A Simulation-based Approach to Highly Iterative Prototyping of Ubiquitous Computing Systems

Eleanor O'Neill, David Lewis, Owen Conlan
Knowledge and Data Engineering Group | KDEG
School of Computer Science and Statistics
Trinity College Dublin, Dublin, Ireland
{Eleanor.ONeill | Dave.Lewis | Owen.Conlan} @cs.tcd.ie

ABSTRACT
Ubiquitous computing (ubicomp), as envisaged by Weiser [22], is heavily user-centric and largely concerned with applications specifically designed to meet end-user needs. Sensor populated ubicomp environments differentiate these applications from existing mobile and distributed systems through context awareness. For the system developer, the problems of heterogeneity and scalability are felt most keenly when designing this adaptive behaviour. A context-aware ubicomp system needs to operate reliably over the wide variety of situations that may be encountered. In this paper we present a technical architecture which has been implemented to support scalable, cost-effective, runtime experimentation using a framework of models to support informed decision making in an iterative design cycle.

Categories and Subject Descriptors
I.6.8 [Simulation and Modelling]: Types of Simulation, Animation and Gaming; C.4 [Performance of Systems]: Design Studies; D.2.m [Miscellaneous]: Rapid Prototyping

General Terms
Design, Experimentation.

Keywords
Ubiquitous computing, adaptive behaviour, context awareness.

1. INTRODUCTION
Ubiquitous computing (ubicomp), as envisaged by Weiser [22], is heavily user-centric and largely concerned with applications specifically designed to meet end-user needs. Sensor populated ubicomp environments differentiate these applications from existing mobile and distributed systems through context awareness. Context awareness means that these systems must respond and adapt to the social, task and environmental context [21], in which they are deployed, if they are to achieve Weiser's application-centric vision.

SIMUTools'09, March 2–6, 2009, Rome, Italy.

For designers of these systems, heterogeneity of sensor data and scalability of environments are major challenges during the design cycle [17]. Specifically when designing this adaptive behaviour. A context-aware ubicomp system needs to operate reliably over the wide variety of situations that may be encountered. This wide variety of situations encountered arises from the combinations of physical settings, environmental and location sensing and, most unpredictably, the behaviour of users in such situations.

These design challenges have resulted in even basic, everyday ubicomp systems, such as simple motion sensor driven lighting systems and smart air-conditioning units failing due to the designer not fully appreciating the specific factors present in a specific situation e.g. the lighting timer is set to too short a cycle for a particular user. These nuances in how users actually carry out their daily activities and how they differ from the way designers and developers expected the system to be used, present challenges that successful applications need to overcome.

Although existing design tools for distributed systems, using traditional software design and evaluation techniques, are a useful basis for design in ubicomp, they cannot address the complexity introduced by context-awareness. A rapid iterative prototyping approach alleviates this problem by speeding up the design cycle and enabling developers to evaluate early system implementations with low investment and fewer repercussions if a system fails. Prototyping approaches have proven a successful choice for many ubicomp design tools [3, 13, 14, 15].

However, to date these toolkits have largely favoured rapid creation and deployment of applications [9] with less emphasis on structured feedback as part of the cyclical design process to support informed decision making during the next design iteration. New factors affecting user acceptance of ubicomp systems raise the need to develop new analytical approaches that will enable investigation of the underlying causal relationships which precede any instance of unwanted behaviour being exhibited to an end user.

Through this research, we have identified the following issues, which can be addressed through simulation-based tool support to help developers and designers as they explore this relatively immature field. There are three core issues:

- Systematic investigation of relevant context and its change over time to determine areas of unwanted adaptive behaviour in the system.

- The capability to reliably configure environmental conditions to repeatedly visit a scenario which is problematic for the ubicomp system under test (SUT).
- Supporting tools to determine causal relationships behind a SUT behaving in an unexpected and unwanted manner.

In this paper, we start in section 2 by discussing related work with a focus on the role of both real-time and non-real time simulation in context-aware system design, the success of rapid-prototyping for ubicomp development and finally issues surrounding requirements engineering for ubicomp. In section 3 we discuss our requirement considerations for this work, followed in section 4 by a brief case study of a real-world problematic ubicomp system which we have observed. Section 5 describes the framework of models we use to support this informed design approach and section 6 describes the technical architecture implemented for runtime experimentation. Finally section 7 discusses our evaluation so far and some future work, followed by section 8 which draws conclusions about this paper.

2. RELATED WORK

Ubiquitous computing has seen many uses of simulation in design tools [2, 3, 4, 16] and has allowed researchers to conduct otherwise costly and time-consuming research more affordably. Ubiwise [2] one of the earliest ubicomp simulators enabled prototyping of hardware and low-level software e.g. protocols. At the time of Ubiwise's development, the ubicomp research community were caught in a situation where application developers were waiting for protocols and hardware to be developed, while hardware developers were waiting to see what kind of applications would need to run on their devices. Ubiwise offered simulation as a way to break this circle of dependence. Ubiwise was developed as a first person interactive simulator because it needed to allow users to interact with device interfaces.

Discrete event simulations have also proved useful for ubicomp systems, specifically smart traffic management systems. Reynolds et al [4] have developed a large-scale 2D grid simulation tool to model sensors, actuators and the environment. For added flexibility, they propose an emulation framework for testing applications and middleware. As an initial test case, they focus on city-wide traffic simulations and model a wide range of ubicomp scenarios. Their 2D discrete simulation approach is very well suited to issues such as traffic light communication however we propose a more user centric approach that is needed when trying to anticipate user acceptance of ubicomp systems.

Rapid prototyping is another technique which has been very successful for ubicomp development. Carter and Mankoff [7] found paper prototypes very useful when trying to identify useful aspects of their system design. This is only really a very early stage design tool as they also uncovered that this technique just doesn't scale and so can only address very narrowly defined questions. A more advanced ubicomp design tool was context widgets [18] which insulated developers from the underlying sensor network in much the same way as graphical user interface tools do. These widgets provide benefits of abstraction and reusability and are very useful when creating new ubicomp applications or retrofitting existing applications to support context-awareness. Widgets are specifically an implementation tool and speed up the development process.

The team behind Topiary have demonstrated a lot of success providing a graphical tool to rapidly prototype both ubicomp scenarios and ubicomp applications. Topiary is specifically designed to look at applications for mobile devices and is capable of generating a deployable prototype from very high level instruction. Prototypes can be run on a 2D desktop simulation where users are moved around in much the same way as a board game. Alternatively prototypes can be deployed on devices in the wild so that users can interact with the system interface on a mobile device. This latter form of testing takes a sort of Wizard of Oz approach since a tester must follow the test-user to play the role of the SUT.

As already mentioned simulation and prototyping have proved to be very useful tools in the design and development of ubicomp systems. We will build on this proven success to enable user-centric testing of adaptive context-aware systems but including model-driven experimentation with prototypes to support feedback on causal relationships in the adaptive behaviour and assist informed decision making in subsequent iterations of the prototyping cycle.

3. REQUIREMENTS

3.1 Requirements Considerations

The ubicomp applications we are targeting with this platform are those that the user experiences in their environment but not specifically on mobile devices. We are not concerned with user interface design but more with evaluating the adaptivity presented to end-users at run time. We are particularly interested in evaluation of ubicomp applications which adapt along the temporal and spatial dimensions. Bandini et al [8] have identified orientation, proximity and containment as the key spatial relations for ubicomp in their Common Sense Model. As part of their future work they list enabling applications to reason along the time axis about changes in the environment as a key concern.

The approach in this document uses the factors in figure 1 to describe the primary factors that influence users' acceptance of these target context aware applications. The factors considered in this diagram are explained as follows:

- Behavioural Envelope: The bounds of potential service behaviour that can be exhibited.
- Context changes: Changes in the task, social or physical context surrounding an ubicomp system.
- Exhibited Behaviour: The behaviour as seen by the end-user.

To find a solution that universally resolves these factors is a study that intersects many fields taking in both technical issues and human factors, many of which are highly subjective and difficult to concretely measure. In addition, no single standardised set of criteria exist to define good behaviour in an adaptive service, although work is underway on this front [1]. To help alleviate this situation, the method presented here is aimed at conducting a thorough investigation of both the adaptive behaviour space and the context space of a SUT, to identify occurrences of unwanted behaviour that may lead to a prototype service being rejected by end users.

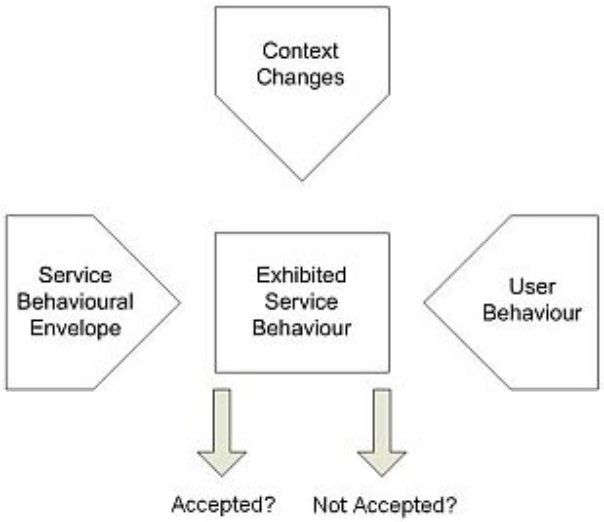

Figure 1. Factors affecting Ubicomp Service end user Acceptance

3.2 Requirements

Our overall objective is to enable identification of the causal relationships leading to unwanted occurrences of adaptive behaviour. Based on this, the factors laid out above, the need for cost effectiveness and both Davidoff's [9] and our own identification of a gap in the research for ubicomp analysis tools, we set out the following requirements for this evaluation tool:

1. An iterative experimental approach must be developed that supports reasoning about users, the system and the environment with a view to identifying unwanted system behaviour.
2. A technical architecture must be implemented to actualise the result from requirement 1.
3. The tool must have access to a cost-effective test environment sufficiently flexible to test many situations and provide a diverse, heterogeneous flow of context information.
4. Tool must enable rapid reconfigurations of the test environment and models to support the iterative prototyping cycle that has proven successful for ubicomp application development.

4. CASE STUDY

Before we go on to discuss our implementation, we present here a short case study of a real-life problematic smart lighting system. We have used this case study to supplement our requirements gathering process for this evaluation tool and will refer to it throughout the remainder of this paper.

4.1 Real World Smart Lighting System

A smart lighting system is designed to automatically switch lights on in public access areas for building occupants. Lights are timed to switch off a set period after a user has been detected. Offices and lab areas remain on a manual light system. Motion sensors switch lights on when they detect movement. A timer is incorporated to switch lights off following a period in which no occupant has been detected. Essentially the system performs the simple action of toggling between on and off states. Figure 2 shows a simple diagram of this system.

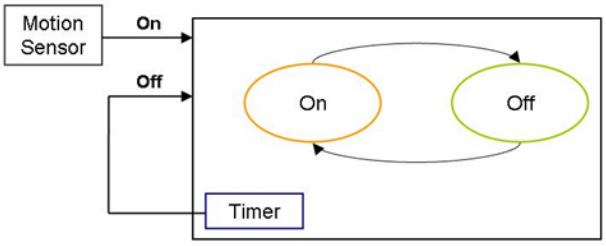

Figure 2. Simple System Diagram

4.2 Run Time Problem

Figure 3 shows the building floor plan where this system was deployed. Stairwells and lift lobbies are the entrance and exit points on each floor. Motion sensors controlling each floor's lighting are installed in each stairwell and lobby. Offices space is marked in grey and is accessible only with a key; white space is publicly accessible.

A problem occurs for late night workers when the number of occupants in the building is very low. When a worker emerges from their room after working for a couple of hours, the timers have switched off the lights and the lights will not turn on again until the worker has reached the exit for that floor. The user must find their way to an exit in the dark but the lights switch on just as the worker exits the hallway, beneficial to neither the user nor the energy saving scheme.

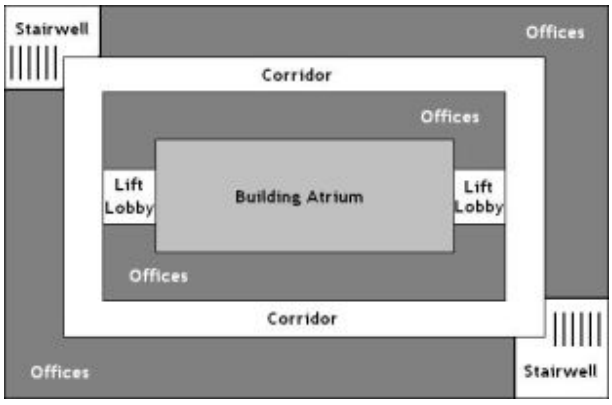

Figure 3. Lighting System Floor Plan

4.3 Problem Discussion

Although Figure 2 clearly depicts how the system operates, it omits two important elements that significantly impact on the operation of the system. The position of motion detectors in the environment heavily impacts on the usefulness of this system and how accurately the system selects lights to switch on.

The framework and modelling approach which we present in the next section attempts to uncover problems such as the one presented in this section by assisting testers to fully explore the behavioural and context spaces for a service using a rapid prototyping and simulation approach.

5. FRAMEWORK MODELS

This section discusses the models that satisfy requirement one in section 3.2. The models listed below will be described in detail through the following subsections before beginning the discussion of the technical architecture.

The following models have been defined for this framework:

- System Logical Requirements Model (SLRM): The set of logical requirements, written as rules, within which the system should operate at runtime
- System State Model (SSTM): Describes all possible states for the system.
- 'Safe' Assumptions Model (SAM): Set of safe assumptions drawn up as guidelines of the maximum-minimum range of boundaries/limits within which the system will be expected to operate. These assumptions will be used to drive experimental cases and full evaluations of the behavioural envelope of a system.
- System Alert Report (SAR): Record of requirements violations generated during an experiment
- Environment Model (ENVM): The environment is modelled using 3D simulation tools, modified to accommodate sensor placement and configuration. The will be discussed in Section 4.5 as part of the technological framework.

5.1 Modelling Requirements as Rules: SLRM

Requirements here are termed as conditions that should be considered non-negotiable and where an application does not meet these, the system is no longer useful to the stakeholders involved. The SLRM defines the requirements that a system must operate within when performing any adaptivity. In the absence of a universal benchmark for good ubicomp design, this approach focuses on determining whether an ubicomp application falls into the trap of unwanted behaviour [12] or whether it can survive everyday use [11].

To conduct this evaluation, requirements modelled as JBoss Rules [5] act as the yardstick against which the ubicomp system's usefulness and effectiveness is evaluated. JBoss Rules works efficiently to minimize the number of conditions that must be evaluated through the use of rule patterns. The benefits of using JBoss Rules for this work are:

- Vocabularies: JBoss Rules are built on top of vocabularies implemented as Java beans. This enables creation of an extensible vocabulary specifically designed to cater for ubicomp applications.
- Separation of Logics/Data: Separation of logic (rules) and data (facts) is well suited to keeping the 3D simulator loosely couple with the rest of the system to support rapid (re)configuration.
- Java OO facts: OO design provides a convenient way to encode context from the environment.
- Speed: JBoss Rules works fast which is essential for large experiments where the flow of context from the environment happens quickly.

Potential pitfalls in using JBoss Rules arise if rules are poorly written resulting in excessive cross products. JBoss is fast to calculate cross product results, however badly designed rules running on large data sets can fail due to insufficient memory. In this approach, good rule design and an optimised environment model will be used to avoid this potential problem. Cross products are an issue for all rule engines and not a problem unique to JBoss Rules.

At runtime when a SUT's behaviour is outside the boundaries of its requirements not all test conditions will succeed and the Rule Engine will generate an Alert in the SAR (discussed in detail in section 5.4). For example in the case of the smart lighting example, a verbose requirement for the system states that 'A user should never be left in the dark'. The rule checks the user's current state; if the user is in the dark then an alert is raised detailing the relevant context. Important context for this event includes the user's location and the user's identification.

5.2 Modelling the System: SSTM

The SSTM models the behavioural states of an ubicomp system, and although not directly used by the technological framework, it was the driving element in recognising the need for the SAM (Safe Assumptions Model), discussed in section 5.3. Figure 2 showed the SSTM for the lighting case study described in section 4. Modelling the lighting system as in Figure 2 does not accommodate the effects of context on this system. In the case of this system, the problem was known in advance of the assessment and so by reverse engineering it was recognised that the post-deployment temporal and spatial issues that would impact on this system were not fully investigated.

Logically the lighting system works well. A user enters a public access area in the building and the lights switch on. Public access areas are marked in white in figure 3. After a defined period of time, the lights switch off again. However since all corridors are not covered by motion sensors, exiting an office will not reactivate the lights.

The SSTM demonstrated the need to supplement the design evaluation with additional information, specifically because design experts are relatively non-existent in this field, there are few people qualified to spot errors such as this one even in this very simple system. To assist evaluation of ubicomp systems and exploration of their behaviours within specified context spaces, a general approach was abstracted in the form of the SAM model to drive experimentation and full evaluations based on safe everyday assumptions.

5.3 Exploring the Behavioural Space with Safe Assumptions: SAM

The manner in which users work/live alongside an ubicomp system and the idiosyncrasies found in their daily activities, impact heavily on the success of an ubicomp system. A modelling approach is required to explore the situation space.

The situation space refers to the many situations that may be forced upon the system in its deployment environment. To design a system that will, not only support the defined use cases, but also operate successfully within the bounds of non-task related user attributes, designers need extra guidance to assist them in performing a thorough exploration of the design space.

To this end, the 'safe' assumption model (SAM) provides a list of safe assumptions (axioms) about conditions in which the SUT can be expected to operate within. The purpose of the assumptions is to provide guidance to experimental designers about the minimum and maximum limits, within which the system should be tested. Assumptions take the place of behavioural patterns for a specific environment and users, when these behavioural patterns are unknown. This approach is taken because information about behavioural patterns and user idiosyncrasies is not always available to systems designers e.g. in the case of a previously unoccupied building. Assumptions should be tested up to and including the limits they set out for the system, the following illustrates a worked example.

5.3.1 SAM Example

Background: Case Study from Section 4

Scenario 1: OfficeWorkerA arrives to work, enters their office building. OfficeWorkerA must cross three public access areas to walk to their office.

Safe Assumption 1: OfficeWorkerA will spend up to, but no more than, 4 hours continuously at their desk before they will need a refreshment break. Experimental Factor: Experiments should be run to investigate regular time intervals from ~1mins to 4 hours.

Safe Assumption 2: Offices have peak and off-peak times. A building occupant working late may find themselves working alone on their floor. Experimental Factor: Experiments should be run for low and high building occupancy.

Experimental Design: Based on these assumptions, experiments should be run to investigate how the system behaves at regular time intervals from ~1min to 4 hours and also to investigate behaviours at low and high building occupancy.

5.4 System Alert Report: SAR

The aim of this framework is to produce a report of alerts raised during an experiment for post experimental analysis. Alerts are generated at runtime during an experiment when system deployment/behaviour is not inline with the system requirements set out for the SUT.

Requirements are tested continually during an experiment. The evaluation of these requirements will for the most part be affected by user activity inside the smart environment and thus must be evaluated for all potential user behaviour. An example of a live requirement for a smart lighting system might state that a 'user must not be left in the dark'. This would have to be evaluated throughout the experiment as the user changes location to ensure that the system's design meshes well with the building's sensor configuration.

The resulting alerts are compiled into a report which will include information to identify the specific requirement that was violated, the context surrounding the violation and the user involved. An example of a generated alert report for the smart lighting system design is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<system-alerts>
 <alert id="1">
  <modified time="1221255106656" />
  <rule id="1" />
  <assumption id="1" />
  <location>Corridor1_Floor1</location>
  <user>Eleanor</user>
  <verbose-explanation>
        Lights out for user: Eleanor
 </verbose-explanation>
 </alert>
```

Figure 4. Excerpt from Sample SAR Report: Reporting an instance of unwanted behaviour

6. TECHNICAL ARCHITECTURE

This section discusses the technical architecture which has been built to support runtime experimentation using the model framework from section 5. The implemented framework consists of a simulated experimental test environment, database of models, a JBoss Rules based adaptive engine and finally a proxy acting as the integrating link for the platform. Each of these components will be discussed in the following sections after first discussing the process model used for the framework and the configuration tools used during the experimental set up procedure.

6.1 Process Model

The overall aim of this experimental platform is to produce a report of alerts raised during an experiment for post experimental analysis. Alerts are generated at runtime during an experiment when exhibited ubicomp SUT behaviour is not inline with the requirements set out for the SUT.

The process model which this platform has been designed to support is an iterative cycle, illustrated in Figure 5. An experimental cycle begins with setup, to create and configure the virtual test environment. The virtual environment used in the platform is a modified version of the Half-Life 2 games engine [10] which is supplied with its own SDK. We have modified the map editor to allow developers position and configure sensors in the virtual world.

The execution phase of an experimental cycle allows for either multi-player or single user, bot populated experiments. Multiplayer simulations allow up to 32 users to experiment with the SUT simultaneously in the context of the virtual world. Bot driven simulations on the other hand involve a single user testing the service while role playing bots also roam the virtual world testing defined scenarios. We also intend to use role playing bots

to conduct large scale experiments at higher speeds for rapid scenario testing where a user is not required.

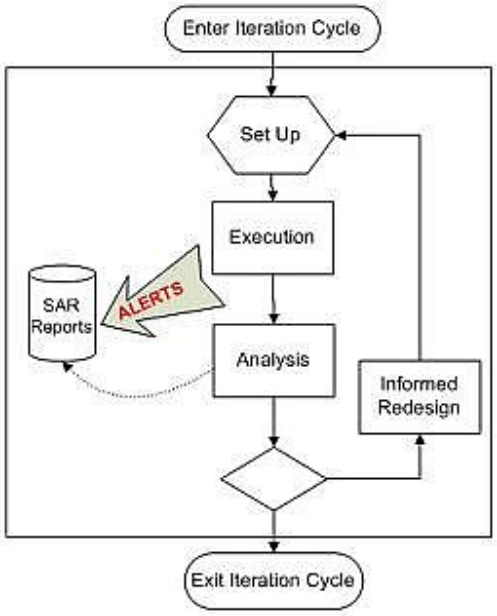

Figure 5. Process Model

During either user driven or bot driven executions, alerts are generated to create the SAR file stored in an eXist database. This data is generated for post experimental analysis and to lead informed decision making in the next design cycle. Analysis of this data is not something we will address in this paper, we will focus largely on the experimental process involved in generating SAR reports.

6.2 Experimental Configuration Tool

The virtual sensors featured in this platform have been added to an existing Half-Life SDK tool called Hammer, see figure 6. Hammer is used to construct maps for the game engine. Our modifications allow a designer to configure the simulated world using a range of sensors. The availability of Hammer as part of the SDK enables rapid reconfigurations and diverse sensor types since we can add any of the sensors to this tool which we develop for the virtual environment.

In more general terms for gaming purposes Hammer is also used to:

- Construct the physical space i.e. walls and doors.
- Add bots and bot trajectories

Although developing a large map takes some effort, considerable productivity can be achieved by using a blank version of an existing environment to outline an experiment. The effort to populate blank maps with sensors is minimal by comparison to developing a map of a new environment from scratch. The experimental design and set-up process makes use of reusable resources in keeping with the iterative and incremental approach required by rapid development, testing and experimentation. Among these reusable resources are the map files that define the experimental environment, the sensors and the experiment definition XML profiles for a service.

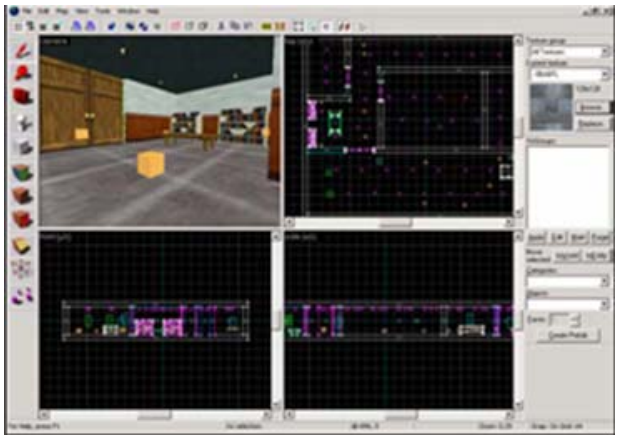

Figure 6. Hammer Configuration Tool

6.3 3D Interactive Simulation Environment

The cost and effort involved in creating smart environments are still prohibitively expensive for large scale or frequently reconfigured testing. A virtual model of the deployment environment provides the flexibility to overcome this and carry out experiments in many settings. For this reason, a simulation environment is used in the place of a live environment, figures 7 and 8.

The additional benefit of the simulation environment over a lab-based setting is the ease with which the environment can be reconfigured, both in terms of the sensor deployment and configuration, and the physical layout and construction of the building. The simulation environment called Pudecas [6] builds directly upon an earlier smart simulation prototype called Tatus [16]. The functionality from Tatus, originally programmed into the Half-Life engine, has been moved and upgraded to the Half-Life 2 engine improving the quality of the simulations, see Figure 7 and 8.

The simulator presented here allows:

- Generation of simulated sensor data at runtime. Primarily these are location sensors but we also use pressure mats and work has been done modelling Ubisense and ZigBee outside of the games engine [19].
- Actuations of entities in the virtual world e.g. lights, automated doors. These actuations happen when signalled by the SUT.
- Virtual sensors are activated when players move around the virtual world, in the same manner as would happen in the real world. This provides the stream of context required to drive the SUT. To improve the fidelity of the platform, work has been done by McGlinn [19] to produce more realistic and reasonable context, for example so that location information is not supplied to the SUT with absolute accuracy that is inherent with the grid based positioning system of a games engine.

Figure 7. Real Building (left); Simulated Building (right)

Simulated sensors have been modelled to be visible or invisible. We use visible simulated sensors to represent physical devices e.g. pressure mats or wireless access points. Invisible simulated sensors are used to model the field of view or signal range of these devices where required. The sensors are programmed to be event-driven, polling or a combination of the two. For instance, a pressure mat responds to the event of a user stepping on it, where as a Bluetooth master polls to detect new slaves. Using a game engine allows flexibility in the type and quantity of sensors featured by the test environment. For the most part, this is not yet realisable in the real-world where the expense and logistics are prohibitive.

6.4 Proxy

Interfacing the SUT to the simulator is done via a Java application or Proxy. The platform can host and manage the connections between multiple services and multiple test environments simultaneously. This allows multiple services to access a single environment, or vice versa, a single service to access multiple environments. Services are not obliged to subscribe to all simulated environments and only receive information about relevant experiments.

A new experiment commences when a service contacts the simulator with an experiment configuration file. This configuration file contains an experiment ID, a map name, a game-server address and data subscription information. The service is registered and the simulator creates a new database [8] collection using sensor information parsed from the map file. The simulator invokes a new game-server on the remote host and subsequently establishes a connection with the simulation for experimental data transfer.

At run-time, messages flow between the virtual environment and the adaptive service. Data leaving the simulator becomes the contextual information on which services base their decisions and thus respond to the user's needs. In response, services send asynchronous instructions to alter the state of the environment through device or entity actuation, e.g. opening a door or switching on a light. Only a single connection to game-server hosting the experiment is required since underlying game infrastructure ensures game-clients are also updated in a time that is imperceptible to the player/developer. Ultimately, the sensors will send their information to the services under test via a contextual services layer.

Figure 8. Office space in the simulated environment

6.5 Adaptive Engine

In this work we are using an adaptive engine originally designed for eLearning purposes [20]. In the place of the original eLearning models we have developed a set of models specifically tailored towards ubicomp design analysis. The major benefit for us in using this adaptive engine has been the separation of concerns between the engine and the models. Model semantics are not embedded in the adaptive engine meaning development and integration of new models is relatively quick and easy.

The adaptive engine is designed to reconcile the input models, SLRM, SAM and the flow of simulated context, resulting in the output model, the SAR file. The biggest challenge in the integration work was accommodating the significantly dynamic nature of this type of experimentation. Context changes are much more rapid than in an eLearning course where adaptation is user invoked as the user achieves learning goals. However in this experimental platform, context is pushed on the adaptive engine by activity in the simulated environment. An update service was developed to handle this extra functionality for the adaptive engine. Figure 9 shows an overview of the integrated system.

In addition to the cost benefit of the simulated environment, another major benefit is access to environmental information. This has been key to the success of this approach. In order to analyse the lighting system from the case study in section 4, we needed to determine when the lights went on or off. In the real-world we would have need observers or light sensors for this task. However in the virtual world we can set up a system of notifications which ensure that the adaptive engine is updated with information when a light's status changes. Having this information, while also being 100% certain of users' location at all times, we can easily determine if any user is in darkness.

When we are generating context, the high accuracy of the games engine is not ideal and so work continues to improve the fidelity of this [19, 16]. However, when it comes to analysis, this level of accuracy allows us to compare the services ability to meet user needs with the user's actual current situation.

6.6 Performance

For the lighting case study, the adaptive engine is driven by changes in a user's location and changes in a light's status. The engine runs each new contextual situation through the rules base. So far we have found JBoss Rules is capable of supporting our

analysis however we intend to monitor this for larger rule-bases and larger context spaces.

Our virtual three storey office building features 104 rooms, comprised of offices, computer labs and lecture rooms and is furnished with 520 desks, 352 chairs and 257 replica desktop computers. Optimised mapping techniques continue to allow experimental environments to grow in size and complexity. The latency of messages, between game-server or game-client, and a service, is of the order of milliseconds when crossing up to four LAN connections. The platform architecture has proved to be a viable solution. Services receive data in a timely manner while users do not suffer perceivable or adverse delays in service response times.

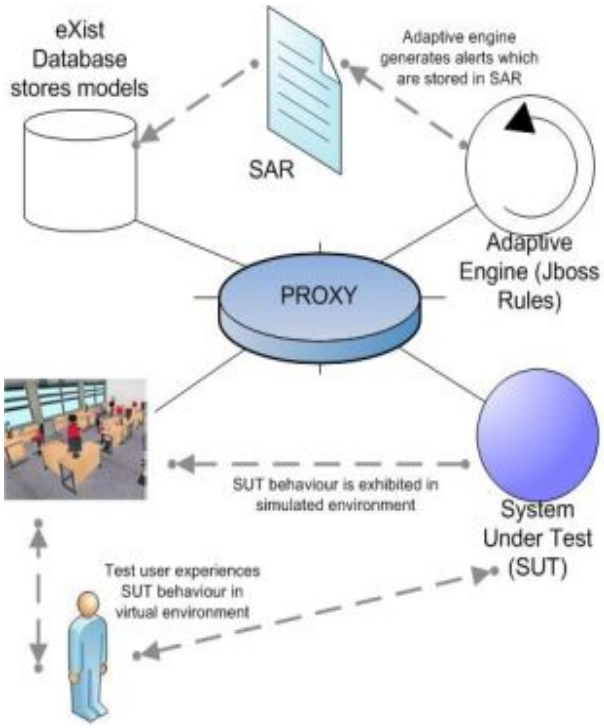

Figure 9. System Overview

6.7 Summary of Integrated Framework

This section presented the technological framework which has been implemented to support runtime experimentation using the modelling approach from section 5. A first person, interactive, 3D simulator provides affordable, cost-effective easily configurable test environments. An adaptive engine has been integrated to reason about behaviour exhibited by the SUT. Finally a proxy has been developed to integrate the system components and the eXist database.

7. EVALUATION

Evaluation of the work presented here is being completed in two steps:

- Evaluation of the framework against the requirements from section 3.2
- Evaluation of the framework when used to investigate known problems in the lighting case study (section 4).

7.1 Evaluation against requirements

This section discusses how we satisfied the requirements from section 3.2 in our design.

Requirement 1: An iterative experimental approach must be developed that supports reasoning about users, the system and the environment with a view to identifying unwanted system behaviour.

Design Solution 1: Using our multi-model approach we can reason about:

- Users in terms of their activity in the virtual environment and the resulting context which they force on the SUT.
- The system in terms of its ability to meet its requirements (SLRM) under a wide range of context situations (SAM).
- The environment in terms of the spatial relationships that exist in the 3D simulation, specifically containment (i.e. room granularity), proximity and orientation.

Requirement 2: A technical architecture must be implemented to actualise the result from requirement 1.

Design Solution 2: We have modified a state of the art computer game to generate simulated context at runtime based on player activity. A JBoss Rules powered adaptive engine reasons about the behaviour exhibited by the system in relation to the actual context of the user. The combination of a games engine and rules engine enables extensive testing in large scale environments. We currently test in a simulated version of a real-life 3 storey office block. A second virtual office block is currently under development to recreate an office block which is affiliated with another university.

Requirement 3: The tool must have access to a cost-effective test environment sufficiently flexible to test many situations and provide a diverse, heterogeneous flow of context information.

Design Solution 3: The modification of a games engine and its SDK reduces the cost significantly compared to field studies.

Requirement 4: Tool must enable rapid reconfigurations of the test environment and models to support the iterative prototyping cycle.

Design Solution 4: The modifications to the Hammer map editor allow rapid placement of sensors, configuration of bots and bot activity and construction of the physical space. Our experience with our on site Ubisense installation shows that Hammer is both more convenient and exceedingly faster than trying to work with a real world installation. We have also seen continued success over the past three years, with large numbers of undergraduate, masters and PhD students choosing to work with the virtual ubicomp environment rather than integrate their systems with a real sensor network for testing due to the time and effort involved. Some of the applications these students have tested included location based games, location based instant messaging and policy based managed systems.

7.2 Case Study Evaluation

Evaluation of this approach will be conducted in three parts. The first focussed on evaluating the tool against the requirements set out for it. The second two parts will evaluate the tool in terms of:

- Efficiency:
    1. Determines the effort involved in setting up and configuring an experimental iteration i.e. efficiency in terms of human effort.
    2. Determines the effort required to develop suitable models for uncover a system's flaws
- Reliability: Determines how often the framework falsely reports unwanted behaviour by using sanity checking on results.
- Repeatability: Determines the ability to recreate situations which generated alerts for closer inspection.

7.2.1 Bot Driven Simulation

Bot-driven simulations are used to rapidly perform a massive exploration of the temporal and spatial design space of a system. This experiment investigates the reliability, efficiency and repeatibility with which role-playing bots can uncover the unwanted behaviour exhibited in the ubicomp lighting system. Role-playing bots will be configured to explore the behavioural envelope of the system guided by the set of assumptions about how building occupants will use that space.

- Efficiency will be evaluated in terms of the effort involved in creating a bot-driven experiment; time to take to create suitable bot roles, time taken to set up the environment.
- Reliability will be evaluated by using sanity checking on the SAR files generated during bot-driven experimentation.
- Repeatability will be evaluated by taking alert instances from the SAR reports and recreating them for a human, first person test.

7.2.2 Design Team Led Experiments

This experiment involves pairs of ubicomp researchers working as test users of the design tool. The objective of the experiment is to determine that using this approach a developer/designer can identify the problem behaviour of the smart lighting application and the root cause of the unwanted behaviour. Developers will be supplied with instructions for the design tool, the implemented lighting system and a configured simulation environment. To complete the experiment, the test users will need to derive:

- The set of JBoss Rules required to generate the SAR report.
- The set of assumptions that drives thorough scenario investigation.

8. CONCLUSIONS

This research has developed an approach for conducting thorough investigations of both the adaptive behaviour space and the context space of a service, to identify occurrences of unwanted behaviour that may lead to a prototype service being rejected by end users. The approach features a set of models which define an experiment in terms of users, the environment, the SUT and the requirements which that system must satisfy throughout the context space in which it operates.

To test the service at runtime, a technical architecture has been developed linking the test environment, an adaptive engine, the SUT and the definition of expected behaviour for the SUT. The platform has the capability to capture a full snapshot of context surrounding instances of unwanted behaviour. This information is recorded in the SAR report for analysis and informed decision making in subsequent design iterations.

We have designed this platform specifically to address design issues for adaptive systems that largely exhibit their behaviour in the user's environment rather than on handheld devices. We have not been concerned with user interface design in this research but are more interested in the identification of situations which are problematic for a SUT. Through our work in helping system designers to identify unwanted behaviour in their systems, we hope to support the analysis phase between iterative cycles. We expect that for more complex systems it may not be possible to fully resolve all instances of unwanted behaviour however we think that the comprehensive information that we can provide using this framework will be a useful tool in assisting negotiations within design teams. We also consider that the ability to replay scenarios exactly and determine causal relationships in these complex systems, are useful tools for designers of ubicomp systems.

9. ACKNOWLEDGMENTS

This work is supported by Enterprise Ireland under the PUDECAS project [TD 2005 217-A/B].

10. REFERENCES

[1] Jean Scholtz, Sunny Consolvo, "Toward a Framework for Evaluating Ubiquitous Computing Applications," IEEE Pervasive Computing, vol. 3, no. 2, pp. 82-88, Apr-Jun, 2004

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

[3] Li, Y., Hong, J. I., and Landay, J. A. 2004. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM Symposium on User interface Software and Technology* (Santa Fe, NM, USA, October 24 - 27, 2004). UIST '04. ACM, New York, NY, 217-226. DOI= http://doi.acm.org/10.1145/1029632.1029671

[4] Reynolds, V., Cahill, V., and Senart, A. 2006. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the First international Conference on integrated internet Ad Hoc and Sensor Networks* (Nice, France, May 30 - 31, 2006). InterSense '06, vol. 138. ACM, New York, NY, 1. DOI= http://doi.acm.org/10.1145/1142680.1142682

[5] JBoss Rules

http://www.jboss.com/products/rules

[6] Eleanor O'Neill, David Lewis, Kris McGlinn, Simon Dobson: Rapid User-Centred Evaluation for Context-Aware Systems. DSV-IS 2006: 220-233

[7] Carter, S. and Mankoff, J. 2005. Prototypes in the Wild: Lessons from Three Ubicomp Systems. *IEEE Pervasive Computing* 4, 4 (Oct. 2005), 51-57. DOI= http://dx.doi.org/10.1109/MPRV.2005.84

[8] Stefania Bandini, Alessandro Mosca, Matteo Palmonari, 2005. A Hybrid Logic for Commonsense Spatial Reasoning. AI*IA 2005: Advances in Artificial Intelligence, Volume 3673/2005, pp 25-37

[9] S. Davidoff, S. Carter and J. Mankoff. Can Early-Stage Tools and Techniques for Iterative Design Help Researchers Understand a Problem Space? Pervasive 2005 UbiApp Workshop, Munich, Germany, May 2005.

[10] Half-life 2 / Hammer

http://www.half-life2.com/
http://developer.valvesoftware.com/wiki/Main_Page

[11] Abowd, G. D. 1999. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international Conference on Software Engineering* (Los Angeles, California, United States, May 16 - 22, 1999). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 75-84.

[12] Sitou, W. and Spanfelner, B. 2007. Towards Requirements Engineering for Context Adaptive Systems. In *Proceedings of the 31st Annual international Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007) - Volume 02* (July 24 - 27, 2007). COMPSAC. IEEE Computer Society, Washington, DC, 593-600. DOI= http://dx.doi.org/10.1109/COMPSAC.2007.223

[13] Balasubramanian, M., Chaturvedi, N., Chowdhury, A. D., and Ganesh, A. 2006. A framework for rapid-prototyping of context based ubiquitous computing applications. In *Proceedings of the IEEE international Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (Sutc'06) - Volume 01* (June 05 - 07, 2006). SUTC. IEEE Computer Society, Washington, DC, 306-311. DOI= http://dx.doi.org/10.1109/SUTC.2006.7

[14] Liogkas, N., MacIntyre, B., Mynatt, E. D., Smaragdakis, Y., Tilevich, E., and Voida, S. 2004. Automatic Partitioning: Prototyping Ubiquitous-Computing Applications. *IEEE Pervasive Computing* 3, 3 (Jul. 2004), 40-47. DOI= http://dx.doi.org/10.1109/MPRV.2004.1321027

[15] Bannach, D., Amft, O., and Lukowicz, P. 2008. Rapid Prototyping of Activity Recognition Applications. *IEEE Pervasive Computing* 7, 2 (Apr. 2008), 22-31. DOI= http://dx.doi.org/10.1109/MPRV.2008.36

[16] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, Dirk Pesch, "A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments," tridentcom,pp.60-69, First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05), 2005

[17] Pham, H. N., Mahmoud, Q. H., Ferworn, A., and Sadeghian, A. 2007. Applying Model-Driven Development to Pervasive System Engineering. In *Proceedings of the 29th international Conference on Software Engineering Workshops* (May 20 - 26, 2007). ICSEW. IEEE Computer Society, Washington, DC, 193. DOI= http://dx.doi.org/10.1109/ICSEW.2007.43

[18] Salber, D., Dey, A. K., and Abowd, G. D. 1999. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: the CHI Is the Limit* (Pittsburgh, Pennsylvania, United States, May 15 - 20, 1999). CHI '99. ACM, New York, NY, 434-441. DOI= http://doi.acm.org/10.1145/302979.303126

[19] Kris McGlinn, Eleanor O'Neill, David Lewis, 2007. Modelling of Context and Context-Aware Services for Simulator Based Evaluation. MUCS 2007, 4th International Workshop on Managing Ubiquitous Communications and Services (part of IM 2007)

[20] Conlan, O., Wade, V., Bruen, C., and Gargan, M. 2002. Multi-model, Metadata Driven Approach to Adaptive Hypermedia Services for Personalized eLearning. In *Proceedings of the Second international Conference on Adaptive Hypermedia and Adaptive Web-Based Systems* (May 29 - 31, 2002). P. D. Bra, P. Brusilovsky, and R. Conejo, Eds. Lecture Notes In Computer Science, vol. 2347. Springer-Verlag, London, 100-111.

[21] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing* (Karlsruhe, Germany, September 27 - 29, 1999). H. Gellersen, Ed. Lecture Notes In Computer Science, vol. 1707. Springer-Verlag, London, 304-307.

[22] Weiser, M. 1995. The computer for the 21st century. In *Human-Computer interaction: Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. Buxton, and S. Greenberg, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 933-940.