A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments

Eleanor O'Neill[1], Martin Klepal[2], David Lewis[1], Tony O'Donnell[1], Declan O'Sullivan[1], Dirk Pesch[2]

[1]*Knowledge and Data Engineering Group,*
Department of Computer Science, Trinity College Dublin
{oneille|Dave.Lewis|Tony.ODonnell|Declan.OSullivan}@cs.tcd.ie

[2]*Centre for Adaptive Wireless System*
Cork Institute of Technology
{mklepal|dpesch}@cit.ie

Abstract

Core to ubiquitous computing environments are adaptive software systems that adapt their behavior to the context in which the user is attempting the task the system aims to support. This context is strongly linked with the physical environment in which the task is being performed. The efficacy of such adaptive systems is thus highly dependent on the human perception of the provided system behavior within the context represented by that particular physical environment and social situation. However, effective evaluation of human interaction with adaptive ubiquitous computing technologies has been hindered by the cost and logistics of accurately controlling such environmental context. This paper describes TATUS, a ubiquitous computing simulator aimed at overcoming these cost and logistical issues. Based on a 3D games engine, the simulator has been designed to maximize usability and flexibility in the experimentation of adaptive ubiquitous computing systems. We also describe how this simulator is interfaced with a testbed for wireless communication domain simulation.

1. Introduction

Weiser's vision of pervasive or ubiquitous computing [1] involves devices, sensors and actuators that are inter-networked and embedded in the fabric of everyday life. These collaborate to sense and attend to the tasks users in a space are currently aiming to perform, adapting the behavior of available software services to meet that task while reflecting the context of the situation. More so than other distributed systems, ubiquitous computing systems must dynamically adapt to the needs of the user and to the current operational context. This requires ubiquitous computing systems to exhibit a high level of adaptive behavior in order to respond to a wide range of contexts and stimuli that may be difficult to define a priori in a comprehensive manner. In addition, the context and input stimuli experienced by adaptive software applications and the effectiveness of the adaptive behavior they exhibit are highly dependent on the user's involved and the environmental and social context in which they are attempting the tasks which the adaptive systems aim to support. Experimentation and testing of such adaptive software is therefore problematic due to the number of independent variables associated with such context. Accurate assessment of the perceived human benefits achieved by using such adaptive systems therefore requires that this context is accurately controllable, recordable and replicable. In developing such adaptive systems, human interactions must be recorded and later replicated in synchronization with their interactions with the physical ubiquitous computing environment. This experimental requirement is further complicated by the need to test scenarios that involve multiple users collaborating in a ubiquitous computing environment.

This paper introduces a ubiquitous computing simulator called TATUS that has been developed to support research and development of adaptive software for ubiquitous computing environments.

A system-under-test (SUT) connected to the simulator assimilates exported state in order to develop

its own representation of the world, as depicted in Figure 1. The SUT makes decisions to change its behavior in reaction to user movements and behavior as well as other environmental factors such as network conditions, ambient noise or social setting.

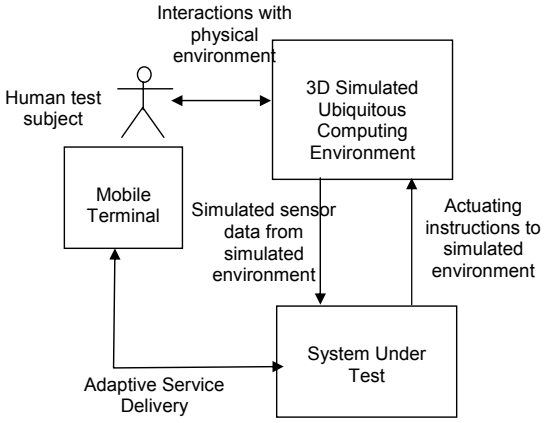


Figure 1. High-level simulator overview

When developing devices, protocols and software to support a ubiquitous computing environment, the common problem set encountered involves cost, logistics and location. Acquisition of adequate resources, such as embedded and hand-held devices, and their configuration within the various test scenarios applied to a SUT are complex, time-consuming and, therefore, expensive tasks. In addition, most ubiquitous computing applications require testing and experimentation in their target location for valid and beneficial results to be produced. Through careful device positioning, a standard office can be adapted to resemble another environment such as a living-room, however it is difficult to simulate large or specialized locations, for example an airport. In response to these issues, this platform provides a virtual ubiquitous computing environment. The simulator aims to reduce the complexity of setting up adaptive ubiquitous computing software tests by emulating sensors and actuators and allowing physical settings and scenarios to be easily set up by the researcher/tester. The test can be configured and monitored from a single desktop machine. The virtual world used in the test can be built and varied rapidly using intuitive graphical tools.

The simulator was initially designed to support various adaptive system research that is being conducted in the ubiquitous computing domain at Trinity College Dublin. Examples include: the use of Bayesian Networks to infer user intent from gestures and voice commands; the use of policies to constrain adaptive service behavior and the adaptive collection of context information.

In summary, therefore, the project aimed to develop a 3D simulator to satisfy the following objectives:

- To allow researchers/testers to readily connect a SUT to the simulator.
- To simulate conditions in a physical environment and to simulate corresponding sensor events.
- To allow researchers to define simulator events of interest for notification to the SUT.
- To provide researchers with a mechanism to allow the SUT to control actions within the simulator.
- To be flexible in handling a diversity of test scenarios
- To support connection to multiple SUT simultaneously.
- To readily reflect the changing state of ubiquitous computing technology especially new types of sensors and actuators.
- To replicate experiments using saved settings, both of test subjects' control of the simulated environment and the resulting simulated sensor events and adaptive software responses.
- To review experiments by rerunning recorded and logged experiments.

In the rest of this paper we examine previous attempts at ubiquitous computing experimentation with simulators, examine the characteristics of 3D games engines in this context, briefly present the TATUS simulator and review experiences in its application, before exploring its integration with a further, sophisticated heterogeneous wireless network simulator currently implementing IEEE802.11 and sensor networks.

2. Background

Techniques for evaluating and assessing ubiquitous computing environments have not yet been well-established [3]. A variety of practices are currently used to test ubiquitous computing environments, with many research groups developing testbeds specifically tailored towards their own experiments. This section presents existing approaches to simulation and testing of ubiquitous computing systems that influenced this work.

The Sentient Computing Project [4] is moving away from the conventional view that human-computer interaction is all premeditated and involves explicit deliberate actions with a computer interface. The project is working to develop applications that can

model a true representation of the world so that a person's natural surroundings become in essence a user interface. The natural movements and gestures of people occupying the space become the input commands to the application controlling the environment. The Sentient Computing Project has set up a physical test-environment which is replicated as a 3D graphical representation. The application under test receives input from sensors and actuators embedded in the room and uses the virtual representation to indicate its understanding of the status of devices, typically not visible in the real world, e.g. coloring the virtual representation of a phone red when it is in use.

Ricardo Morla and Nigel Davies have built and evaluated a location-based ubiquitous computing application using a hybrid test environment [5]. A wearable remote medical monitoring system was implemented as the test application. Using existing network and context simulators the team simulated the potential conditions that occur in a user's home e.g. temporary disconnection from the network. From this they were able to verify that the application does perform reliably under target conditions.

The UbiWise platform [2] targeted the development and testing of hardware and embedded software for ubiquitous computing devices. The mutual dependency between developing these two technologies, i.e. portable devices that are networked and context-aware, had been hindering real-world development of these types of devices. The UbiWise project aimed to simulate the existence of devices in order to develop the software that would run on them before physical prototypes are available. UbiWise emerged from an amalgamation of two existing simulators, UbiSim and WISE. UbiSim aimed to produce context information in real-time using a semi-realistic environment. It worked by taking raw simulated data outputted from the Quake III Arena (Q3A) [11] first person shooter gaming environment and processing it in the Context Toolkit. The context server was also capable of inserting data produced by real-world sensors and using the result to deliver meaningful context to applications and services. The WISE environment consists of a 2D display of a simulated device. The user manipulates the device to communicate with real-world Internet services or other simulated devices. The underlying supporting software is a Web Client that connects to the desired Internet services and interacts with the outside world via the HTTP protocol.

UbiWise thus made use of the graphical interfaces provided by each tool. UbiSim provides the 3D model of the simulated world that the user can navigate around in the first person manner as is normally done when playing a first-person shooter game. This is called the physical environment view. In this view, the user can call up a 2D view of the wireless device, with which they can interact, called the Device-Interaction View. This is a Java window as part of the WISE system where screen areas are mapped to particular device buttons so the device can be controlled by a mouse and keyboard. UbiWise offers three usage roles. The first role, the user, interacts with the simulated environment when running an experiment or playing out a scenario. This involves navigating around the 3D world using the game controls or using the mouse and keyboard to interact with the Device-Interaction view.

The second role is that of a researcher where the user adjusts the simulated environment, pre-run time, setting up the world to suit a particular scenario. Generally when using this type of simulator it is necessary to consider in advance the actions that will be carried out. For example a meeting requires a conference room with appropriate facilities e.g. large table but to execute a lecture there must be projection facilities in the room.

The third role is that of a developer, it is the most technical role and is filled by anyone extending the simulator to improve the ubiquitous computing environment. This includes incorporation of new devices and wireless media. A developer must therefore understand the underlying software structure of the simulator, including the underlying games software.

Ubiwise thus focussed on device emulation with a simulated 3D environment and therefore did not seem to develop the original context-awareness capabilities (which were focussed on character position) to make the configuration of a simulated sensor environment more flexible. This was however a major requirement for our simulator so a slightly different approach was required, where less emphasis was place on the developer role for configuring different experiments, with the aim that this can be conducted by the researcher, without the need for any game code modification. In addition, our simulator differs from the Ubiwise in that it does not attempt to simulate user devices, but instead relies on the test subject accessing adaptive services via devices such as PDAs which they manipulate during the experiment. This decision supported the aim of keeping the modification of games code to a minimum, so all complex human-device interaction is performed external to the simulator, and thus more realistically through the use of actual devices.

2.1. 3D FPS Games

Many of the 3D first-person-shooter (FPS) network games released for PCs since the late 1990's have also

released software development kits (SDKs). These SDKs allow programmers to modify the game through the inclusion of new rules, physics, weapons and characters. The term mod is appropriately used to refer to the games resulting from such adjustments.

The general aim in choosing to use one of these games was to exploit the 3D graphics engine to provide a realistic user experience, while mapping the projects requirements into the SDK code to provide a readily configurable test-bed for researchers. In addition the LAN style implementation of these games provides potential for multiple researchers to interact in a single experiment. Finally, the SDK also provides limited AI capabilities and scripted sequences to include non-player-characters (NPCs) allowing a single researcher to run tests independently.

When choosing the game engine for this project the potential candidates were Half-Life [7], Quake III Arena [11] and Unreal Tournament [12]. These games are designed on the same basic principles and in fact Half-Life is derived from Quake with about 30% of the original code remaining at its core. The Half-Life SDK [8] was selected based on its use of standard C/C++ rather than proprietary languages and the availability of relatively comprehensive, albeit unofficial, SDK documentation [10].

2.2. Introduction to Half-Life

Half-Life (HL) uses a client-server architecture allowing up to 32 players to compete in a single game. Each client has enough built-in artificial intelligence to estimate player movements in the case of lost messages from the server, correcting to the true picture of the world when contact is re-established. A total-conversion of HL involves creating new maps, weapons, characters, physics and rules. The simplest method to modify the game uses map creation alone. To adjust or add new instances of the remaining items, the HL SDK must be reprogrammed.

2.2.1.Map Creation

New maps or levels can provide the illusion of an entirely new game. Every world is a combination of basic shapes but through careful application of textures the map's terrain can be varied dramatically. Valve released a map editor called Hammer [9] which is a drawing tool for building maps. Hammer compiles maps to the BSP (Binary Space Partitioning) format used by Half-Life.

BSP files have been designed to improve game play by minimizing the calculations involved at run-time when drawing the environment. The BSP file saves the topology of a map as a binary tree. Objects that are geographically close in the map are stored in neighboring nodes within the tree. In addition to the BSP file, Hammer produces a MAP file which provides a textual representation of the world. It lists information about objects in the world such as their name, type, size and coordinates. A view of Hammer can be seen in Figure 1, showing the map building views and a navigable view of the environment being generated.

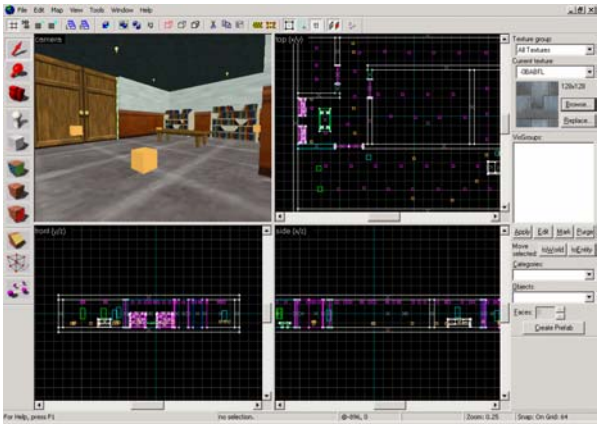


Figure 2. Hammer games editor for Half-Life

The principle elements of a map are brushes and entities. Brushes are the three dimensional solid objects that represent the physical structure of the room e.g. walls, doors, furniture. Textures are applied to these blocks to give a realistic representation of a door, whiteboard, carpet and walls. Entities, on the other hand, are neither visible nor physically tangible during game-play. They exist only through the effects they supply to a map e.g. sound/light. Hammer shows the positioning of entities through the use of icons e.g. the light bulb at the centre of Figure 3. Entity-Brushes are the result of selecting a brush and associating an entity with it using a technique called tying which can be performed via the map editor. When tied, the combination provides a functional object e.g. a door or button.

Triggers are essentially entity-brushes, however unlike the example of an entity-brush that is a door, triggers are invisible during game-play. They are used to generate events based on a player's movements and location. For this reason, they must be invisible so that it is not possible to consciously avoid them. When a player enters a region of a map occupied by a trigger the associated event is activated e.g. a door is opened. As a result, a normal entity alone cannot act as a trigger since the boundary of the trigger must be detectable to the game engine. In Figure 3 triggers can be seen as the patterned bars surrounding the door and the lecturn. In this instance the triggers are present to open the door when a player approaches. During the

setup process for the trigger, the door's targetname is stored as the target for the trigger. At runtime the engine can perform a lookup using the target value to search for the entity to be activated.

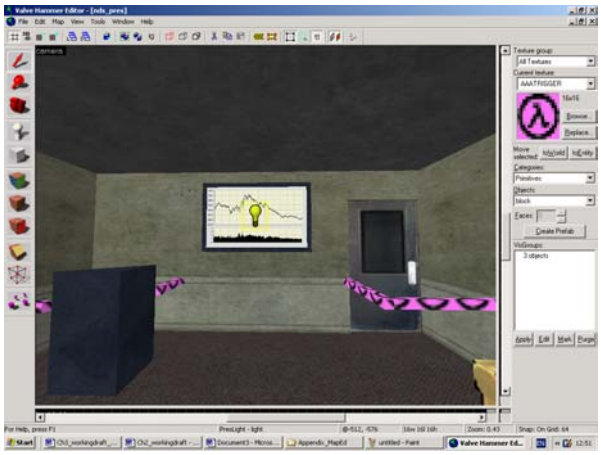


Figure 3. Map editor's rendering of a prototype space

Together these features of the HL SDK provide the flexibility needed to model a variety of sensors and actuators. Only simple actuation is possible, since entities at most support a single un-parameterised 'use' action. However, the ability to extract position of entities, proximity of other entities with a given radius and presence of other entities within a field of view allows for a large range of sensor types to be simulated.

3. Ubiquitous Computing Simulator

Though the HL SDK makes it is easy to isolate modification of the game's code to a limited set of files, the discovery stage and learning curve were both difficult and time-consuming. As a result it was deemed too complex to require researchers or test staff to have to ascend that curve in order to effectively use the simulator. We therefore aimed to provide a convenient and flexible 3D virtual ubiquitous computing environment that researchers can use to test ubiquitous computing applications currently under development, without the need to develop game level code. The resulting features of the simulator are:

- 3D Graphical Interface: Provision of a 3D interactive graphical user interface using the SDK for simultaneous use by a number of test-subjects
- Separation of simulator and SUT: The SUT is physically separate from the games engine, running on another machine if necessary. Simultaneous connection to multiple SUTs is supported.
- Realism: The simulated ubiquitous computing environment realistically modes the equivalent real-world physical implementation of sensors and actuator. The framework emphasizes avoiding the use of the precise positional and attitudinal information available form the simulator, and simulating actual sensor data instead.
- Flexibility: The simulator supports the test of a range of software. It is generic and not tailored to provide specific state or to interface to a particular piece of software, through the use of a proxy that mediates messages from the games engine to the SUTs
- Usability: The configuration of an experiment is conducted entirely through the combination of existing map editors and an additional message definition tool which allows the information passed to and from entities in a specific map to be defined via a simple GUI.
- Extensibility: The underlying SDK, though not typically used by a tester, can be readily adapted to extend the features offered by the simulator framework.
- SUT API: This offers selectable state extraction. Researchers are provided with a mechanism to select a subset of the state information most suited to the goals of the SUT experiment. This is to avoid a full state dump, potentially containing surplus data and requiring unnecessary processing by the researcher. It also offers an interface to impose changes (i.e. actuation) on the simulated ubiquitous computing environment.

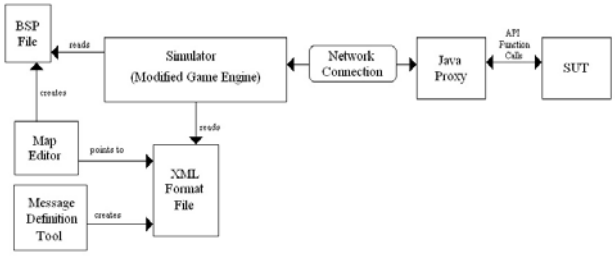


Figure 4. Design Overview of Simulator Framework

An overview of the design is shown in Figure 4. Binary Space Partitioning (BSP) files are generated by the map editor and fed into the simulator, while the message definition tool provides XML definitions of the state extraction and actuation message information to be passed between the simulator and the SUT during the experiment. These XML definitions are referenced by the map editor and fed into the simulator with the BSP files defining the environment.

The modified game engine and SUT needed some method of communication to exchange information. Messages travelling outbound from the simulator contain state information about the simulated environment. Messages traveling inbound to the simulator contain instructions to adjust the simulated environment. Two specific feature of this framework are:

- Network Connection: The network connection allows the simulator and SUT to run on separate computers. This is important because when both programs are run in parallel on a single machine the simulator's graphics absorb the entire screen. In addition, the keyboard and mouse are dominated by Half-Life's player controls. Running each program on a separate machine means a researcher can view and control both programs concurrently. This is particularly relevant when debugging test software.
- Proxy: The Proxy removes any need to integrate a network connection into the SUT code by providing a ready-made link to the simulator. This is supplied with a view to reducing set-up time when initially connecting new SUT to TATUS. The Proxy also provides an API that offers function calls to send and receive messages to and from the simulator.

Figures 5 and 6 show two screen shots from a ubiquitous computing meeting room scenario. The trigger entities visible in the map editors view of this scenario are invisible in these views, as are the trigger entities used to detect the characters in Figure 6 standing up and sitting down. The other characters in Figure 6 are non-player characters, though in this particular test scenario the test operator can control when the standing figure gets to his feet and subsequently sits down through the use of XML commands sent from the proxy-level. The other characters are controlled by script based AI behavior. In an alternative collaborative test scenario, these characters could be controlled by other human test subjects on a remote game clients.

Initial usage of the simulator by a TCD researcher has shown that single room scenarios can be configured using the map editor and message definition tool in a matter of hours.

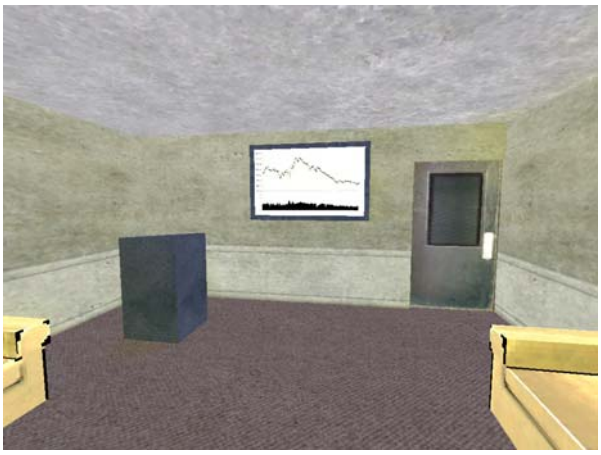


Figure 5. Screen shot from meeting room scenario

Figure 6. Screen shot from simulated meeting room scenarios showing other characters

4. User Interaction with Wireless Networks

Currently TATUS does not simulate any aspect of the communications networks that must support any operational ubiquitous computing environment. The application of wireless access network technologies to ubiquitous computing environment, e.g. 802.11, Bluetooth, UWB, 3G and their fusion in 4G is an active area of research as is research into network architectures that addresses the intermittent connectivity, lack of fixed infrastructure and limited power requirements that characterizes ubiquitous computing, e.g. mobile ad hoc networks and sensor networks.

A wide range of wireless network simulators exist that support this type of research. However they typically use statistical models of user behavior, thus not allowing for evaluation that is able to assess the reaction of human test subjects to the adaptive

behavior wireless networks can provide. We have already started examining integration with simulators that would enhance the model of the sensed environment, in particular the TOSSIM TinyOS sensor network simulator [17], which would provide a more realistic view of sensor network latency and sensor reachability problems.

In this section, however, we will present in more detail work that is underway in interfacing TATUS with a heterogeneous wireless network simulator that will allow us to evaluate the impact user behavior and actions in the environment have on the wireless network and indeed vice-versa.

4.1 Wireless Network Simulator

In order to evaluate the performance of a communication systems, especially those used in ubiquitous computing environments, a system simulation is being developed that comprises of six main components as shown in Figure 7.

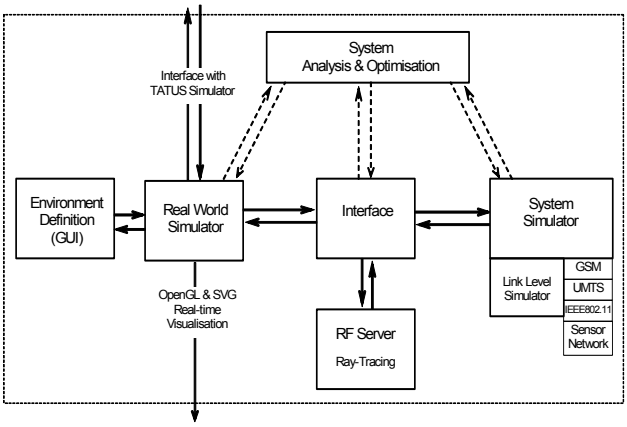


Figure 7. Structure of the Simulator

- **Environment Definition**: Graphical user interface for the description of environment configuration, heterogeneous network infrastructure, user behaviour, etc. The environment definition is compatible with the BSP file based environment description of TATUS and in a joint simulation case user behavior is provided by TATUS.
- **Real World Simulator**: Implements various models that capture behavior of entities in the simulated environment in a realistic form, especially people and vehicle mobility, models of environment ambient parameters as light, temperature, humidity, noise etc. This component also implements a web services interface for interfacing with the TATUS simulator.
- **RF Server**: Component containing a ray-tracing tool based on the Motif Model [14] to predict wireless channel condition for space-time specific environment configuration, RF actuators range, etc.
- **System Simulator**: this component comprises of a number of subcomponents for the simulation of different wireless network services..
- **System Analysis & Optimization**: Captures and analyses performance of the simulated system. It also implements a optimization engine based on evolutionary computing strategies for optimization of the system performance or infrastructure topology/layout.
- **Interface:** Management system interfacing the main components of the simulation environment, which is a highly distributed system.

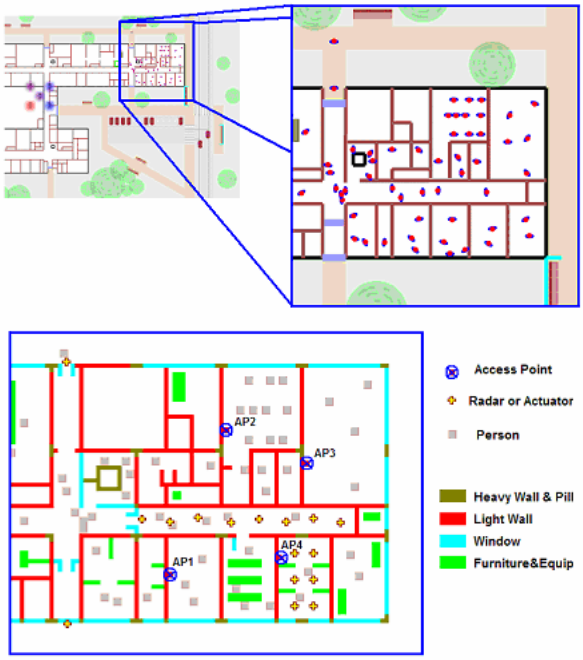


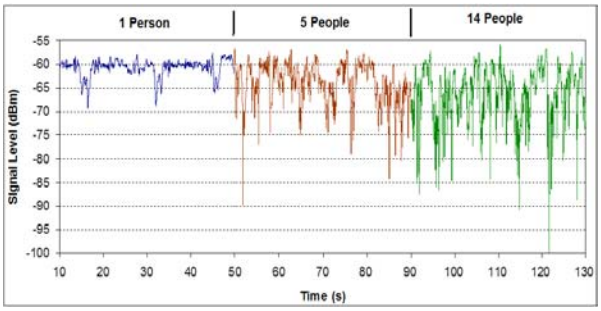
Figure 8. Example of environment segmentation for ray-tracing prediction

4.2 Wireless Link Parameters Prediction

In order to predict wireless channel conditions and thus wireless system performance as realistically as possible for a particular environment configuration, the influence of presence and mobility of people on the channel conditions was studied and associated models implemented into the simulation system. The effects of

people shadowing are especially important in indoor environments, where the deployed wireless systems provide wireless access through access points (APs) placed in convenient locations such as on ceilings, walls or some times even placed on desks near which wireless access is desired. From the radio wave propagation point of view, the signal between the AP and the user terminal propagates rather horizontally over the coverage area, crossing obstacles of various types such as desks, chairs and people etc. The net effect is an attenuation caused by static obstacles and a more varying signal due to moving obstacles such as people. As a consequence, there are rapid and frequent transitions between line-of-site and non-line-of-site situations, causing a variation in the statistics of fast fading, which is closely associated with the shadowing process. The characteristic of shadowing caused due to moving people resembles fast fading in other propagation environments.

Figure 9 shows an example of measured signal level fluctuation when a person is randomly crossing LOS between transmitter and receiver. The description of the measurement and ray-tracing prediction of effects of moving people shadowing on the performance of a wireless system for both LOS and NLOS case can be found in [15, 16].



b)

Figure 9. Sample of measured signal level fluctuation when 1 person, 5 people and 14 people are randomly crossing LOS between transmitter and receiver

In order to accurately predict the signal quality in the channel, at every point of the investigated scenario, all parameters, except AWGN (Additive White Gaussian Noise), must be site-specifically predicted. Path loss and channel parameter prediction (Figure 11) are performed by a deterministic ray-tracing model known as Motif Model [14]. The prediction of the level of interference is based on the appropriately filtered mean signal level predicted from surrounding interferers such as access points, microwave ovens and other appliances.

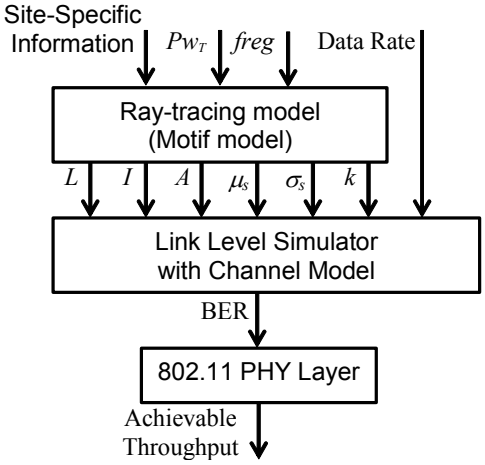


Figure 10. Architecture of the Wireless Link Parameters Prediction

Once the empirical parameters are delivered by the propagation model, they are used as inputs to the Link Level Simulator, using the channel model described above, and an estimation of BER can be obtained and used as input for the evaluation of IEEE802.11 WLAN performance. Figures 10 to 15 show site-specific predictions of channel parameters in every point of the investigated environment section.

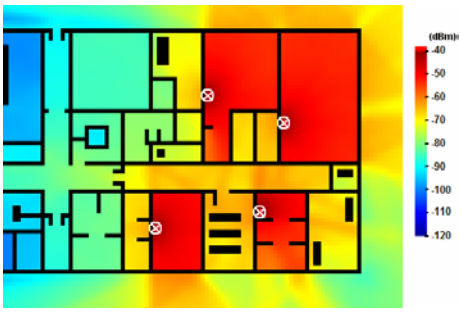


Figure 10. Signal level experienced by users of IEEE802.11 WLAN

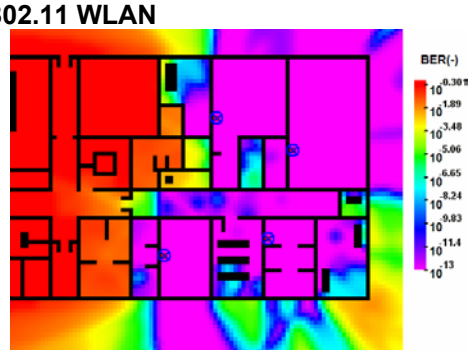


Figure 11. BER experienced by the users of IEEE802.11 WLAN (CCK 11)

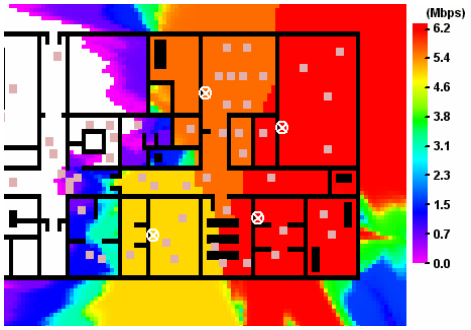


Figure 12. Throughput experienced by the users of IEEE802.11b WLAN

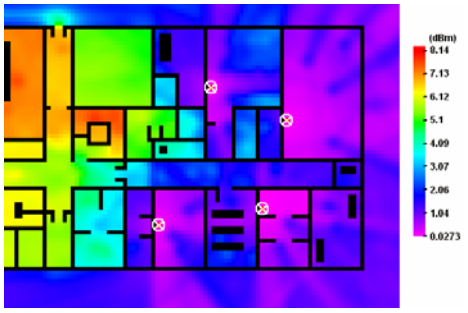


Figure 13. Extra signal attenuation caused by people

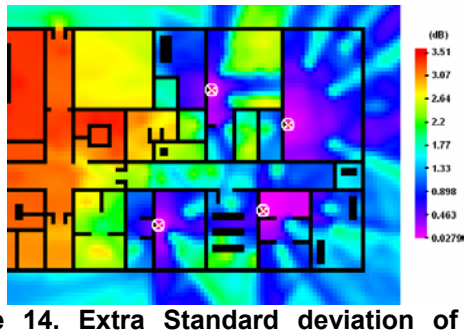


Figure 14. Extra Standard deviation of the signal fluctuation caused by people

4.3 Simulator Integration

In order to assess the impact of people movement and the effects of wireless communications usage an integration of the two simulators is desirable. This integration involves:

- The use of the same BSP based representation of the test environment for both simulators.
- The passing of character movement from TATUS to the wireless simulator so that the resulting changes in wireless signal propagation can be used to define 802.11 throughput calculations. The wireless simulator can also be used to predict the accuracy of location tracking mechanisms that use 802.11 signal strength.
- Throughput figures can then be used to control application data flows between the SUT and user terminals, e.g. a PDA, used by the test subjects, in concert with the simulator, using a suitable network emulator, e.g. DummyNet [18]

Integration between the platforms will take a Web Service approach similar to that described in [5]. This allows for flexible deployment of elements that make up the joint simulator.

Figure 15 depicts the target operational configuration for the integration of the two simulators. A single map editing tool will generate maps formats for both TATUS and the Wireless Network Simulator. TATUS provides the SUT with stream of sensor data as configured by the editor, while it provides the Wireless Network Simulator with real-time updates of the position of all humans in the simulator, both player driven and non-player characters. The Wireless Network Simulator is then in a position to use the resulting updates to its wireless signal propagation map to provide accuracy-adjusted location tracking data to the SUT that reflects the errors introduces in 802.11-based tracking due to signal fluctuations. It can also use similar data to control a traffic throttle placed between the SUT and the PDA used by the test subject to access ubiquitous computing services offered by the SUT, thus allowing the impact of signal fluctuations on these services to be assessed.

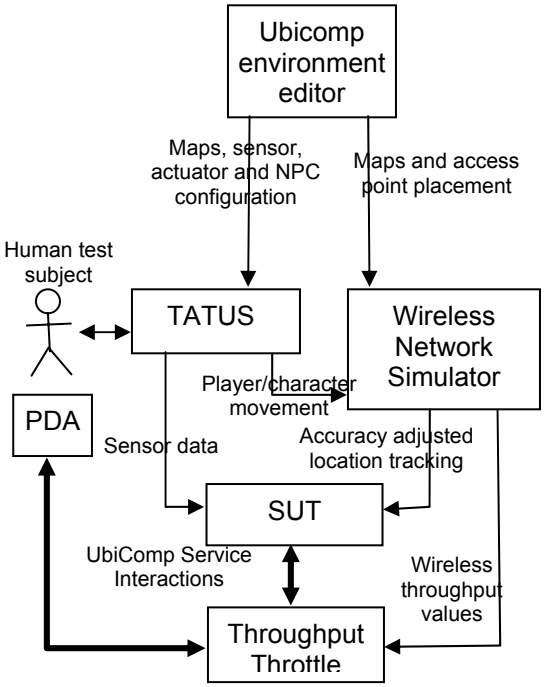


Figure 15: Operation of integrated simulators

6. Conclusions

In this paper we attempt to identify the difficulties of experimenting with and testing adaptive systems developed for ubiquitous computing environments. Such adaptive systems must react to changing user requirements and preferences, network connectivity and available services i.e. they are context aware. Controlling, recording and replicating such context and the corresponding human interaction with adaptive systems is a challenging and typically expensive undertaking, and hinders the development of effective adaptive software for ubiquitous computing. We describe a simulator that builds on experience of previous ubiquitous computing simulators that exploit the capabilities of first person 3D games engines to support such testing and evaluation activities. This simulator, TATUS, is novel in that it removes the need for experimenters to develop games level code, while retaining a large level of flexibility in the scenarios that can be readily developed by researchers. A different simulator focusing on the variation of wireless throughput that occurs in populated indoor environment is also reviewed and the integration of these two simulators into a comprehensive, person-centered testbed for adaptive ubiquitous computing systems is described.

Future work will involve detailed evaluation of the integration of the two simulators. We need to assess in particular whether the limited interaction between the user and simulated environment has a significant impact on results. We are constructing a live ubiquitous computing environment in which scenarios that might require such interaction can be conducted and user evaluation compared with the equivalent simulator-based experiments. We will also examine different types of APIs for use by the SUT, in particular ones that replicate existing sensor networks, such as TinyDB [6]. Simulating the fidelity of simulated sensors is also an important issue, with more sophisticated models being built using the map editor to more accurately reflect sensor field of view and range and within those variations of sensing accuracy.

Acknowledgements

This work was partially funded by the Irish Higher Education Authority under the M-Zones research programme.

References

[1] The Computer for the Twenty-First Century Mark Weiser Scientific American, 1991,Vol 265, No. 3, pp. 94-104

[2] UbiWise, A Ubiquitous Wireless Infrastructure Simulation Environment John J.Barton, HP Labs Vikram Vijayaraghavan, Stanford University Copyright 2002, HP.

[3] User Study Techniques in the Design and Evaluation of a Ubicomp Environment, S. Consolvo, L. Arnstein, B. Franza, in proc of the 4th International conference on Ubiquitous Computing, Sept 2002

[4] Sentient Computing Project Cambridge University Engineering Department Andy Ward, Pete Steggles, Rupert Curwen, Paul Webster http://www.uk.research.att.com/spirit

[5] Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment. Ricardo Morla, Nigel Davies, Lancaster University Pervasive Computing, IEEE, pp48-56, July-Sept 2004

[6] TinyDB: In-Network Query Processing in TinyOS Sam Madden, Joe Hellerstein and Wei Hong http://telegraph.cs.berkeley.edu/tinydb/documentation.htm

[7] Half-Life, Valve Corporation, http://www.valvesoftware.com

[8] Half-Life SDK v2.3, Valve Corporation, http://dev.valve-erc.com

[9] Valve Hammer Editor v 3.4, http://collective/valve-erc.com

[10] Botman's Bots, HL SDK Tutorial, http://www.planethalflife.com/botman

[11] Quake III Arena, id Software, http://idsoftware.com

[12] Unreal Tournament, Epic Games, http://www.epicgames.com

[13] Bylund, M., Espinoza, F., "Testing and Demonstrating Context-Aware Services with Quake III Arena", Communications of the ACM, Jan 2002, Vol 45, No 1, pp46-48

[14] Large Dynamic Range Prediction of AOA, AOD and PDP for MIMO Systems, M. Klepal, P. Pechac, in Proc. IEE 12th International Conference on Antennas & Propagation, Exeter, March 2003, pp. 775-779, ISBN 0-85296-7527

[15] Influence of People Shadowing on Optimal Deployment of WLAN Access Points, M. Klepal, R. Mathur, A. McGibney, D. Pesch, in Proc. of IEEE Vehicular Technology Conference Fall 2004, Los Angeles, CA, USA, September 2004

[16] Influence of People Shadowing on Bit Error Rate of IEEE802.11 2.4GHZ Channel, R. Mathur, M. Klepal, A. McGibney, D. Pesch, in Proc. of 1st IEEE ISWCS, Mauritius, Sept. 2004

[17] TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications, Levis, P., Lee, M., Welsh, M., Culler, D., in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)

[18] Luigi Rizzo "Dummynet: A simple Approach to evaluation of Network Protocols" ACM Computer Comm Rev. vol 27, no. 1, 1997, pp 31-41