an object. New objects can be inserted at this stage as well, or else drawn directly using native SVG tools, such as Text or Cubic Curve. Modifications follow a similar process, and updates of attributes are shown directly. All changes can then be saved to the database before the final visualisation occurs.

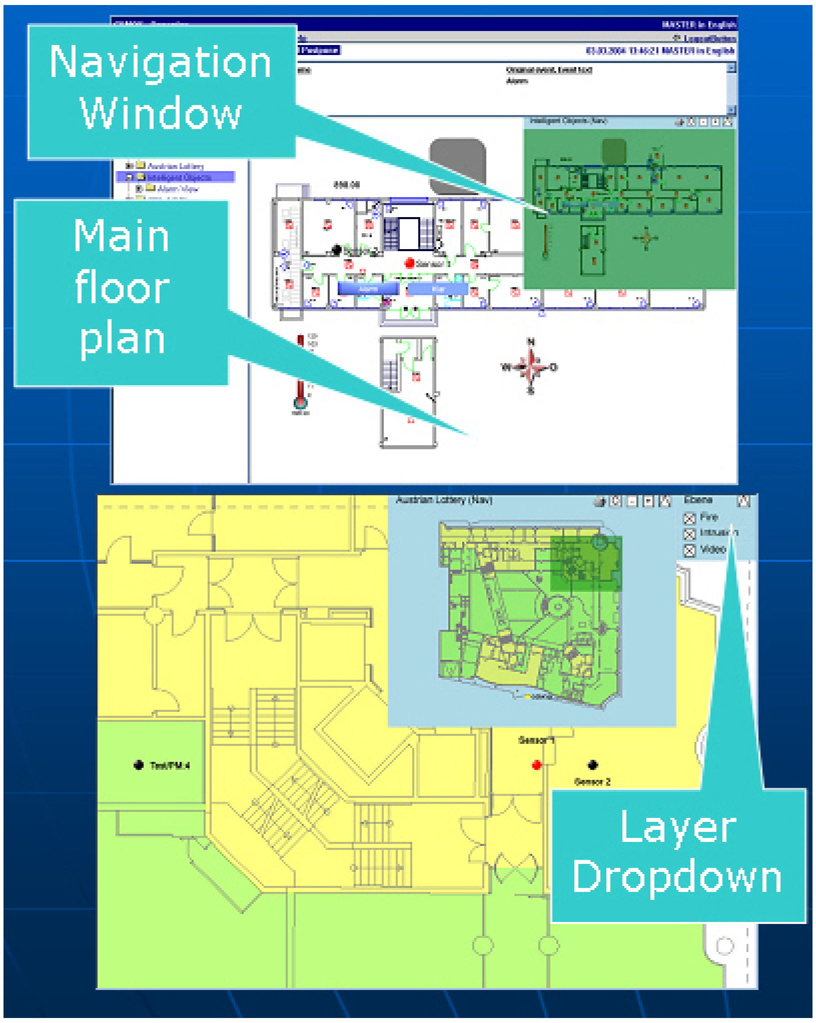


Figure 6: GEMOS, Real-time floor plan display

GEMOS enables real-time status information to be viewed and navigated through its SVG interface, which is displayed using the Adobe SVG viewer [21] and Microsoft Internet Explorer. By combining ECMAScript [22] with SVG, zooming and panning functionality has been added to the display, as has layer showing/hiding, which is facilitated through DOM manipulation. If an object's status changes, i.e. an intrusion has been detected, a predefined refresh script is pushed to the browser, which locates and updates the relevant attributes of the object. Overall this system represents one of the most impressive implementations of SVG to date, with its integration with databases and ECMAScript highlighting the powerful potential that SVG applications have.

2.3 Analysis

Though it has been shown that ubiquitous computing simulators enable the rapid development of context-aware experiments, there is a noticeable lack of an overview visualisation tool included in them. Such an application would enable researchers, at a glance, to understand what is happening in the experiment at a particular moment in time, or to save visual representations of a test for posterity. Furthermore, the visualisation of data would aid the debugging process, give a greater insight into spatiotemporal interactions, and could represent sensor events in more accessible fashion than plain text, which allows for trends to be determined. SUT integration, or the testing of new interaction models, would also benefit from their juxtaposition with simulator events.

As has been shown, SVG visualisation offers a viable way of presenting such data to users of ubiquitous computing simulators. Indeed the goals of the *Gruben Glacier* experiment (visualisation of a dynamic data set, and the investigation of the suitability of SVG for such a purpose) are similar to aims of this dissertation, though in a vastly different domain. However, the *Gruben Glacier* visualisation was designed as a once-off representation of a specific data set, rather than a generic tool for the visualisation of glaciers. This is in contrast to the ubiquitous computing visualiser, which will allow any XML output from the test experiment to be visualised onscreen.

How extensible the visualisation system should be (allowing configuration of new maps and SUTS) will be explored in this project, as extensibility would make the software an extremely flexible tool for developers of experiments to use. Such extensibility has been implemented in the GEMOS system, in relation to adding new maps, but it currently has no facility to integrate SUTs into its framework. Moreover, GEMOS' visualisation of complex data is not as elegantly displayed as it might be, and its lack of availability for this project meant adapting an existing version of its system was rejected as a viable approach.

2.4 Conclusions

Chapter 2 introduced state-of-the-art ubiquitous computing simulators and how they can be used to speed up the design and implementation of pervasive computing systems. By reducing the cost and logistics associated with such developments, rapid prototyping and user-centred evaluation of platforms can take place. However, limitations in the presentation of such heterogeneous data were highlighted, and examples of SVG visualisation were introduced to show how visual representation of dynamic data can be a valuable asset to researchers in ubiquitous computing.

Chapter 3: Design

This chapter introduces the design principles and core technologies used in this project. The visualiser's architecture and the types of XML messages used are also discussed, as are the various components that make up the system. The chapter concludes with an examination of pertinent security issues relating to the visualiser.

3.1 Technologies Used

This section briefly describes the technologies used in this project and how this combination has resulted in a flexible and versatile framework. For instance, the employment of XSL allows users to customise what messages to filter out of the XML log, which is important for extensibility. Likewise, if any PHP functions need to be called (such as character replacement in text), the mechanisms are already in place.

The use of Internet Explorer and Adobe SVG Viewer creates a cohesive platform for SVG, JavaScript, HTML and DOM to interact, which is core to the operation of this software. Furthermore, if new SVG or JavaScript needs to be created (for floor plan, SUT or interface), it only requires this code to append or replace the current content, and then for it to be loaded into the viewer. Finally, all the technologies used are freely available, and the ample documentation and support available for each will ease system implementation. A short description of each follows:

Adobe SVG Viewer is a plug-in for web browsers that allows users to view and interact with SVG content.

Document Object Model (DOM) allows XML files to be represented in a tree form. The entire content of the document must be parsed and stored in memory, allowing elements and nodes to be accessed and manipulated via scripting languages like

JavaScript. This project uses the ActiveX DOM Object preinstalled with Microsoft Internet Explorer.

JavaScript is an extension of the ECMA-262 standard. It is a scripting programming language that runs at the client-side, is interpreted, and is loosely-typed. JavaScript functions can be embedded or included in HTML pages, and they allow interaction with DOM objects.

HyperText Markup Language (**HTML**) is a markup language that allows the creation of web pages that can embed objects such as SVG. It needs to be displayed in a web browser such as Microsoft Internet Explorer or Mozilla Firefox.

Scalable Vector Graphics (**SVG**) is an open standard created by the W3C that allows static and animated vector graphics to be represented in XML.

Extensible Markup Language (**XML**) is another open standard recommended by the W3C, which allows the creation of user-defined markup languages capable of describing any data. Its main purpose is to enable interoperability on the Internet, and the sharing of data across different systems.

Extensible Stylesheet Language (**XSL**) is a family of languages which allows the transformation of XML documents. A new file is created based on the content of the original file, and this allows the conversion or filtering of XML data.

PHP is an open source programming language that enables dynamic web content and server-side programs. It has good support for a large number of databases including MySQL, Oracle, Microsoft SQL Server and PostgreSQL.

Apache is a popular open source HTTP web server that runs on many platforms. There is excellent support for applications using Apache and PHP.

Sablotron is an open source XML toolkit that provides XSL extensions for the PHP programming language.

3.2 Data Flow in Generic Visualisation System

A variety of different technologies could have been used to develop this ubiquitous computing visualisation system. Figure 7 below, describes the flow of data through such a framework in generic terms, and highlights how different technologies are needed to facilitate the transport, editing, storing and display of raw data.

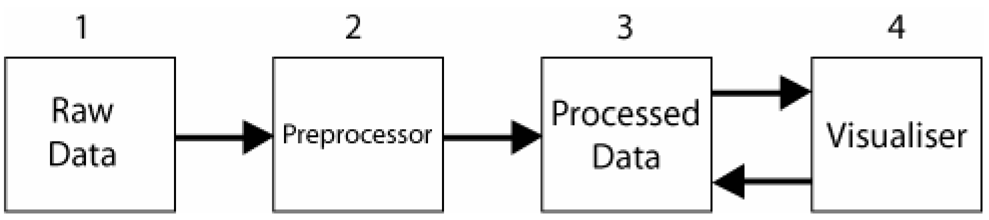


Figure 7: High Level Interaction within a visualiser.

1. Raw data is outputted from the simulator and SUT.

2. This is filtered and altered in the pre-processor until it is in the requisite format.

3. The processed data is stored and made available to the visualiser's template

4. The visualiser interacts with the processed data to update its display

3.3 Architecture of Visualisation System

When an experiment is run on the ubiquitous computing simulator, the positioning of avatars and their interactions with sensors are recorded in an XML log. A SUT that is integrated with the simulator can also have its readings interspersed into this XML log. Figure 8 displays the system architecture, and the processes of the visualisation tool.

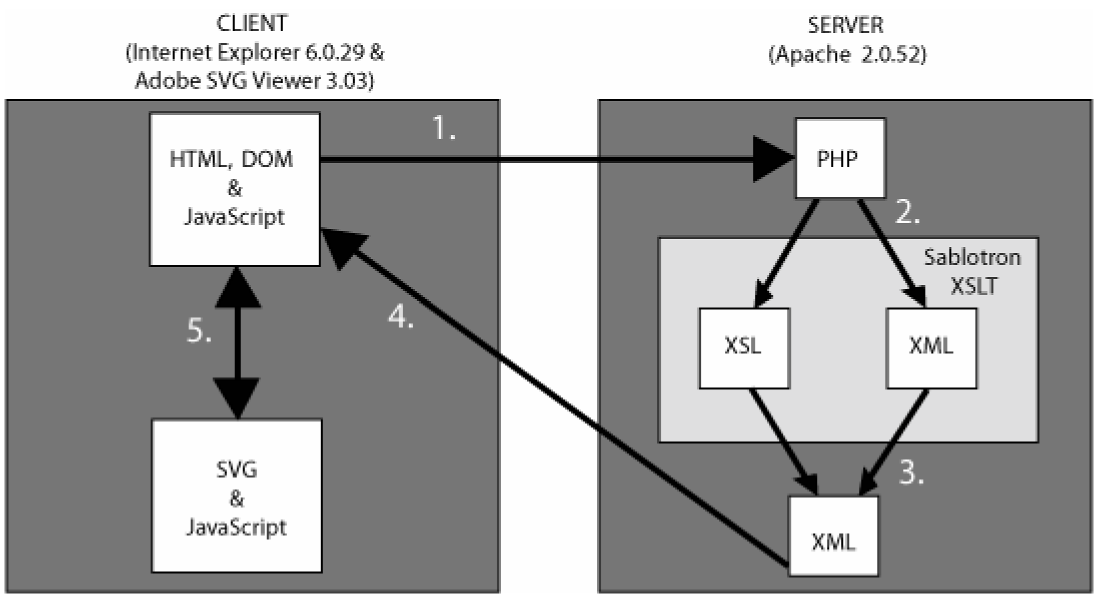


Figure 8: Architecture of Visualisation System

1. When the application's HTML holder file is opened by the Internet Explorer Browser, a PHP script is automatically called on the Apache 2.0.52 HTTP Server.

2. This filters the XML log (using the Sablotron XSL extension for PHP 4.42) according to the XSL file created by the experiment developers.

3. Because of restrictions with special characters, this PHP file also replaces any *'&lt;'* strings with a '<' character and any *'&gt;'* strings with a '>' character. This allows the entire log to be used as XML, because it converts any relevant character strings to XML elements.

4. The processed XML file is returned by this PHP script, and is automatically loaded into an ActiveX DOM object. If there is also SUT information interspersed in the XML log, steps 1-4 are repeated using a second PHP script. Similar XSL filtering and string replacement is performed, before this SUT related XML is loaded into a DOM object.

5. The HTML holder file has the SVG file embedded in it, and this is displayed using the Adobe SVG Viewer 3.0.3. The SVG file has some base elements and local JavaScript functions that it can call, as well as functions it can access from the head of the HTML file. When required, the SVG extracts

information from the DOM objects, and appends this new information to itself.

3.4 System Under Test Architecture

The system under test visualised in this project is based on the Location-aware Instant Messenger described in Section 2.1.1 of this document. It consisted of a number of separate entities (including the Simulator) interacting as described below.

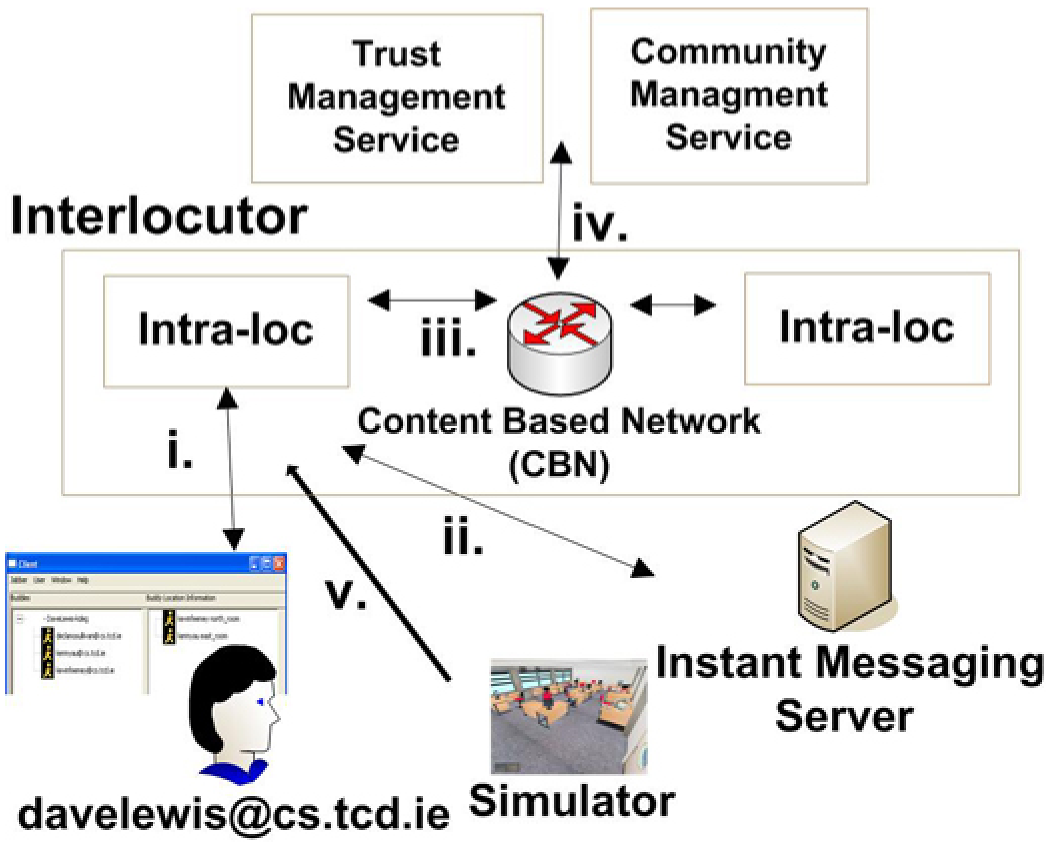


Figure 9: Architecture of chosen System Under Test (SUT)

What this system allows is for different users to simultaneously use the simulator, and find out the locations of other users on their team. Conversely, any users who are not on the team will have their location in the simulator blocked from them. In order for this to operate, a user must first use the Instant Messaging Server to select another person they would like to locate. This sends an *Access Control Request* to the *Trust Management* and *Community Management Servers*, which makes a decision as to the success of the request. An *Access Control Decision (ACD)* is sent back to the user, and if positive, it results in the requested user's location in the simulator being updated in the instant messenger, each time the user changes room. These messages containing new location information are referred to as *Location Updates*.

In choosing what elements of the SUT to use, *Access Control Decisions* and *Location Updates* were prioritised for visualisation, as it was felt that the most important thing from a developers' perspective was to know what access a person had to the locations of other users at any moment in time, and when did they receive updates of their location. Other messages that are used in this system include *Jabber Messages*, *Elvin Messages*, *Community Control Requests* and *Community Control Decisions*. It would be reasonable to represent any of these messages that propagate through the system, depending on what the researcher was using the visualiser for and what they wanted to better understand. For instance, it would be possible to visualise the time latencies it took for a single request to traverse each part of the system, and to return a reply to the sender. However, due to time constraints and the fact that it would not have added significantly to the concept under test, it was decided to concentrate purely on the *Access Control Decision* and *Location Update* elements of the SUT. Once the main principals of the concept have been proved, all future implementations can use a similar visualisation mechanism.

3.5 Types of XML Messages

As has been mentioned, the 3D ubiquitous computing simulator outputs a number of different messages, representing different actions that occur within it. These include the location of users and the events they trigger, such as entering rooms. These two messages are what were utilised in the implementation of the system, but it would have been possible to include a variety of other messages had they been available, such as wireless sensor propagation levels, motion detectors and heat sensors. As well as the Simulator itself producing XML messages, the SUT also generates its own, depending on its configuration. For the purposes of this application, the SUT messages that were visualised were *ACDs* and *Location Updates*.

3.5.1 XandY Messages

XandY messages are sent out periodically, and contain either location information for users inside the simulator, or event information for actions occurring within the 3D environment.

3.5.2 Location Message

The message below is identifying the X, Y and Z coordinates for user4@jabber at a specific point in time of an experiment.

```
<Term typeID="XandY">
     <ReceivedFrom>SIMULATOR</ReceivedFrom>
     <SendingTo>ELVIN</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:13:9:328</Time>
     <TimeUTC>1146147189000</TimeUTC>
     <UserName/>
     <Direction>IN</Direction>
     <Message>
          <msg>
               <users>
                    <user>
                         <id>user4@jabber</id>
                         <location>
                              <x>449.717224</x>
                              <y>-400.385742</y>
                              <z>-296.968750</z>
                         </location>
                    </user>
               </users>
               <event>
                    <none/>
               </event>
               <time>693.299988</time>
          </msg>
     </Message>
</Term>
```

3.5.3 Event Message

The following message is instantiated when *user2@jabber* enters the *Atrium* at a specific time.

```
<Term typeID="XandY">
     <ReceivedFrom>SIMULATOR</ReceivedFrom>
     <SendingTo>ELVIN</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:13:28:734</Time>
     <TimeUTC>1146147208000</TimeUTC>
     <UserName/>
     <Direction>IN</Direction>
     <Message>
          <msg>
               <room>Atrium</room>
               <location>
                    <x>485.791443</x>
                    <y>773.135742</y>
                    <z>-296.968750</z>
               </location>
               <user>user2@jabber</user>
               <event>
                    <enter/>
               </event>
               <time>711.195007</time>
          </msg>
     </Message>
</Term>
```

3.5.4 Access Control Decision Message:

This message is the reply generated to an Access Control Request that has been made by a user. In this example *user4@jabber* has had its request to be made aware of the location changes of *user1@jabber* declined.

```
<Term typeID="1146147133000-85">
     <ReceivedFrom>ELVIN</ReceivedFrom>
     <SendingTo>CLIENT</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:12:13:484</Time>
     <TimeUTC>1146147133000</TimeUTC>
     <UserName />
     <Direction>IN</Direction>
     <Message>
          <ELVIN-NOTIFICATION>
               <ACCESS-CONTROL-DECISION>ID</ACCESS-CONTROL-DECISION>
               <ACD-PAYLOAD>DECLINE</ACD-PAYLOAD>
               <ACD-TARGET>user1@jabber</ACD-TARGET>
               <ACD-SERVICE>LOCATION</ACD-SERVICE>
               <ACD-SUBJECT>user4@jabber</ACD-SUBJECT>
          </ELVIN-NOTIFICATION>
     </Message>
</Term>
```

If the Access Control Decision had been successful for *user4@jabber* it would have meant that the <ACD-PAYLOAD> element would have looked like this instead:

```
          <ACD-PAYLOAD>*</ACD-PAYLOAD>
```

3.5.5 Location Update Message

This message is sent to users updating them on the location of other users in the simulator.

```
<Term typeID="1146147213000-120">
     <ReceivedFrom>ELVIN/SOAP</ReceivedFrom>
     <SendingTo>CLIENT</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:13:33:296</Time>
     <TimeUTC>1146147213000</TimeUTC>
     <UserName>user1@jabber</UserName>
     <Direction>OUT</Direction>
     <Message>
          <message-with-applied-access-control>
               <LOCATION-SERVICE>LOCATION</LOCATION-SERVICE>
               <PAYLOAD>Lobby_Nth</PAYLOAD>
               <SUBJECT>user3@jabber</SUBJECT>
          </message-with-applied-access-control>
     </Message>
</Term>
```

If the Location Update message had been unsuccessful for *user1@jabber* (i.e. it hasn't received a successful Access Control Decision) the <PAYLOAD> element would have looked like this instead:

```
          <PAYLOAD>DECLINE</PAYLOAD>
```

3.6 Design Principles

When planning the design for this visualisation tool it was essential that all elements of the interface (both visual and functional), were conducive to users getting increased knowledge from the information in an easy and intuitive fashion. The combination of good interface design with information visualisation enables this.

> *For information to become knowledge, we need to interpret and understand it. Visualization in general responds directly to this need* [23]

This section contains the design principles that were followed in the creation of the interface and controllers, and the visualisation of the floor plan and SUT.

3.6.1 Interface and Controllers

- Best effort should be made to keep the interface design as clean as possible.
- There should be a clear separation between the SUT and the simulator components.
- All references to an avatar (iconic or textual) must maintain the same colour scheme throughout the interface.
- Timelines should clearly show the temporal context an event occurred at.
- Standard icons for controlling animations should be used throughout.
- Visual representations should be backed up by a textual explanation, if it clarifies the intended meaning.

3.6.2 Floor Plan

- The floor plan should only contain objects that are absolutely vital to that particular experiment.

- Any map data superfluous to the experiment should be deleted or have its opacity set to zero.

- The event timeline should contain all simulator events, even if they are many different types

- Labels on the floor plan should be numerical and a have a corresponding legend.

- The main static elements of the map (floor and walls) should only be drawn in black, grey, or white, so that superimposed sensors or avatars can be clearly contrasted against the background.

3.6.3 System Under Test

- SUT visualisations should use the simplest shapes as possible to convey their meaning, as long as this does not hinder cognition.

- If SUT visualisation comprises of more than one component, it should be obvious to users which region is active

- The SUT timeline should contain all SUT events, even if they are many different types

- The choice of colours in the SUT should complement the rest of the interface, reinforcing the knowledge already conveyed.

3.7 Interface Design

It was important that the interface for this application allows researchers to get an instant overview of what is happening in the experiment, as well as facilitating a more detailed analysis of events occurring. Thus, the optimal visualisation of the floor plan, its controls, and their positioning in the interface is central to the success of this application. Furthermore, it is essential that there is a clear separation between the information relating to the simulator and that relating to the SUT, though allowing the two sections to work in tandem is paramount. Several interface designs were tested on paper before an implementation on computer, with Figure 10 showing the final layout of the interface and its various components (See Appendix C, for larger version). The design of each part of the interface is discussed in the next sections.

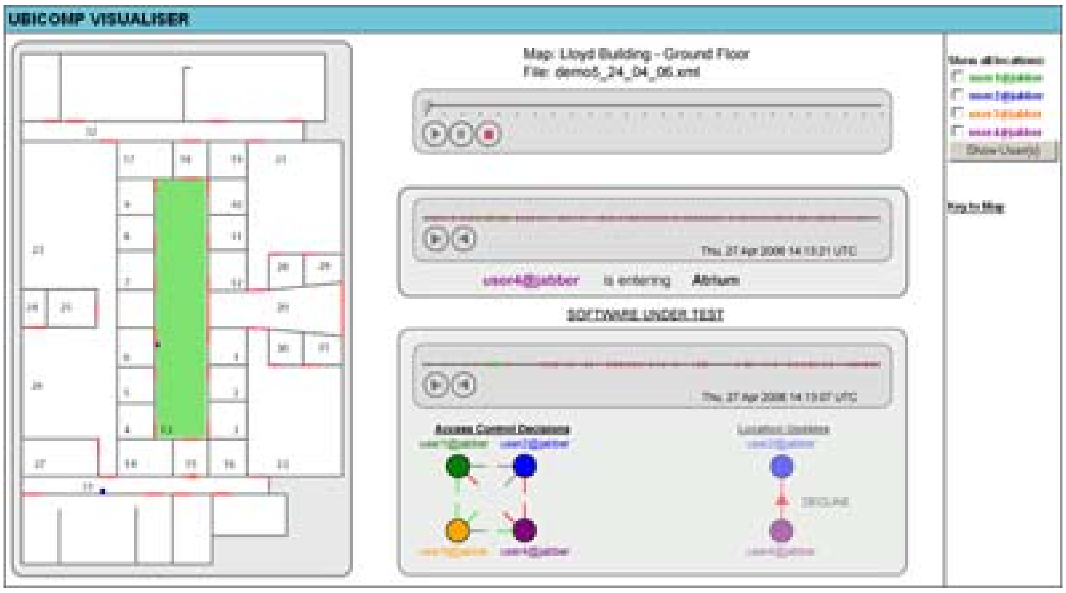


Figure 10: Visualiser Interface

3.7.1 Visualising the Floor Plan

In order to maintain perspective and maximise clarity for users, an overhead projection was used to visualise the map of the ground floor. Information essential to the experiment was maintained in the map, which meant that only walls and doors were visualised in this minimalist design. Superimposed on top of each room was a number that could be cross-referenced with a list of room names, meaning the extra clutter of text did not have to be visualised. If users triggered a room sensor while walking in the simulated building, the corresponding room would change colour to green to signify that they had entered.

The grey and white colour scheme is deliberately understated to allow the colours of the doors, avatars and sensors stand out more during animation. Consistency in colour is also catered for in the circular icons representing avatars in the simulator. Thus, the colour of each user is maintained in all visual and textual references to it in the interface. Though the Adobe SVG Viewer has inbuilt zooming and panning controls, they are not going to be essential to the use this visualisation, as the floor plan and controllers are compact enough to appear on a single screen, yet large enough to display information with clarity.

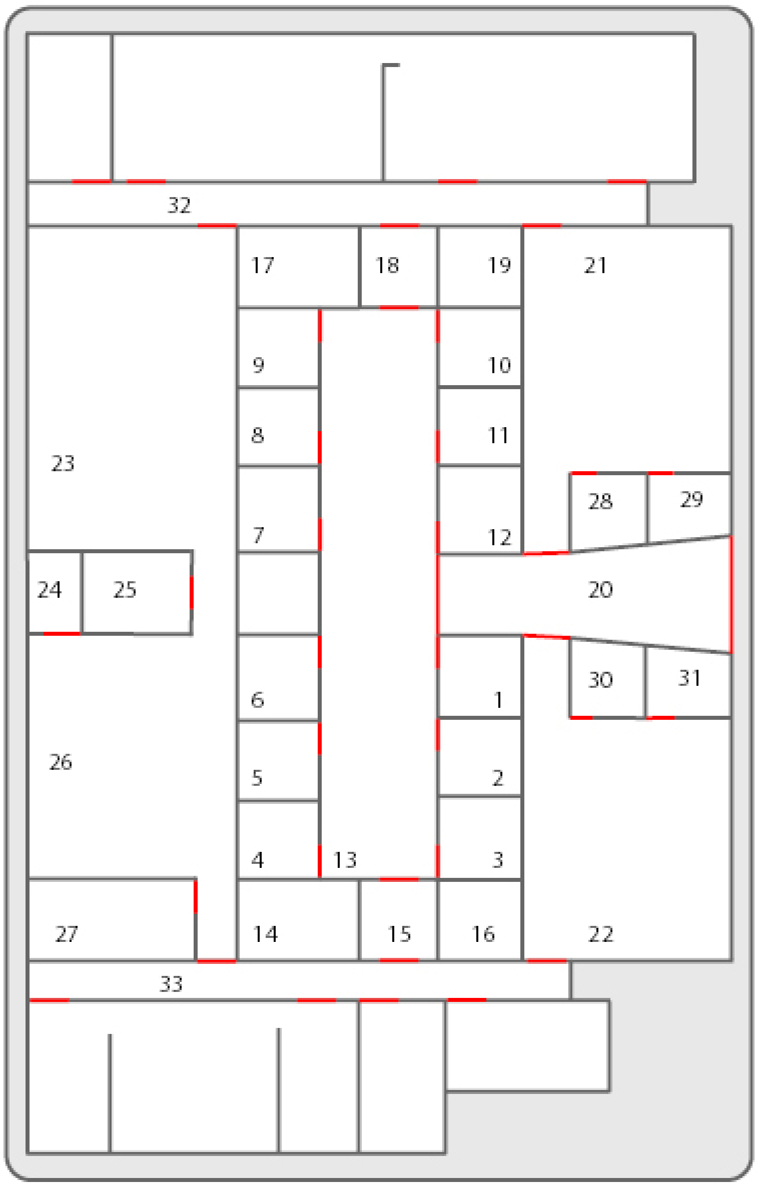


Figure 11: Visualisation of Floor Plan

3.7.2 Visualising the Animation Controls

In order to give users more flexibility as to how they viewed interactions within the simulator, two separate animation controllers were implemented in the interface. Figure 12 shows the control a user would use if he wanted to play the floor plan animation. It uses standard icons for the *play*, *pause* and *stop* functions, which means that new users will be instantly familiar with the visual metaphor on display. A *thumb* moves along the timeline when the animation is in motion, which informs users of the temporal context current actions are occurring. Finally, the current time of the animation is displayed in the right hand side of the controller beneath the timeline.

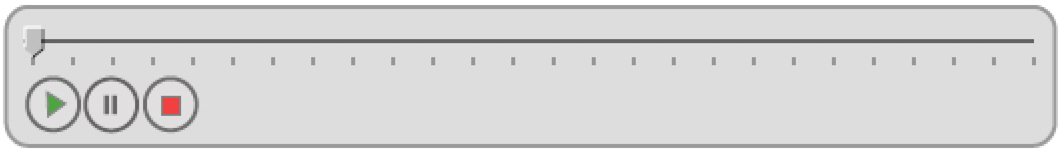


Figure 12: Animation Control for Floor Plan

The second floor plan controller is designed to allow users to step through the animation on an event-by-event basis. This allows a more forensic analysis of the information than the overview animation offered by the other controller. Each event is represented by a red vertical line on the timeline, with the current event turned green and enlarged to make it more visible. Again this allows users to now the temporal context of an event. Users can navigate forward and back along the timeline using the standard buttons, and the corresponding event time is updated on the right hand side of the controller. If an event signifies an avatar entering a room, as opposed to its x and y coordinates, the name of the user (in the appropriate colour) is printed, as is the room being entered.

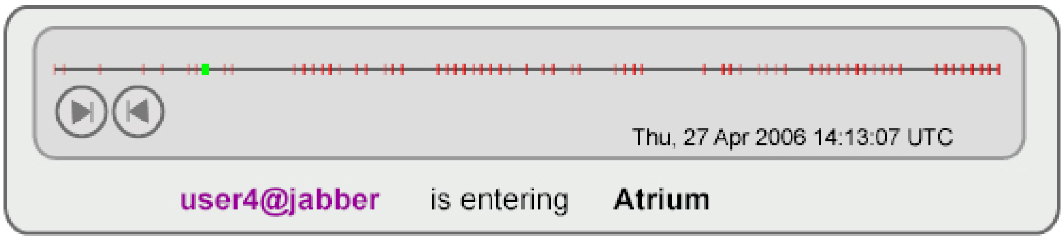


Figure 13: Event-by-event animation controller, after room sensor has been triggered

It was decided not to synchronise the floor plan animations with the SUT visualisations, because events can occur in rapid succession, sometimes with just a few milliseconds between them. This would result in the interface blinking rapidly, proving a distraction to users, and not allowing new knowledge to be conveyed. The best mechanism for allowing close comparison of SUT and simulator events was to allow users to synch the two event timelines manually, as required.

A decision was also made not to interpolate between location readings of avatars in the simulator, which would have been possible using SMIL. This choice was made, because it may have resulted in avatars walking through walls if the frequency of their location messages was not high enough. Moreover, it would have given an unrealistic visualisation of user interactions within the simulator, and wouldn't have visualised correctly the limitations of location sensors.

3.8 SUT Visualisation

The aim of visualising the SUT was to allow developers to see the current state of the system at a glance. As mentioned previously, a decision was made to separate this component from the animation controls of the floor plan for reasons of clarity, though both parts still do complement each other. The next aspect to be finalised was the placing of events relating to both *Access Control Decisions* and *Location Updates* together on the one timeline. Because both event types were intrinsically linked to each other, it was felt that this was an appropriate solution. Furthermore, their connection would be further reinforced by placing their respective visualisation components just beneath the timeline.

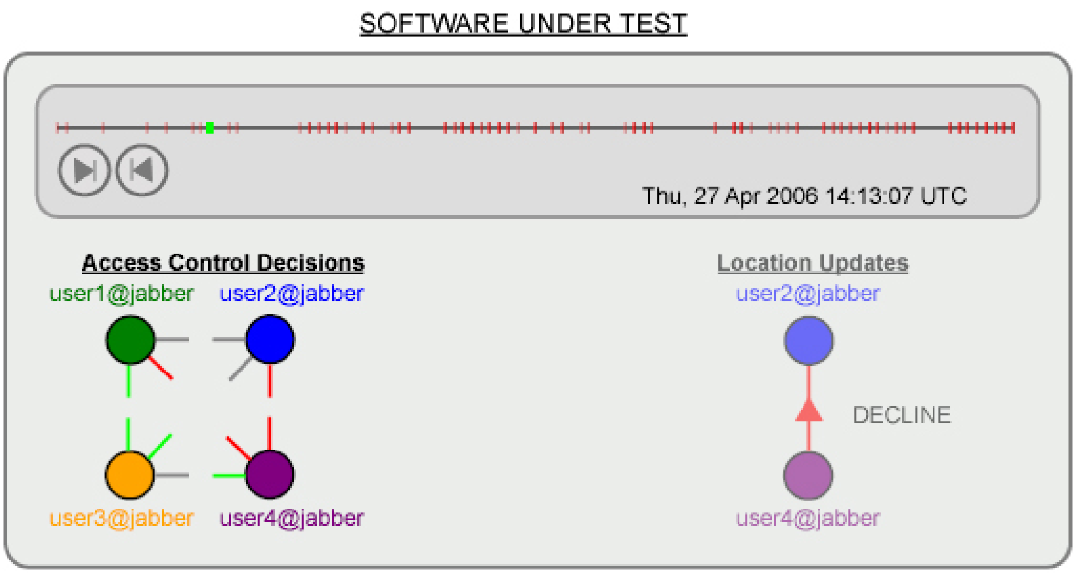


Figure 14: Visualisation of SUT component

Because there are two types of messages being represented on the one timeline (*ACDs* and *Location Updates*) it was important for users to be able to distinguish which event on the timeline corresponded to what SUT visualisation. Hence, when an *ACD* is being visualised the *Location Update* section is greyed out and visa versa. This accentuates the active component, while still allowing the other visualisation component to be seen. This is vital in order for researchers to spot that *Location Updates* are not contradicting what the *ACD* component is displaying.

For continuity with the rest of the application, the look and feel of the timeline was identical to that of the event based controller for the floor plan. The user was able to scroll forward and back through events with the relevant event on the timeline highlighted in green. Again the corresponding time would be displayed just beneath the timeline.

3.8.1 Access Control Decision Visualisation

Numerous design iterations were undertaken for this component before a representation was finalised. It was vital that with minimal learning a prospective user would grasp the visual metaphor employed. Each of the four users is represented by a circle whose colour is consistent with their representation elsewhere in the application. They are accompanied by the text of their name to reinforce in the mind of researchers which colour is representing each user, but it is anticipated that quite quickly they will associate users with the colour, negating them having to read the text.

In essence, each circle (representing a particular user) has three lines protruding from it pointing towards the other three circles. If a line is coloured green it means that that user has had a positive *ACD* received from the user the line is pointing at. If the line is red it means that the *ACD* was negative, and if the line is grey it means no *ACD* at all had been received from the user it is pointing towards.

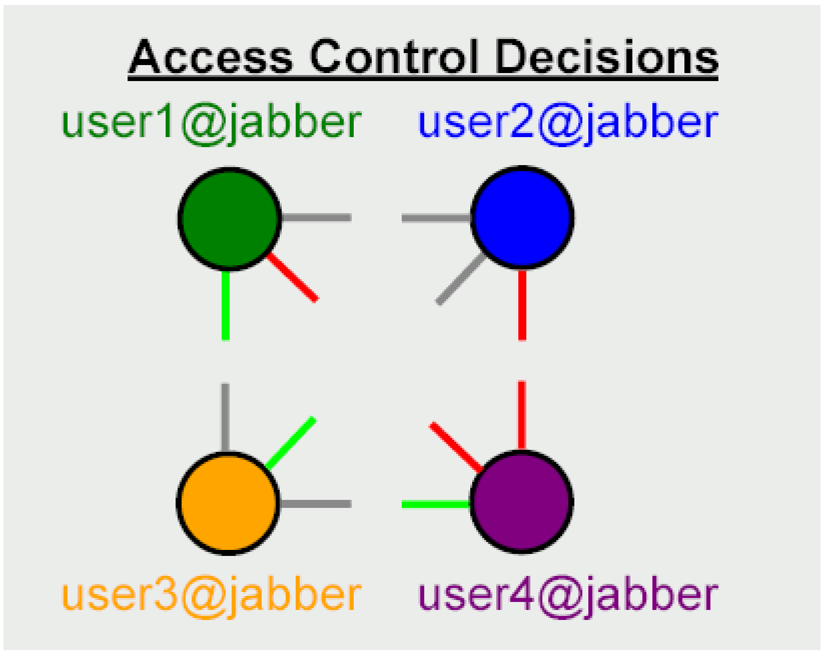


Figure 15: Visualisation of Access Control Decisions

For instance, in Figure 15 it is clear that *user1@jabber* has received a positive *ACD* from *user3@jabber* (i.e. it can receive *Location Updates* for *user3@jabber*) and that it has also been delivered a negative *ACD* from *user4@jabber*. This means that it will not receive any information as to the location of *user4@jabber*, because it is not on the same team. Furthermore, *user4@jabber* has received a positive *ACD* from *user3@jabber,* and negative *ACDs* from *user2@jabber* and *user1@jabber*. *User2@jabber* has not received any *ACDs* at all from *user1@jabber*, so the default is that it does not receive any *Location Updates* from him. Thus, this design will allow users of the simulator to see at a glance the current state of affairs regarding *ACDs* at any point of time in the experiment lifecycle.

3.8.2 Location Updates Visualisation

As with the visualisation of the *Access Control Decision* component it was decided after numerous design attempts to strip the visualisation of the Location Updates to the bare minimum in order to enhance the clarity of the information conveyed. To ensure design continuity, users inside the simulator were again represented by colour-coded circles, with their names printed in the corresponding colour. Whenever a *Location Update* event occurred, the visualisation component was invoked as in Figure 16, where *user1@jabber* received an update from *user2@jabber* that he was in "hall_Nth". The green line connecting the two circles reinforces the meaning that these users have an agreement that *Location Updates* should be sent from *user2@jabber* to *user1@jabber*, and echoes their linkage in the *ACD* panel.

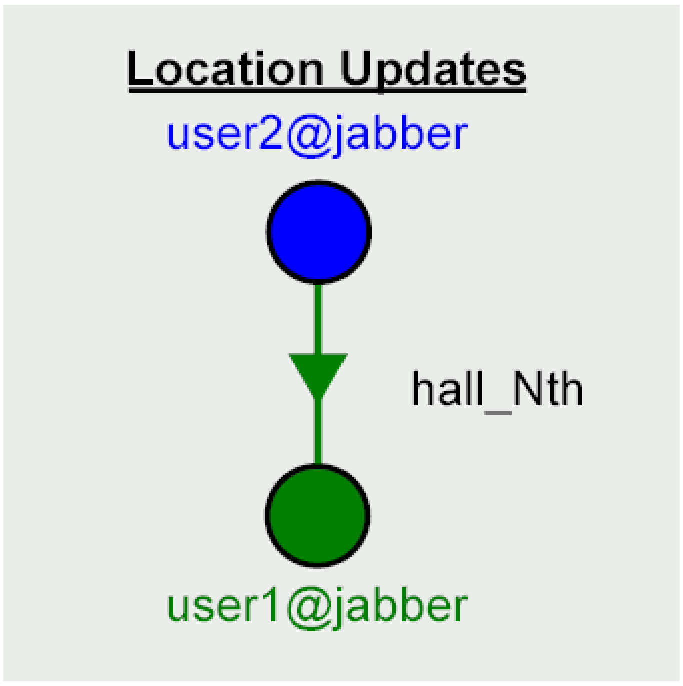


Figure 16: Visualisation of Positive Location Update

In Figure 17 it is clear that *user4@jabber* has received a *Location Update* from *user2@jabber* declining it any information regarding its location. The red line joining the circles reinforces this information and again should be consistent with the colours of the *ACD* component at that moment in time.

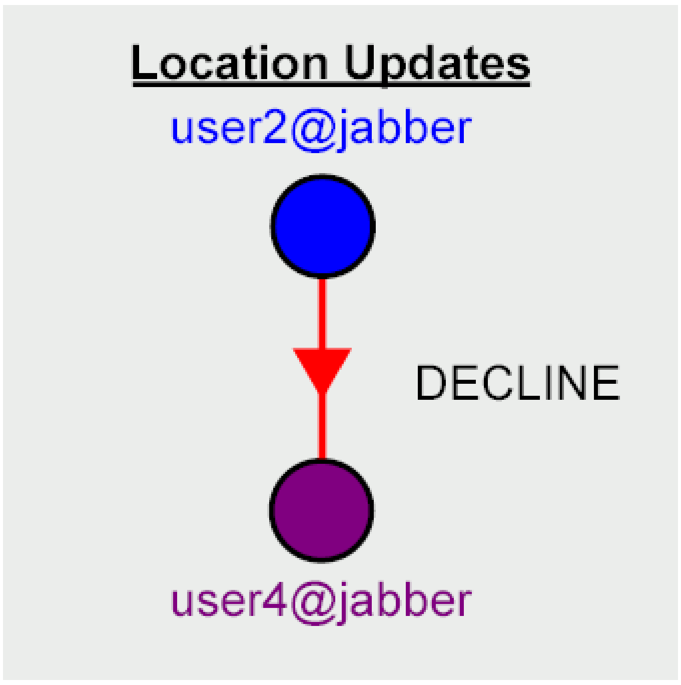


Figure 17: Visualisation of Negative Location Update

3.9 Separation of SUT and Floor Plan

The design of this system will ensure that there is a separation between the SUT component and its floor plan counterpart. They both use separate DOM objects to populate their displays and they both have distinct clocks. Thus, the only linkage between the two is the universal colour scheme of the avatars. Users work these components in tandem by aligning their clocks together, and analysing the two displays. Because there is nothing intrinsically linking the SUT to the floor plan, this design will allow the integration of a different SUT with minimal, if any, disruption to the floor plan code.

3.10 Integrating New Maps

Because of the design used in this project, it should be possible to replace the current floor plan with any SVG map. All that is needed, is the scale of the SVG floor plan in comparison to the 3D original, and the correct offset for *X* and *Y* values. This ensures that all avatar locations can be superimposed correctly onto the map. Furthermore, if adjustments need to be made to the existing floor plan (new sensors or furniture added), this can be accomplished easily by appending the relevant SVG code to floor plan document.

3.11 Supplementary Controls

It was clear that there needed to be a section of the interface that would allow supplementary functions, not central to the animation of the floor plan, be stored. Figure 18 highlights the *Show all locations* widget, which allows all the locations of an avatar, or group of avatars, to be displayed on the floor plan at the same time. A researcher can simply select or deselect any avatar they want using the checkboxes, and by then pressing the *Show User(s)* button, the appropriate locations will be superimposed on the map. The avatars' names are again colour co-ordinated with the rest of the interface to maintain consistency. A link to the map legend is also positioned in this section (a pop-up window displays the correlation between floor plan numbers and room names), and any additional visualisation functions can have their controls located here.

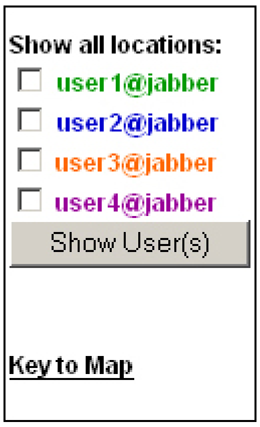


Figure 18: Supplementary Controls

3.12 Security Considerations:

Because the information the software relies on to visualise data comes from the XML output of the simulator and the SUT, any viewer of the 2D visualisation is relying on this log to be correct and uncorrupted. It is envisaged that the 2D visualisation will become a key component in evaluating the success of various SUTs and sensor configurations, thus in any large scale implementation there could be vested interests in effecting the results of such experiments. For instance it would be possible for a malicious user of the simulator to doctor the log, skewing the results to their benefit.

As the integrity of the data is paramount to the success of a fully implemented web based version of this application, it should be insisted that the database have no direct contact with the Internet, employing a DMZ as a buffer zone. All relevant databases should be hardened against attack with default/unused accounts deleted, insecure passwords changed, and access to the files limited to those with the correct authority. Furthermore, any important information stored should be backed up at regular intervals according to a strict backup policy, so that there can be easily retrievable in the case of data being corrupted. A final measure that should be employed, would be to ensure that all XML logs be digitally signed, in order to maintain their integrity and allow authentication.

A more general issue relating to the integrity of the XML log (though one which could be an exploitable security vulnerability) is the synchronisation of timestamps from the simulator, and timestamps from the various SUTs connected to it. In order for researchers to garner reliable information from their experiments, it must be ensured that each timestamp is accurate, and that their reliability has not been affected by a malicious user or a platform malfunction. Because the platform supports many separate components working in tandem over a distributed network, mechanisms must be put in place to guarantee that either a central log maintains accurate information, or that separate logs can have their timestamps co-ordinated for future visualisation.

Another obvious vulnerability of the web application is that there is a potential for hackers to eavesdrop on potentially sensitive information whilst in transport, such as the XML log. Hence I would recommend that TLS [24] be used to encrypt any information sent from the client to the web server, in order for its contents to remain private. It might be prudent to include client authentication for some administrator accounts as another step to protect against malicious remote access. Finally, as the translation of XML to SVG is the central function of the application, it should be ensured that the XSL files within it are protected from any tampering, as if not, it would be possible for an attacker to change these files resulting in misleading visualisations, despite the XML input being accurate and its authenticity verified.

3.13 Conclusion:

This chapter has introduced and discussed all the design aspects relating to this project. The technologies used and the underlying architecture of the system were examined, as were the types of XML messages that are central to the frameworks operation. General design principles were drawn up, and the design of each component of the visualisation tool presented in detail. Finally, relevant security considerations were discussed, as well as issues relating to the software’s extensibility.

Chapter 4: Implementation

The implementation of the visualiser's design posed numerous challenges that needed to be overcome in order to satisfy the project objectives set out in chapter one. This chapter describes the implementation process and the issues encountered in the course of its construction.

4.1 Converting Character Strings to XML

The format of the messages being outputted from the simulator meant that a lot of important data was contained in character strings rather than in XML. This meant that a *Find and Replace* operation needed too occur on the file before it could be loaded into a DOM object. The following code shows an example of the XML output, before and after the character replacement algorithm was implemented.

Before PHP *Find and Replace* algorithm is applied:

```
<Term typeID="1146147133000-85">
     <ReceivedFrom>ELVIN</ReceivedFrom>
     <SendingTo>CLIENT</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:12:13:484</Time>
     <TimeUTC>1146147133000</TimeUTC>
     <UserName/>
     <Direction>IN</Direction>
     <Message>&lt;ELVIN-NOTIFICATION&gt;&lt;ACCESS-CONTROL-
              DECISION&gt;ID&lt;/ACCESS-CONTROL-
              DECISION&gt;&lt;ACD-PAYLOAD&gt;DECLINE&lt;/ACD-
              PAYLOAD&gt;&lt;ACD-TARGET&gt;user1@jabber&lt;/ACD-
              TARGET&gt;&lt;ACD-SERVICE&gt;LOCATION&lt;/ACD-
              SERVICE&gt;&lt;ACD-SUBJECT&gt;user4@jabber&lt;/ACD-
              SUBJECT&gt;&lt;/ELVIN-NOTIFICATION&gt;
     </Message>
</Term>
```

After PHP *Find and Replace* algorithm is applied:

```
<Term typeID="1146147133000-85">
     <ReceivedFrom>ELVIN</ReceivedFrom>
     <SendingTo>CLIENT</SendingTo>
     <Date>27/4/2006</Date>
     <Time>2:12:13:484</Time>
     <TimeUTC>1146147133000</TimeUTC>
     <UserName/>
     <Direction>IN</Direction>
     <Message>
          <ELVIN-NOTIFICATION>
          <ACCESS-CONTROL-DECISION>ID</ACCESS-CONTROL-DECISION>
          <ACD-PAYLOAD>DECLINE</ACD-PAYLOAD>
          <ACD-TARGET>user1@jabber</ACD-TARGET>
          <ACD-SERVICE>LOCATION</ACD-SERVICE>
          <ACD-SUBJECT>user4@jabber</ACD-SUBJECT>
          </ELVIN-NOTIFICATION>
     </Message>
</Term>
```

4.2 Creating the SVG Floor Plan

Having exported a 2D raster image of the building floor plan from the Half Life 2 engine, Adobe Illustrator [25] was used to resize and trace the outline of the building. It was hoped that there would be an automatic tool that would parse the 3D map format and export out a 2D SVG version, however when this was attempted the resulting file was over 6MB in size, as opposed to a hand traced version of only 11K. This was largely due to thousands of smaller lines being created by the parser, where one larger one would be have been more efficient. Moreover, the parser missed some sections of the map altogether, which meant that it was not a viable option to use its output as the floor plan for this project. However, it is not unfeasible that an automatic tool could be developed in the future which would mean that the manual tracing of SVG maps would be eliminated.

A decision to deliberately strip down the map to a minimum of lines as possible was taken and the only the walls and doors were shown in the final version. This was essential to reduce visual clutter and noise. Numbers representing each of the rooms were superimposed onto the map, as well as an additional layer of room shapes that would turn green if a room sensor was triggered by a user entering the room. Though Adobe Illustrator produced the desirable result visually, it added a lot of superfluous SVG metadata which was unnecessary, as well as doubling the file size.

Hence a further processing step was taken by editing the file in Exchanger XML 3.2 [26] so that all that remained was the core SVG needed for the experiment.

Because this application was limited to one floor of the simulator it meant that the Z coordinate remained the same throughout the experiment, and that only the X and Y coordinates needed to be aligned with the SVG floor plan. The SVG map produced was scaled down by a factor of five from the simulator floor plan so that it would be small enough to fit comfortably on a screen without necessitating scrolling, but large enough to allow the clear visualisation of user and sensor activity within the simulator. In order to get the right mapping from the X and Y coordinates in the simulator to those in the SVG floor plan required some trial and error in getting the correct offset to apply, but once these figures were attained (x + 105, y + 284), it was straightforward to superimpose user movement onto the map.

4.3 Initialising Functions

Once the XML files have been loaded into a DOM object, a function is called to iterate through the XML and extract out all the event times and load them into an array. Because the timeline was 500 pixels long, it required subtracting the time of the first event from the last event to find out the duration of the experiment, and then dividing this figure by 500 to find out what unit of time a single pixel along the timeline would represent. Hence it was possible to position each event at an appropriate place on the timeline, so that users would be able to see in what context that particular event occurred. The same procedure was followed in the production of the SUT timeline.

In addition, another array was created that contained the IDs of the user relating to each event. After each ID was added, it was checked whether this was the first time this user had caused an event in the simulator; if so it was assigned a new colour to represent it from then on. As the IDs of users were being extracted from the XML, their X and Y co-ordinates were done so likewise. These were used to superimpose a relevant coloured circle on the map. However, once appended to the SVG file, each circle had its opacity set to zero. It would only become visible if triggered by the *play*, *next event* or *previous event* buttons. Thus in essence, when the map is loaded up, all events are already visualised on it, but they have their opacities set to zero.

If the next event occurring is an *enter room* event, then the name of the user and the room entered is passed to the appropriate function. This function locates the room in the SVG document by comparing the name with each room's unique ID. It then changes the rooms' colour attribute to green, and turns it back to white when the next event occurs. A textual display of this event i.e. *user2@jabber is entering the Atrium* is also appended beneath the event timeline, with the text of the avatar's name shown in the appropriate colour.

4.4 Implementation of Animation Controls

In this application there are two mechanisms implemented to control the animation of the map. One allows the user to play through the animation in real time, with further options to pause or stop the motion. The other allows for the animation to be progressed forwards or backwards on an event by event basis, which allows for a more forensic analysis of the interactions within the simulator.

With regards to the former, once the play button is hit, a JavaScript timer is instantiated which increases indefinitely. If the time of this counter is equal to or exceeds the time of the next event in the array of event times, then the circle representing this event is located on the map, and its opacity set to 100%. If the user responsible for this event has already had a previous event visualised, the older event has its opacity set to 0%.

As the animation is playing, the thumb of the timeline moves along from left to right to inform users of where the current part of the animation comes in relation to the entire piece. This was achieved by storing the final event time of the animation, and then continually passing the current time to a function which calculated what percentage of the animation had been completed. It was thus possible to position the thumb at the correct distance along the 500 pixel timeline.

If the pause button is hit during the animation, the time of the counter at that instant is stored, so that when play is resumed, it restarts from the correct position. Pressing the stop button resets the counter to zero and repositions the thumb of the timeline to the extreme left.

The other control mechanism uses *forward event* and *back event* buttons to progress the animation in either direction. There is also an event index which keeps track of what is the current event on display. Thus, when either button is pressed, the appropriate event has its opacity increased to 100%, and the event index increased or decreased depending on which button has been pushed. If another event by the same user is already on display on the map, it has its opacity reset to 0%.

When the animation is progressed forward or back, its representation on the timeline (as opposed to the map) is highlighted in green and enlarged slightly so that it gives users' the context of this events position in time. A similar mechanism of forward event and back event buttons is used to control the movement of time of the SUT events. Likewise its timeline follows the same structure as its map counterpart.

In order to reduce learning time for new users of the system, the control mechanisms were based upon standard icons and timeline metaphors that have been employed in countless applications. Hence, the *play*, *stop* and *pause* functions were instantly recognisable, and the combination of the timeline changing colour when either the *next event* or *previous event* buttons were pressed meant that controlling the animation was straightforward. The main technical challenge was to ensure that the three elements (buttons, timeline and clock) were working in tandem with each other, and not referencing separate events.

4.5 Implementation of SUT

In visualising the output of the SUT that was under consideration for this experiment, a number of issues had to be considered. It was essential that researchers could be aware of the current status of users at a glance, and that the meaning was intuitive for them. The SUT component was activated by users pressing on the *next event* or *previous event* buttons beneath the SUT timeline. When this occurred, it was first necessary to use a conditional statement to check what type of event (*ACD* or *Location Update*) needed to be visualised in the SUT component. If it was an *ACD*, the *subject*, the *target*, the *verdict*, and the *time*, were extracted from the DOM object. Likewise, if it was a *Location Update*, the *subject*, the *username*, the *payload*, and the *time* were the relevant fields.

Each of these nodes extracted from the DOM tree had a corresponding SVG element to reference. Thus if there was a new *ACD*, the SVG text holders for *subject*, *target* and *time* were updated with their new entries, and the corresponding *verdict* line had its colour attribute changed to green if the decision was positive, and red if negative. If a new *Location Update* needed to be visualised, then the SVG text holders for *subject*, *username*, *payload* and *time* were updated, and the *payload* line and arrow changed to green if positive and to red if declined. Furthermore, the circles representing the sender and the receiver had their colours matched to those of the *subject* and the *username* respectively.

4.6 Implementation of *Show All Locations*

In order to allow users see all the locations that an avatar (or avatars) had been sensed in the simulator, a combination of HTML check boxes, JavaScript, DOM and SVG were used. A researcher can select any number of users they want to see using checkboxes, and once this is submitted using the HTML form, a JavaScript function is automatically called. This function uses DOM to locate all the relevant circles in the SVG file, and sets their opacity to 100% if their checkbox has been ticked, and to 0% if not.

4.7 Discussion of Implementation

The implementation of the design described in the previous chapter was limited to a short experiment of less than two minutes, where avatars roamed around the simulator whilst the SUT (a Location-aware Instant Messenger) was operating. This was the only file available during the implementation period, but longer experiments would have been able to slot into the framework, without necessitating code changes. Some informal user testing also occurred during the implementation process, which resulted in the circles representing avatars being enlarged to make them easier to see. Furthermore, a version which showed the trails of avatars' movement in the simulator (fading in opacity the older the reading) was discontinued, after users found it cluttered the screen and created confusion.

Because many events were only milliseconds apart from each other, when signifying a current event on the timeline (by colouring the red line green, and enlarging it), it was often partially covered by the events immediately following it. However, despite this problem, it was still always possible to make out where on the timeline the active event was residing. Initially when text needed to be appended to the screen, such as updating the clock time, the previous SVG text node was removed and an updated text node appended. However, during implementation a more efficient method was discovered, which involved using a JavaScript command to update the content of an SVG text placeholder.

Throughout the implementation process, the lack of detailed JavaScript error messages hindered debugging, though was not detrimental to the overall result. However, due to time constraints, it wasn't possible to implement a fully debugged feature, that allowed users skip to anywhere in the animation by clicking on the timelines themselves. Hence it was left out of the final version, as it was more productive to devote time to other aspects of the project. However, the means to create such functionality does exist in this visualisation framework.

4.8 Conclusion

This chapter described the implementation process of both the visual and technical aspects of the project. Problems that were overcome in the course of the development and issues not resolved due to time constraints were also examined. Overall, the implementation of the design set out in chapter three was largely successful, and many of the techniques employed could be replicated in future versions of the software.

Chapter 5: Evaluation and Discussion

This chapter describes the qualitative evaluation of the visualisations employed and the functionality of the application. Expert users of the ubiquitous computing simulator, as well as those with little experience of the domain, were utilised,so as to get a wide opinion on the usability of the interface, and the choice of visual representation. All comments and suggestions were noted and the main findings described below.

5.1 Overview

In order to determine how effective the visual metaphors chosen were at conveying the correct information in an intuitive manner, a qualitative study of the project was undertaken. In all, thirteen users were involved in the evaluation of the visualisation system. Ten of these users had little or no familiarity with the ubiquitous computing simulator, whereas three of them (two PhD candidates who develop for the simulator, and one MSc candidate who designs SUTs) had considerable experience working with the platform.

The format of the evaluation was as follows. Users were presented with a short written description of the visualiser's function to read (see Appendix A for a full copy of the description, and tasks/questions set for evaluators) and then a brief tutorial regarding its operation was given. They then had two to three minutes to browse the interface at their own will. The participants were then given eight brief tasks to complete which involved interacting with the visualiser, and attempting to elicit the correct answers from the display. These tasks covered a range of operations that the visualiser provided, and offered an insight into how easy users would learn to correctly identify salient information. For instance, they were set tasks like *"Three Location Updates were sent at Thu, 27 Apr 2006 14:13:20 UTC, what were they?"*, and *"Does the Location Update at Thu, 27 Apr 2006 14:13:22 UTC contradict the* *information displayed by the ACD panel at that time, or confirm it?”*. Further to this, there were two additional questions which offered participants the opportunity to suggest ways of improving the clarity of data presentation, and the usability of the interface.

The cooperative evaluation observational technique was used while users performed the tasks, which meant encouraging them to *think aloud* and talk through their thought process [27]. The evaluator was able to ask the user for clarification of their actions, and the user could ask for help if they were having difficulty in completing a task. All comments and thought processes made by participants were noted for analysis later. Because three of the thirteen users were developers of systems for the ubiquitous computing simulator, there were four supplementary questions for them to answer, which explored the usefulness of a visualiser from the perspective of an expert user.

5.2 User Comments

The first thing that could be inferred from the evaluation was that there was no appreciable difference between the expert users and novice users in understanding the visual metaphors. However, because the experts were already familiar with the differences between *Location* messages and *Location Updates*, and the symbiotic relationship between *Location Updates* and *Access Control Decisions*, it meant that there was understandably a slightly shorter learning curve for them.

5.2.1 Colour Coding

The consistent colour coding of users in the simulator throughout all parts of the interface was very successful, with many evaluators commenting on the ease in which they could correspond avatars’ activities on the floor plan, to those in the SUT. Likewise the use of green and red to signify positive and negative actions in the SUT panel meant that users got an immediate understanding of the information being conveyed.

5.2.2 Animation Controllers

Currently there are two separate animation controllers for the floor plan, one which allows the automatic playing of events, and one which permits an event-by-event analysis of simulator interactions. In order to maximise screen real estate, a suggestion was made to combine these two controllers into one master controller, which would have the functionality of both. Though both controllers were found to be intuitive to use, it was clear that users would have appreciated the option to click anywhere on the timeline with the animation continuing from that point, rather than always having to begin at the start. Likewise, if a slow motion or double speed option had been implemented, users would have found the automatic playing of the animation more versatile.

Navigation wise, the consensus was that the interface and controllers were in appropriate locations, with the clear separation of the SUT component not affecting its ability to work in tandem with the floor map animation. In undertaking the tasks for evaluation it wasn't necessary for users to play the automatic map animation. However, depending on the type of sensors in the simulator and the type of SUT, evaluators mentioned how they envisioned that this functionality would be useful for focussing their attention on the relevant area, and allowing them to *step back* and get an instant overview of all interactions.

5.2.3 Timelines

Though both the SUT timeline and the simulator-events timeline each contain two different types of event (*ACD* & *Location Update*, and, *avatar location* & *enter room* respectively), all events are represented by a red line on their respective timelines. Some users felt that it would be an improvement to have a different colour or shape for each event type, which would mean that at a glance it could be determined where clusters of specific event types occurred. This technique would contrast favourably with having to search through a large XML file for specific event types.

Because many events happened within milliseconds of each other, it meant users couldn't distinguish a gap between some pairs of events on the timeline, however this did was not cited as hindrance to user understanding. However if longer experiments were to be visualised in the future, it would be prudent to offer a facility to zoom into the timeline, without using the inbuilt zoom function of Adobe SVG Viewer, which zooms that entire document.

5.2.4 SUT Visualisation

Though the overall consensus was that the *ACD* and *Location Updates* were visualised in an elegant manner, some users felt that the lines in the *ACD* component could have been thicker to ease visibility, and that an arrow on each line facing towards the circle would have reinforced the direction the *ACD* was coming from. Furthermore, when an *ACD* or *Location Update* event occurred, some felt that a textual display similar to one that occurs when an avatar triggers an *enter room* sensor should appear. By placing this beneath the SUT timeline, similar to the way the *enter room* text appears beneath the simulator-events timeline, it would improve clarity slightly.

With regards to the SUT component some users felt that the greying out of the inactive component was not clear enough, though the vast majority were able to distinguish between the two. Because the ACD display showed the state of all received ACDs, and not just new events as they occurred, as in the *Location Update* component, users agreed that it should always be displayed onscreen. However, a minority of users felt that instead of greying out the *Location Update* component, it should not appear at all unless it is active.

All users were able to comprehend easily how you could check for contradictions between what *Location Update* had been sent, and what the *ACD* status was at that time. They could see the potential this would have in debugging new SUTs, and it was suggested that a warning flag should appear automatically if contradictory information occurred, thus simplifying the debug task for developers. Though this warning flag would be specific to ACDs and Location Updates, future implementations could have a generic warning system that would be available for any type of SUT, if so required.

5.2.5 Room Numbers

When asked to locate the name of a room when given its number, a significant portion of users anticipated that the quickest way to do this would be to click, or hover, over the number on the map, so that a popup window or tool tip would display the room name. Though not a core feature of the system, tool tips containing the room name should be activated by hovering over the room number and this useful function should be implemented in future versions.

5.2.6 Show All User Locations

Because of SVG's lack of native support for interface widgets such as drop-down menus and check boxes, it was necessary to implement the *Show user locations* function in HTML. This was located on the top right hand side of the interface, and some evaluators felt that there wasn't enough cohesion between it and the floor plan it controlled. In hindsight, a more central location would have reinforced its connection with the map, though because of SVG's limitations it would not have been as straightforward to implement.

5.3 Expert Users' Suggestions

The users of the ubiquitous computing simulator offered some valuable suggestions so that future implementations could be of maximum help to their work. The benefits of 2D visualisation were already apparent to users when they noticed redundant *Location Updates* were being sent out, which would not have been as easy to spot from looking at the XML log. With this new knowledge (easily inferred through visualisation) it's possible to reduce the number of unnecessary messages propagating through the system, increasing platform performance. It was also mentioned that the visualisation technique used for *Location Updates* could easily be adapted to represent new features of the Location-aware Instant Messenger, such as a users' mood at a particular time. Another immediately obvious advantage that visualisation brought was the ability to see that that *enter room* messages were been triggered by sensors, even though an avatar was already present in that room. Again it was cited that this would not have been so apparent from looking at an XML database.

One benefit of 2D visualisation that became apparent to developers of ubiquitous computing experiments was that it would be much easier to isolate problems within the 3D simulator, as the Half-Life 2 game engine doesn't allow you to pause its output messages. Because these messages appear rapidly it can be hard to locate problems within them, and the task of debugging involves scouring a large text file. Thus the visual representation of XML messages means the complexity of this task is eased significantly. One suggested feature to add to the visualiser interface was the facility to have the current XML being visualised displayed onscreen in text, perhaps in a popup window. This would be applicable to both simulator and SUT messages, allowing direct comparison of the XML and its visual representation, thus ensuring that there has been no error in the transformation.

5.3.1 SVG Graphs

The use of graphs to display both simulator and SUT information was also suggested as another function that would add value to the visualisation tool. For example, it was mentioned that while testing a new SUT, it would be useful for graphs to display the latencies of messages propagating through the system. By automatically computing statistics and outputting them as graphs, it would be easier to spot trends and isolate areas of interest. Moreover, it was said that the ability to filter by message type would enhance this further, making it possible to infer from spikes in the graph that system performance was hindered by an overload of certain message types propagating through the platform.

5.3.2 Sensor Displays on the Floor Plan

Because there is potential for numerous types of sensors to reside inside the simulator (some outputting continuous fluctuating readings, others changing from one fixed state to another), developers stated that it would be useful to be able to display the current state of all sensors in layers above the map. The importance of being able to control whether these layers were visible or not was stressed by simulator users, as they wouldn’t want the 2D floor plan to be crowded with unnecessary information. Furthermore, graphs of a sensors’ (or group of sensors’) readings against time, were suggested as something that would help developers evaluate the setup of their experiments, and allow them to optimise their configurations. For instance, in a simulation of an emergency in a pervasive computing enabled building, graphs could help highlight the movement of users towards exits and give an insight into why the spread was not optimal.

5.3.3 Synching Time Controls

Other suggestions made by researchers included the recommendation of a switch that would allow users to lock the time of the animation component with its SUT counterpart, thus allowing greater flexibility in viewing experiments. Because the SUT example in this project gets its timestamp from the same machine as the simulator this would be relatively straightforward to implement. However, implementing this with other SUTs, who use a separate clock for timing, would mean that all times must be offset from the same point, so that they run in synch with each other. Furthermore, having the floor plan animation running in synch with a video capture of the simulator events was also mentioned as a useful way of reviewing experiments, as you could directly correlate the 3D world (with a first person point-of-view perspective) with its 2D overview counterpart.

5.3.4 SUT Configuration

The ability for users to configure the SUT component to their own needs was cited as a feature that would make this a powerful and flexible application for developers. For instance, a developer could create a network diagram through simple shapes in SVG, and associate each element of the map with an XML tag from the SUTs’ output. Thus it would be possible to view how messages were propagating throughout the entire system, giving researchers complete knowledge of the framework, and allowing the platform to be debugged in a much more efficient manner. The need for such a tool is exacerbated by the increasing complexity of the platform, and its aim of integrating many separate components (eXist databases [28], Interlocutor, Half-Life 2 simulator, Jabber Instant Messenger [29], etc) into one cohesive whole. It was stressed that this generic SUT tool should be built around the XML available and not visa versa.

5.3.5 Conclusion

Though there was agreement that visualisation offered considerable benefits in both the design and analysis of new experiments, it was stressed that the users would have to have full confidence in its robustness, and be able to adapt the software to their needs in an easy and intuitive manner. If too complicated to use, the end benefits wouldn't justify the learning curve, and if the integrity of the data displayed was not guaranteed, then it merely becomes another potential source of error in an already complicated framework.

5.4 Evaluation of SVG

There were a huge number of advantages to using SVG as the visualisation technology in this project. First and foremost, it is open source, text based and written in XML, which meant that it was easy to work with, and complemented the XML output coming from the simulator and its SUT. Moreover, it is easy to integrate with JavaScript and XSL, thus enabling graphics to be dynamically updated, which was essential to the work undertaken in this project.

One negative aspect of SVG is that its animation of big files can be sluggish if the computer does not have enough free memory. Other limitations include its lack of native GUI widgets (there are implementations, but they not part of the W3C specification and are still not robust), and the need for a separate plug-in, in order for most browsers to display SVG. There are also some minor cross-browser issues relating to SVG, thus this implementation is currently limited to running in the Adobe SVG Viewer on Microsoft Internet Explorer. However, since these are by some distance the most popular SVG Viewer and browser on the market, this is not a major issue at present.

Some new aspects of SVG elicited during this project, included the fact that complex SVG graphics could be drawn in packages such as Adobe Illustrator, CorelDRAW [30] and WebDraw [31], negating much tedious work. Likewise, Geographic Information System (GIS) [32] software can export their maps in SVG format allowing them to be integrated into SVG applications. Other features of SVG that weren't utilised in this incarnation of the visualiser were: its ability to have sound associated with visual elements; to have its graphics exported to print; to allow its text to be searched and indexed; and to have metadata associated with its elements. These have potential to become central components of any future implementation. In conclusion, the advantages of SVG far outweighs the negatives associated with it, and it is clear that it provides a powerful set of features that can be exploited in a dynamic data visualisation application.

5.5 Conclusion:

As has been shown in this chapter, there are a number of improvements that could be made to the visualisation techniques employed, and the functionality offered in this application. Fortunately, many of these changes though important, are relatively trivial to implement, and would enhance the potential of a future version of this software to become a central part of testing and evaluating new ubiquitous computing applications.

Some of the recommendations are very easy to undertake due to the flexible nature of SVG itself. For instance, changing the appearance of SVG elements, like the thickness of lines in the ACD component, and the colour of events along the timelines, is just a matter of editing their attributes in the SVG document. Likewise, making text appear on the SUT timeline whenever a new *ACD* is received, involves just a few lines of code to create a text holder for it, and then a short function to allow the new SVG text to be appended.

Because the SUT component is independent of the floor plan animation, it means that you could design a new SUT panel and slot it in place of the current one. Any new functions that need to be written can simply be appended to the SVG file. Likewise, because the XML data is loaded into a DOM object, if graphs of data need to be visualised, the raw information is already accessible, and all it requires is a template for the graph to be added to the document. Again, any new functions that need to be written can be attached to the SVG file, and their output visualised in the graph template.

If the new SUT contains completely different messages to the one in current use, a new XSL file to filter the XML can be created to replace the current one. Any new sensors that need to be added to the simulator, merely have there SVG representation added to a new layer of the floor plan. Each sensor is given a unique ID which can then be referenced by any function that needs to change their attributes. Because the floor plan is stored as an SVG text file, the task of updating maps with new features or sensors can be done so very efficiently.

Other features mentioned, such as a configurable SUT component would take considerably longer to implement. It is envisioned that this would allow users to construct their own visualisation component out of SVG shapes, and associate each element with an SUT output message, thus allowing them to visually represent the XML. This could take the form of a network diagram or of a more abstract shape, depending on which is more appropriate. It would be technically challenging to develop such a component, especially as an intuitive configuration menu is essential.

This menu would allow users to associate SUT messages with SVG shapes, and for them to input the attribute changes they would like occur in their visualisation. However, as has been shown in this project, the combination of SVG, JavaScript, HTML and XSL is versatile, and should enable such a tool to be implemented. Because of time constraints it wasn’t possible to implement the changes discussed above; however the major visualisations and functionality desired had already been achieved. Moreover, the overwhelming consensus was that the main objective of visualising the XML output in an elegant and useful manner had been completed.

Chapter 6: Future Work and Conclusions

This dissertation represents an initial investigation into an application with considerable potential for improving the efficiency of developing and evaluating ubiquitous computing simulators. In this final chapter, ideas for the future development of the visualisation tool are presented followed by some concluding remarks about the design and success of the project.

6.1 Future Work

Perhaps the most important thing to come out of the evaluation of this dissertation is the confirmation that a fully extensible version of this tool would be a large help to users of the ubiquitous computing simulator. Thus, it is vital for future versions to facilitate the easy integration of new floor plans and SUTs.

6.1.1 Configurable SUT Component

As touched briefly upon in the evaluation chapter, what would be of great benefit to researchers would be a configurable SUT visualisation component. It is envisioned that this would allow users to configure a visual representation of part, or all, of their SUT. This would be facilitated by allowing developers to *drag and drop* basic SVG elements (circles, squares, lines etc) and to connect them up into a visualisation of some SUT activity. By associating XML messages from the SUT output with the SVG elements, it would be possible to adjust their attributes (colour, size, etc) so as to visualise activity within the SUT in a meaningful manner. An intuitive menu system would have to be designed in order to make this process as straightforward as possible.

6.1.2 Automatic SVG Floor Plan Generation

In tandem with this easy configuration of new SUT visualisation components, a more automated process of importing new floor plans into the system would further improve the flexibility and extensibility of the platform. By automatically converting the BSP [33] files used by the Half-Life 2 gaming engine into SVG, rather than having to hand trace floor plans in Adobe Illustrator, it would reduce the time taken to create new maps considerably, though it is unlikely that it would completely eliminate the need for some human input. Tools such as wad2svg [34] exist for such a purpose, but unless the output is guaranteed to be of a high enough quality, the benefits gained by the automatic conversion may not outweigh the cost of having to fix them up to the requisite standard.

6.1.3 Multi-Storey Buildings

Other work that needs to be done relating to maps includes the necessity to integrate multiple floors from the same building into the one display. Because the Z values of avatars will change once they ascend or descend, it would be easy to transfer their icon from one floor plan to another. A more pressing issue would be how to maximise screen real estate when there are multiple floor plans to contend with. Depending on circumstances, it might be more prudent to adapt a 2.5D view of the building, with floors stacked on top of each other at an angle, or else to continue with the straight overhead projection of each floor. As this decision will be vital to the success of a multi-floor visualisation, it is important that the pros and cons of each are weighed up in detail before proceeding. In order for new floors to be integrated into the visualisation system smoothly, it is essential that the correct X and Y offset is found, thus allowing the coordinates of avatars moving in the simulator to be visualised in the correct location. Hence, it is recommended that a unique reference point in original maps is used, thus making it easier to integrate further floor plans.

6.1.4 Animation Controls & Automatic Graphs

With regards to functionality, the visualiser’s controls should allow users to jump to any part of the animation and play it back at a number of different speeds. It should also be possible to skip to any event on the timeline with just one click, rather than having to shuttle through them one event at a time. As discussed in Section 5.3.1, expert users of the simulator also responded positively to the idea of adding graphs of sensor and SUT information to the application. SVG is the ideal technology to deliver such functionality [35], and as such, future versions should incorporate this feature into the system.

6.1.5 Website & Real-Time Visualisation

As mentioned in Section 3.12 (Security Considerations:), it is envisioned that future versions of this software would be made available via a website, as well through copies running on local machines. The website would enable researchers to log on remotely to the system and load up XML output of experiments they had already conducted, or else use the website to configure new SUT visualisation components.

Though the tool is currently designed to aid the post-hoc analysis of simulator experiments, there is the prospect of using it for real-time visualisation of events. This would involve using a SAX (Simple API for XML) [36] parser instead of DOM, and a certain amount of reconfiguration to the system, but nothing overtly complicated. SAX is event based (call-backs are triggered by predefined tags) and is faster and more memory efficient than DOM. Thus, a real-time visualisation tool receiving a sequential stream of XML messages from the simulator and SUT, would be an ideal framework to employ SAX. This real-time visualisation could also be integrated into a management suite for ubiquitous computing systems that are implemented in real world. Thus administrators could use it to monitor a smart space environment, and perhaps as a tool to control the operation of sensors.

6.1.6 Integrating Semantic Intelligence

Another area of rich potential for future visualisation would be to integrate semantic intelligence to the rendering of simulator and SUT data. Current visualisation techniques attempt to address the problems associated with representing and interpreting large data sets. However, these techniques are semantically ignorant of the data sets, and are not sufficient to address the visualisation needs of highly dynamic information generated by contextually-driven ubiquitous computing environments. Through the application of semantic models to the visualisation of such dynamic information, the end user can cognitively understand and manage highly dynamic environments more efficiently and easily.

6.2 Conclusions

This project has shown how XML output from the KDEG ubiquitous computing simulator can be transformed into an elegant SVG visualisation of events. Both internal simulator events and their SUT counterparts can be represented, resulting in the previously underused XML log having a vital role in easing the complex process of configuring and evaluating new experiments. As well as visualising the XML data, the application allows users to view these events over time, either through an automatic playing of an animation, or by manually traversing the timeline event-by-event.

While this dissertation represents only the earliest stages of a complete solution to visualising virtual environments and their components in 2D, it has highlighted how future work based on this project can provide researchers with a robust, extensible tool, capable of visualising multi-floor simulations and various SUTs. Furthermore, it opens up the possibility of combining real-time analysis of experiments, as well as post-hoc, into one application. Such software would truly offer developers engaged in research with ubiquitous computing simulators a practical insight into their experiments, and would enable them to design and evaluate their work with considerable more efficiency.

Overall, the four objectives set out in the first chapter of this dissertation have been met: Simulator and SUT XML were visualised in a useful manner; post hoc analysis of these visualisations were shown to be of value to designers of simulator experiments; SVG, though not without its flaws, was shown to be an appropriate technology in which to undertake data visualisation; and extensive features that should be integrated into a more extensible future version were discussed. Thus, in conclusion, the work undertaken for this project, though challenging, has proved to be a worthwhile experiment and one that highlights the extra knowledge that data visualisation can bring to this burgeoning field of research.

Appendix A: Evaluation Questionnaire

The KDEG research group, in Trinity College Dublin, have built a 3D simulator which allows ubiquitous computing experiments to be developed and evaluated before being implemented in the real world. For instance, they are currently testing a Location-aware Instance Messaging system which allows users who have the correct permissions to see the location of other users on their team. To initiate this, a user sends a request which is either replied to positively or negatively depending on a number of factors. If a user receives a positive *Access Control Decision* (ACD) it means that they will periodically receive *Location Updates* for that person requested.

What the visualiser is doing is recording all the interactions from such an experiment in a coherent and accessible fashion so that developers have a better understanding of how successfully their test system worked. It consists of two main parts, the floor plan and its controls, and the system-under-test and its controls. The floor plan visualises the physical location of all users inside the simulator and highlights any time an *enter room* event has been triggered. This map animation can be played through or else examined on an event-by-event basis. You can also view all the locations that a user was at by utilising the *show all locations* function on the right hand side of the interface.

The *System-Under-Test* (SUT) section highlights the information being passed around in relation to the Location-aware Instant Messaging system. It visualises two important aspects of this, *ACDs* and *Location Updates*, and allows users to view them on an event-by-event basis. The *ACD* visualisation shows (at a specific point in time) which other users each person has either been granted access, refused access or has not asked for access yet. The *Location Updates* visualisation shows which user was sent it by whom, and if the user receiving had already received a positive *ACD* it shows the senders' location also. Thus the *SUT* visualisation is used in tandem with the floor plan to provide an all round view of what information the ubiquitous computing simulator and the *SUT* are providing its developers.

Usage and Interpretation Questions.

1. What happened in the simulator at Thu, 27 Apr 2006 14:13:21 UTC?
2. What is the name of the room marked 25?
3. Three Location Updates were sent at Thu, 27 Apr 2006 14:13:20 UTC, what were they?
4. At that time (Thu, 27 Apr 2006 14:13:20 UTC) what was the state of affairs with user1@jabber's Access Control Decisions?
5. When was the first time that user2@jabber and user3@jabber were in the same room?
6. What happened in the SUT at Thu, 27 Apr 2006 14:13:07 UTC?
7. Does the Location Update at Thu, 27 Apr 2006 14:13:22 UTC contradict the information displayed by the ACD panel at that time, or confirm it?
8. How many rooms did user2@jabber visit in the experiment?
9. What would you change about the visualisation in order to improve its usability?
10. What would you change about the visualisation in order to improve clarity?

Questions for Developers of Ubiquitous Computing Experiments.

1. Do you think a visualisation tool would be useful in debugging the XML output from SUTs?
2. As a developer of experiments in the Ubiquitous Computing simulator is it useful to have the facility to replay a visual overview of interactions and events of past tests?
3. What functionality would you require from a visualisation tool in order to maximise its usefulness to you?
4. Would graphs based on the readings of sensors in the simulator be of benefit to the configuration of experiments while developing prototype services?

Appendix B: Summary of Evaluation Notes

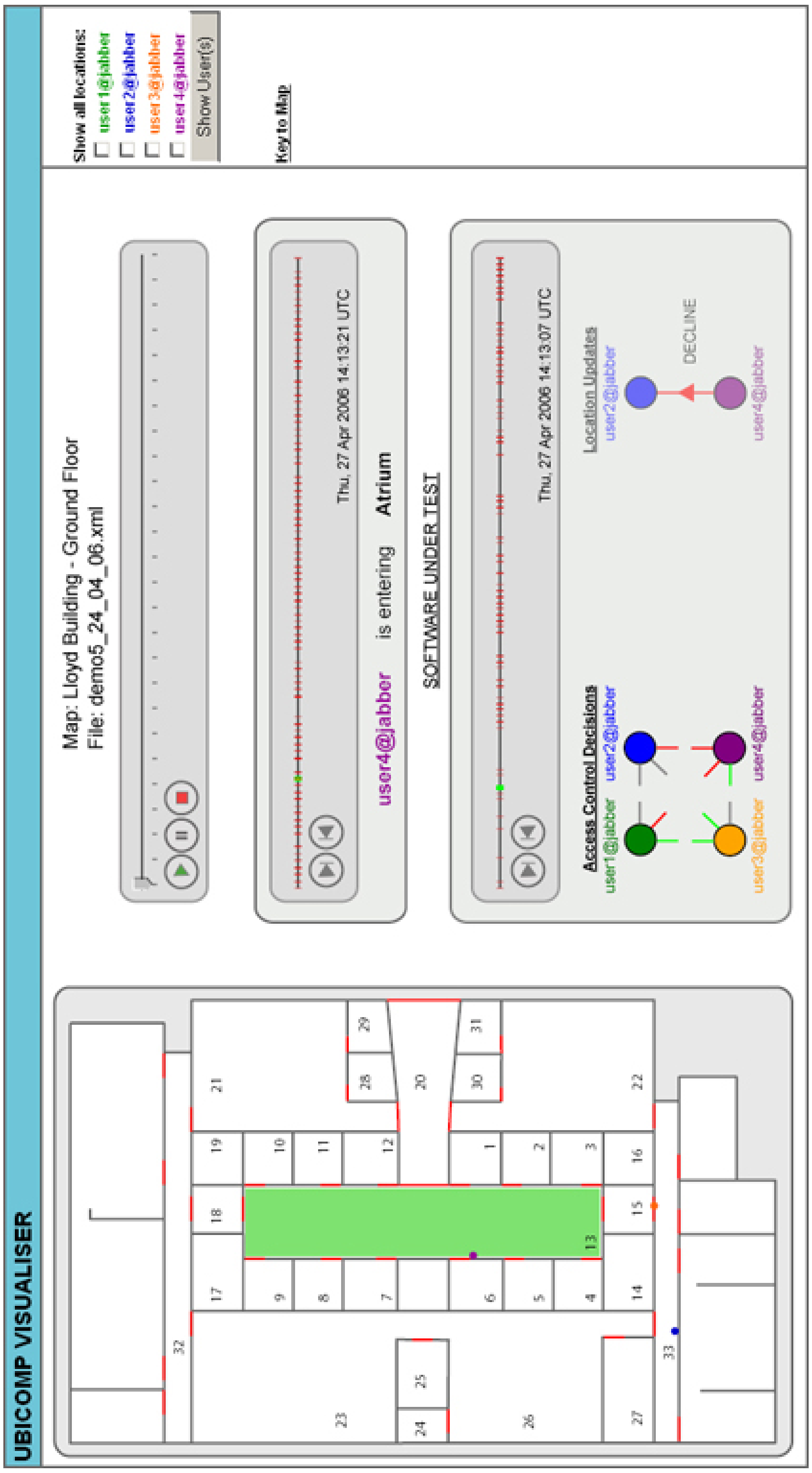
Questions	Notes
1.) What happened in the simulator at Thu, 27 Apr 2006 14:13:21 UTC?	All users found controls easy to use, though 2 suggested combining the two map controllers. 10 users mentioned how they would like to have been able to click ahead on the timeline. All 13 users found easily that an *enter room* sensor had been triggered.
2.) What is the name of the room marked 25?	9 out of 13 users went to the map expecting rollovers. 2 could not locate answer at all. Tool tips or popup window with name of room was suggested by 10 users
3.) Three Location Updates were sent at Thu, 27 Apr 2006 14:13:20 UTC, what were they?	Some non-expert users had to be reminded of the difference between location messages and Location Updates but all users got correct answer eventually. The use of green and red to signify whether it was positive or negative was said to be intuitive. Two expert-users mentioned how they could see superfluous Location Updates being sent which would not have been easy to spot in the XML log. This could now be prevented, thus improving system performance. The visualisation technique used for Location Updates could be adapted to represent new features of the Location-aware Instant Messenger, such as a users' mood at a particular time.
4.) At that time (Thu, 27 Apr 2006 14:13:20 UTC) what was the state of affairs with user1@jabber's Access Control Decision	12 users commented very favourably on the SUT visualisation, words such as elegant/neat/intuitive/clean/clever were used. All users got the correct answer.
5.) When was the first time that user2@jabber and user3@jabber were in the same room??	The consistent colour coding made it easy to equate avatars on the map with their names on the controls. 10 users mentioned this fact. 2 expert users noticed that "enter room" messages were been triggered by sensors, even though an avatar was already present in that room. It was mentioned that this would not have been easy to spot in the XML log.

Questions	Notes
6.) What happened in the SUT at Thu, 27 Apr 2006 14:13:07 UTC?	4 users mentioned that an arrow on each line facing towards the circle would reinforce who received ACD. 3 users felt that the lines in the ACD component could have been a bit thicker as wasn’t as clear as it could be. 12 users got the correct answer
7.) Does the Location Update at Thu, 27 Apr 2006 14:13:22 UTC contradict the information displayed by the ACD panel at that	3 users felt that the greying out of the inactive component was not clear enough. All users felt that the ACD should be permanently onscreen because it displays state information as well as events. 2 users would have preferred Location Updates to disappear altogether when not active. All users were able to get the right answer and saw the potential this would have in debugging new SUTs. 2 users suggested that a warning flag should appear automatically if contradictory information occurred. An expert user commented that future implementations could have a generic warning system that would be available for any type of SUT.
8.) How many rooms did user2@jabber visit in the experiment?	2 users thought that they would have to play through the animation keeping note as they went along. People found it easy to understand the information coming out of "show all users" but some struggled to find the control because of its location on the right hand side.
9.) What would you change about the visualisation in order to improve its usability?	A Slow of Fast motion option for the animation was mentioned by 5 users. Being able to skip to any region of the timeline was mentioned by 10. This would be even more necessary if the experiment was longer. 4 users mentioned that though they hadn't used the play animation feature, though they could see how it would provide a very useful overview in other experiments. 2 users recommended a switch that would allow them to lock the time of the animation component with its SUT counterpart.

Questions	Notes
10.) What would you change about the visualisation in order to improve clarity?	3 users thought that text should pop up on the SUT controller to accompany an event so that users wouldn’t have to look away from the timeline. 4 users felt that it would be an improvement to have a different colour or shape for each event type on a timeline. This would allow it to be determined at a glance where clusters of specific event types occurred, which is difficult to do with a text log. 3 users mentioned that you couldn’t distinguish a gap between some pairs of events on the timeline, however this did not hinder them much. Might be an idea to have a dedicated zoom tool for longer experiments. 8 users mentioned the clear separation of the SUT component did not affect its ability to work in tandem with the floor map animation.
11.) Do you think a visualisation tool would be useful in debugging the XML output from SUTs?	It was stressed that the users would have to have full confidence in its robustness, as if the integrity of the data displayed was not guaranteed it would become another potential source of error. It was also mentioned that it would be much easier to isolate problems within the 3D simulator, as that doesn’t allow you to pause its system messages which output rapidly. Debugging involves scouring a large text file so a visual representation of XML messages would be great. The comparison of ACDs with Location Updates highlighted was nicely done and could be extended to other messages.
12.) As a developer of experiments in the Ubiquitous Computing simulator is it useful to have the facility to replay a visual overview of interactions and events of past tests?	Really depends on type of experiment and what you're trying to prove. Having the floor plan animation running in synch with a video capture of the simulator events would also be a useful way of reviewing experiments, as you could directly compare the 2D and the 3D.

Questions	Notes
13.) What functionality would you require from a visualisation tool in order to maximise its usefulness to you?	It would be useful to be able to display the current state of all sensors, in layers above the map. Should be able to make these layers visible or not, so that the map doesn't get overcrowded. The facility to have the current XML being visualised, displayed onscreen in text would also be good. It would ensure that there has been no error in the transformation. The ability to configure the SUT component to their own needs was cited as paramount. A network diagram made through simple shapes in SVG could show how messages were propagating throughout the entire system. It was stressed that any generic SUT configuration tool should be built around the XML available, not forcing the XML to be changed.
14.) Would graphs based on the readings of sensors in the simulator be of benefit to the configuration of experiments while developing prototype services?	While testing a new SUT it would be useful for graphs to display the latencies of messages propagating through the system. Should be able to automatically compute statistics and output them as graphs so it would make spotting trends easier. Should also be possible to filter by message type. A sensor's readings graphed against time would help developers evaluate the setup of their experiments and allow them to optimise their configurations.

Appendix C: Screenshot of Visualiser Interface


UBICOMP VISUALISER

Map: Lloyd Building - Ground Floor
File: demo5_24_04_06.xml

Thu, 27 Apr 2006 14:13:21 UTC

user4@jabber is entering Atrium

SOFTWARE UNDER TEST

Thu, 27 Apr 2006 14:13:07 UTC

Access Control Decisions

Location Updates

DECLINE

Show all locations:
- ☐ user1@jabber
- ☐ user2@jabber
- ☐ user3@jabber
- ☐ user4@jabber

Show User(s)

Key to Map

Appendix D: Commonly Used Abbreviations

SVG	Scalable Vector Graphics
SUT	System Under Test
ACD	Access Control Decision
DOM	Document Object Model
SMIL	Synchronised Multimedia Integration Language
KDEG	Knowledge and Data Engineering Group

References:

1. Weiser, M., *Some Computer Science Issues in Ubiquitous Computing.* Communications of the ACM 1993. **36**(7): p. 74-84.

2. W3C. *Scalable Vector Graphics (SVG), XML Graphics for the Web.* http://www.w3.org/Graphics/SVG/

3. Accenture. *Executive Summary of Sensors*. 2004 http://www.accenture.com/Global/Research_and_Insights/By_Subject/Radio_Frequency_Identification/Sensors.htm

4. O'Neill, E., et al. *Rapid user-centred evaluation for context-aware systems*. *The XIII International Workshop on Design, Specification and Verification of Interactive Systems*. 2006. Dublin.

5. Valve Corporation. *Half-Life 2*. 2004 http://www.half-life2.com/

6. Kenny, A., D. Lewis, and D. O'Sullivan. *Interlocutor: Decentralised Infrastructure for Adaptive Interaction*. in *3rd International Workshop on Managing Ubiquitous Communications And Services*. 2006. Cork.

7. Nazari Shirehjini, A.A. and F. Klar. *3DSim: Rapid Prototyping Ambient Intelligence*. in *Joint Smart Objects & Ambient Intelligence Conference*. 2005. Grenoble.

8. *Universal Plug and Play*. http://www.upnp.org/

9. *HOUCOM, Framework for Collaborative Environments*. http://www.igd.fraunhofer.de/igd-a9/projects/houcom/index.html

10. Adobe. *Flash Homepage*. http://www.adobe.com/products/flash/flashpro/

11. Eclipse. *SWT: The Standard Widget Toolkit*. http://www.eclipse.org/swt/

12. *Creating a GUI with JFC/Swing*
http://java.sun.com/docs/books/tutorial/uiswing/

13. *The Flash ActionScript Developer Community*
http://www.actionscript.com

14. W3C. *The Extensible Stylesheet Language Family (XSL)*.
http://www.w3.org/Style/XSL/

15. Isakowski, Y. *Visualisation of dynamic glacier processes with SVG animation*. in *SVG Open 2005 conference*. 2005. Enschede.

16. O'Reilly. *The official Perl home page*.
http://www.perl.com/

17. W3C. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*
http://www.w3.org/TR/REC-smil/

18. W3C. *Document Object Model (DOM)*.
http://www.w3.org/DOM/

19. Peto, C. *GEMOS Building Management and Security system's SVG Interface*. *SVG Open 2005 conference*. 2005. Enschede.

20. *Association For Computer Aided Design in Architecture*.
http://www.acadia.org/.

21. Adobe. *Adobe SVG Viewer Homepage*.
http://www.adobe.com/svg/.

22. ECMA. *Standard ECMA-262, ECMAScript Language Specification*
http://www.ecma-international.org/publications/standards/Ecma-262.htm.

23. Carpendale, T., et al., *Extending Distortion Viewing from 2D to 3D.* IEEE Computer Graphics and Applications, 1997. **17**(4).

24. IETF. *Transport Layer Security (TLS)*.
http://www.ietf.org/html.charters/tls-charter.html

25. Adobe. *Adobe Illustrator Homepage*.
http://www.adobe.com/products/illustrator/.

26. Cladonia. *Exchanger XML Homepage*.
http://www.exchangerxml.com/.

27. Dix, A., et al., *Human-Computer Interaction, Third Edition*. 2004, Harlow, England: Pearson Prentice Hall.

28. *Open Source Native XML Database*. http://exist.sourceforge.net/.

29. Jabber Software Corporation, *Jabber Homepage*. http://www.jabber.org/

30. Corel Corporation. *CorelDraw Homepage* http://www.corel.com

31. Uniforce Software. *WebDraw Homepage*. http://webdraw.digitalcreation.us/default.htm

32. ESRI. *The Guide to Geographic Information Systems.* http://www.gis.com/

33. *BSP Homepage*. http://sourceforge.net/projects/doombsp

34. Phipps, C. *Wad2svg Homepage*. http://doombsp.sourceforge.net/wad2svg/

35. Adobe. *SVG Interactive Chart*. http://www.adobe.com/svg/demos/chart.html

36. *SAX Homepage* http://www.saxproject.org/