Situation-Based Testing for Ubiquitous Computing Systems

A thesis submitted to the

University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy

Eleanor O'Neill

Knowledge and Data Engineering Group,

School of Computer Science & Statistics,

Trinity College Dublin,

Ireland.

2011

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

Eleanor O'Neill

Dated: May 3, 2011

Permission to Lend and/or Copy

Acknowledgements

Eleanor O'Neill

University of Dublin, Trinity College

2011

Abstract

A challenge for designing ubiquitous computing (ubicomp) systems, particularly for indoor, sensor-rich environments, is the complexity of causal relationships between contextual inputs, exhibited system behaviour and overall appropriateness of the resulting outcome. Situations in a ubiquitous computing environment are the summative effect of user activity, exhibited system behaviour and physical environment factors. Two key challenges faced by designers of ubicomp systems lie in the difficulty monitoring the deployment environment for inappropriate situations and subsequently tracing the causal factors. The spatio-temporal relationships that exist between mobile users and distributed embedded inputs and outputs can make it difficult for the designer to anticipate both when and where relevant situations will occur. Additionally, ubiquitous computing systems must operate across the physical-digital divide, which leads to the difficulty tracing both the physical and digital causal factors leading to specific inappropriate situations.

This thesis presents a novel situation-based testing approach for assessing the appropriateness of exhibited ubicomp system behaviour in a simulated ubicomp deployment. Situation specification enables the designer to define tests which are used to analyse situational outcomes in a simulated ubicomp deployment environment with the aim of detecting inappropriate situations. Structured feedback supports traceability to investigate the physical and digital causal factors leading to specific situations. This situation-based testing approach is realised in the InSitu Model Framework, InSitu Test Process and the simulation-based InSitu Toolset. The evaluation of the situation-based testing approach presents case studies that investigate correctness, extent and expressiveness of InSitu. This is supplemented with a user trial to provide an objective assessment of both the expressiveness of InSitu and suitability of the situation-based testing approach. Finally, a comparison framework is used as an instrument to compare InSitu to the tools reviewed in the state of the art survey.

Contents

List of Tables

List of Figures

Glossary

Avatar A computer user's representation in a virtual environment, often a character of humanoid form.

Bot A character in a virtual environment controlled by artificial intelligence.

Context-aware computing Refers to systems that use context or situational information to provide relevant information and/or services to the end-user, where relevancy depends on the user's task or situation.

Context-based action system Systems that invoke actions and perform actuations on the physical deployment environment, in order to adjust the state of the physical environment. These systems use a combination of decision logic in the ubicomp system software, and actuators and device controllers embedded in the deployment environment.

Design Considerations Refer to entities or aspects of the environment that impact the state of the environment, but that cannot or should not be forced to change in order to resolve problematic behaviour, i.e. the physical structure of the building and the natural activity or work patterns of users.

Environment state Information which describes conditions that exist in an environment.

Inappropriate situation A situation that arises due to the effects of unwanted behaviour in a ubicomp system. This is not a technical problem of the ubicomp system, but instead arises due to a specific combination of user activity and system activity, in a particular environment, social and task context. (Derived from Fahrmair et al.'s definition of unwanted behaviour [39]).

Mobile computing Refers to mobile devices that use the wireless network to connect to resources and services, and that are portable.

Pervasive computing Refers to physical environments richly embedded with networked sensors, actuators and computing devices to support distribution of information and services.

Physical-digital boundary The interface at which ubicomp system software and ubicomp devices meet the physical environment.

Sensed context Sensed information about the conditions or circumstances relevant to an event or situation.

Situation specification A situation specification consists of one or more assertions about context that are conjoined using logical connectives, and existential and universal quantifiers. Assertions may comprise of further domain-specific attributes, given that the required semantics are available. (Builds on Clear et al.'s definition [28, 27] and draws from the work by Henricksen and Indulska [50]).

Traceability Attributing events and actions to that which caused them. (Generalisation of Weber et al.'s definition [116]).

Ubiquitous computing Refers to environments that feature mobile computing, pervasive computing and context-aware computing, in order to support systems that react or adapt in response to changing conditions and situations in the environment, with the over-arching aim of meeting end-user needs.

Unwanted behaviour The phenomenon where the behaviour of a given system, while free of errors, still differs from the expectations of its current user. (Defined by Fahrmair et al. [39]).

Author Publications

O'Neill, E., Conlan, O. and Lewis, D., *'Modeling and Simulation to Assist Context Aware System Design'*, Simulation: Transactions of the Society of Modeling and Simulation International, vol. 87(1-2), pp. 149 - 170, 2011.

McGlinn, K., O'Neill, E., Gibney, A., O'Sullivan, D., Lewis, D., *'SimCon: A Tool to Support Rapid Evaluation of Smart Building Application Design using Context Simulation and Virtual Reality'*, Journal of Universal Computer Science, J. UCS, vol. 16(15), 1992-2018, 2010.

McGlinn, K., Corry, E., O'Neill, E., Keane, M., Lewis, D., O'Sullivan, D., *'Monitoring Smart Building Performance Using Simulation and Visualisation'*, In Proceedings of Ubiquitous Computing for Sustainable Energy (UCSE 2010), Ubicomp 2010 Workshop, Copenhagen, Sept 25th, 2010.

O'Neill, E., Lewis, D., Conlan, O., *'A Simulation-Based Approach to Highly Iterative Prototyping of Ubiquitous Computing Systems'*, In Proceedings of 2nd International Conference on Simulation Tools and Techniques, Rome, Italy, 2-6th March 2009.

O'Neill, E., McGlinn, K., Lewis, D., Bailey, E., Dobson, S., and McCartney, K., *'Application Development using Modeling and Dynamical Systems Analysis'*, In Proceedings of 1st International Workshop on Context-Aware Middleware and Services (CAMS '09), Dublin, Ireland, 16th June 2009, ACM, 2009.

McGlinn, K., O'Neill, E., Lewis, D., *'SimCon: A tool for modeling context sources for rapid evaluation of pervasive applications using virtual reality'*, In Proceedings of 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, Dublin, Ireland, 2008.

O'Neill, E., McGlinn, K., and Lewis, D., *'A Platform to Evaluate Usability in Adaptive Context-Aware Services through Adaptive Questioning'*, In Adjunct Ubicomp '07 Proceedings, USE '07, 1st International Workshop on Ubiquitous Systems Evaluation, Innsbruck, Austria, 16th Sept 2007.

McGlinn, K., O'Neill, E., Lewis, D., *'Modelling of Context and Context-Aware Services for Simulator Based Evaluation'*, MUCS 2007, In Proceedings of 4th International Workshop on Managing Ubiquitous Communications and Services, part of IM 2007, Munich, 2007.

O'Neill, E., Lewis, D., McGlinn, K., Dobson, S., *'Rapid User-Centred Evaluation for Context-Aware Systems'*, XIII International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 2006), Dublin, Ireland, July 2006, Springer LNCS 4323, pp220 - 233, 2006.

O'Neill, E., Klepal, M., Lewis, D., O'Donnell, T., O'Sullivan, D., Pesch, D., *'A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments'*, In Proceedings of First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM '05), pp.60-69, 2005. *(Citations: 42)*

O'Neill, E., Lewis, D., *'A Platform for User-Centred Evaluation of Context-Aware Services'* (Poster), In Proceedings of 5th Information and Telecommunications Technologies Conference, Cork, Ireland, 2005, pp219-221.

Chapter 1

Introduction

1.1 Motivation

A new design challenge introduced by ubiquitous computing (ubicomp) is the loose coupling, but also dynamic dependency, in the relationship between users, the physical environment and a software system, as noted by Tang et al. [111]. This challenge stems from the combination of decision logic in the software implementation and embedded devices in the physical environment, as discussed by Cook and Das [31]. Smart buildings are a subset of ubiquitous computing environments which are directly affected by this issue. Embedded sensor technology enables ubicomp systems to meet user needs without requiring that users explicitly provide input or instruction. This creates the invisible interface, often referred to in the literature [118, 96, 31], which facilitates user interaction that is a by-product of natural human activity or normal user work patterns. Embedded actuator technology enables distributed ubicomp system outputs, which change the state of the physical environment, and which support the computer to enter the human's world [119, 76]. This distributed, invisible interface allows ubicomp systems to meet not only the technological, but also the physical environment needs of end-users.

However, invisible interfaces only persist while technology performs effectively and appropriately. Technology which disrupts an end-user's activity or normal behavioural pattern immediately draws the user's attention back to the system. Human mental models of their surroundings tend to be partial, non-technical and for these reasons often inaccurate [47]. Users perceive the state of the world, but not the underlying causal or physical factors that lead to changes in that state. This means that users do not perceive the sensing, information processing, decisions or actuation instructions undertaken by ubicomp systems, although expert users may have some technical understanding of these processes. When a user does not have a mental model of an application interface, i.e. due to invisibility, resolution of problematic behaviour increases in difficulty for the user. At best the user may be forced to deviate from their normal work pattern or natural behaviour, at worst the user will have to operate in parallel with a problematic situation. The occurrence of either breaks the ubicomp ideal of systems designed to meet end-user needs, as put forward by Weiser [119]. This motivates a strong need for ubiquitous computing systems that are well-designed so they exhibit behaviour that is appropriate to the user's current circumstances.

Compared to the desktop computing paradigm, the spontaneous nature of human activity has a greater range to impact the effectiveness of ubicomp system behaviour. Banavar and Bernstein [9] note that ubicomp systems must operate in harmony with the dynamism of human-centric environments. Users may spontaneously or serendipitously adjust their behaviour in response to changing circumstances in their environment. This can make it difficult for a designer to anticipate the full spectrum of behaviour that a deployed ubicomp system will encounter. For this reason, it is not sufficient for ubicomp applications to be technically correct, they must also be appropriate to their contextual setting. Fahrmair et al. [39] identify this as a new form of failure, specific to ubicomp environments, which they refer to as *unwanted behaviour*. According to this definition, unwanted behaviour is not a technical error, but it is a problem perceivable by the end-user and that arises due to a certain combination of user, situation and environment. It is termed a failure because under these conditions the system does not meet the needs of the end-user. The result of unwanted behaviour is an *inappropriate situation* arising for the end-user in the physical environment.

A major challenge faced by designers testing ubicomp systems, in order to evaluate the appropriateness of system behaviour, lies in the difficulty monitoring the effect of ubicomp system behaviour because it is distributed in the physical environment. The spatio-temporal relationships that exist between distributed inputs, outputs and user activity raise a challenge for the designer in terms of anticipating both *when* and *where* problematic situations will occur. As ubicomp deployments scale in terms of physical space, as well as the number of users and entities, testing correspondingly increases in complexity and therefore in difficulty. Additionally, systems must operate across the physical-digital divide that exists between the physical environment and the computing environment. This raises a second challenge due to the difficulty tracing causal factors leading to a specific outcome because the causes can be both physical and digital, and often are only loosely coupled to the final outcome. Traceability for this purpose is defined as *the process of attributing events and actions to that which caused them*, a generalisation of Weber et al.'s definition [116].

Two approaches that show promise for designing and testing ubicomp systems are iterative-prototyping and simulation. Iterative prototyping approaches show promise for designing and developing ubicomp systems [111] as well as being generally recognised for their strength in designing complex systems [67]. However, to date prototyping tools for ubicomp systems have largely focused on rapid creation and deployment of applications [94, 69, 6, 70, 12], with less emphasis on structured feedback to inform the next iterative cycle. Simulation has been shown to be beneficial as a testing tool for ubicomp systems, e.g. UbiWise [14], DiaSim [59] and UbiREAL [80]. Simulation can address issues for ubicomp system testers such as provision of heterogeneous context sources, scalability of test environments, rapid deployment configuration and lower cost testing. Additionally, Chalmers suggests that approaches which use models of human behaviour have the potential to support more thorough evaluation of ubicomp systems, particularly to cope with the unexpected and unanticipated outcomes that can arise in a ubicomp deployment

[107].

In summary, these issues motivate the need for a testing approach that can assess the summative effect of system and user behaviour in a ubicomp environment in order to assess the appropriateness of situations that arise. This thesis therefore proposes a situation-based testing approach which analyses a simulation of a ubiquitous computing deployment to identify instances of specified inappropriate situations. It is intended that this will provide structured feedback that can facilitate an assessment of the overall appropriateness of system behaviour, which in turn can inform design refinement within an iterative prototyping cycle. Structured feedback for this purpose is defined as test results that are automatically annotated or marked up in such a way that a designer can conduct an analysis to both assess situational appropriateness and subsequently, where necessary, trace the relevant causal factors. Structured feedback is intended to support traceability of causal factors in order to draw links between distributed inputs, user activity, and resulting situations, and thereby inform resolution of inappropriate situations in subsequent design iterations.

1.2 Research Question

Can situations be identified in a simulated ubicomp deployment, independent of the causal factors, so that structured, traceable feedback can be provided that supports assessment of the appropriateness of ubicomp system behaviour?

1.2.1 Thesis Objectives

The main objective of this thesis is to design, implement and evaluate a situation-based testing approach for ubicomp systems, which can be used in support of the evaluation phase of an iterative prototyping cycle, particularly for indoor environments.

In order to address the research question of the thesis, the following objectives were defined:

O1 To research and survey the state of the art in prototyping, testing and evaluation tools for indoor, location-tracking, ubicomp systems.

O2 To provide a formal definition of the situation-based testing approach by devising a test specification framework.

O3 To define a systematic test process, which includes identification of inappropriate situations and which supports a feedback loop to assist informed design refinement as part of an iterative testing approach.

O4 To realise the framework and test process in a simulation-based test-bed, as a mechanism for evaluating the situation-based testing approach.

O5 To evaluate the suitability and limitations of the proposed situation-based testing approach.

1.3 Technical Approach

A background review of ubiquitous computing was conducted to identify the category of ubicomp systems most relevant to this thesis. Further research of the literature identified a set of recognised challenges and open issues that must be addressed in order to advance support for designers of ubicomp systems, particularly indoor, location-tracking systems.

An exploratory study of the state of the art looked at a broad set of tools ranging from high investment, high fidelity live environments to lower investment, lower fidelity prototyping and simulation tools. These systems were analysed and compared with a view to understanding the advantages and disadvantages of each in terms of addressing the challenges identified in the background review. A comparison framework was devised as an instrument to draw comparisons about the contributions of each tool, specifically in reference to the challenges central to this thesis. The combination of this comparative analysis with the findings from the background review helped shape a list of established and emerging requirements for testing and evaluation tools that accommodate indoor, location-tracking, ubicomp systems.

Research objective ***O1*** is achieved through the combined output from the background and state of the art reviews.

The requirements formed from the analysis of the state of the art review provide the basis for the design of the situation-based testing approach. This testing approach is realised in a two pronged design comprised of the *InSitu Model Framework* and the *InSitu Test Process*. The InSitu Model Framework provides the key building blocks for specifying a simulation-based test including situations, the environment and user activity. The Test Process defines the role and realisation of the situation-based testing approach in the context of an iterative prototyping cycle.

Research objectives ***O2*** and ***O3*** are achieved through the definition of the InSitu Model Framework and the InSitu Test Process, respectively.

A technical implementation of the situation-based testing approach is realised in the InSitu Toolset. The Toolset realises the InSitu Test Process to enable run-time monitoring of a simulated ubiquitous computing environment using an implementation of the InSitu Model Framework to reason about the state of the environment. Output from a test cycle provides the feedback which can be analysed by the ubicomp designer or tester between iterative prototyping cycles. The InSitu Toolset provides the mechanism for conducting the evaluation of the situation-based testing approach proposed in this thesis. Performance testing was conducted to ascertain that the Toolset implementation is suitable as a mechanism to conduct the evaluation of the situation based testing approach.

Research objective ***O4*** is achieved through the implementation of the InSitu Toolset.

The research evaluation methodology is comprised of two case studies, a user trial and a comparison framework analysis. A real life example is chosen as a case study to investigate whether the situation-based testing approach proposed in this thesis can correctly identify real life examples of inappropriate situations. A second case study, builds on this finding, by increasing the complexity of both the physical deployment environment and the situation specifications, for a multi-user problem. A user trial was conducted to objectively assess the expressiveness and suitability of the InSitu approach for testing ubicomp systems. The final part of the evaluation used a comparison framework as an instrument to compare the InSitu approach to the tools reviewed in the state of the art study.

Research objective ***O5*** is achieved through the combined output of these evaluations.

The evaluations led to a set of conclusions which include an assessment of the suitability and limitations of situation-based testing for ubicomp systems, as well as possible future work.

1.4 Thesis Contribution

There are two contributions of this work, one major and one minor. The major contribution is a situation-based testing approach which enables the designer to test the effects and outcomes of exhibited ubicomp system behaviour in a simulated ubicomp deployment environment. This is important because the mapping between system behaviour and the state of the physical environment is not one to one, as is more traditionally the case for the desktop computing paradigm. Instead, a ubicomp system must be able to operate in harmony with the effects of user activity. However, the less bounded scope on inputs and activities in a ubicomp deployment, and the lack of prior knowledge about how spatio-temporal relationships will evolve, make it difficult to test the actual effect of a ubicomp system. The situation-based testing approach is novel in that it provides a generalised approach to testing the summative effect of exhibited system behaviour and spontaneous user activity in a simulated deployment environment.

The minor contribution is traceability of the causal factors leading to specific situations. It is not sufficient to identify inappropriate situations that occur in the environment. Designers must also be supported to uncover how these situations came about. Feedback analysis has been demonstrated as a tool to trace the causal factors, both physical and digital, that are instrumental in bringing about specific situations. While this has significant potential, further research is required to achieve seamless analysis for a broader range of the physical-digital environment.

1.5 Thesis Outline

Chapter Two presents a review of background literature relevant to ubiquitous computing (ubicomp), including a discussion on categorisation of ubicomp systems which narrows the scope of the thesis to focus on action-based ubicomp systems. The chapter also presents the challenges associated with testing and evaluating action-based systems, with particular reference to the combination of the loose-coupling and spatio-temporal dependencies that exist in ubicomp deployment environments.

Chapter Three presents a survey of the state of the art in testing, evaluation and prototyping tools, suitable for action-based ubicomp systems, and particularly those that address the challenges identified in chapter two. A comparison framework is defined and used as an instrument to draw comparisons about the contributions of the reviewed tools.

Chapter Four presents a set of requirements, which were informed by the background and state of the art reviews, and which set out both novel and conformist technical requirements necessary to realise the situation-based testing approach. Chapter four discusses the formal definition of the InSitu Model Framework to support test specification. Also discussed is the InSitu Test Process, which defines the situation-based testing approach in the context of an iterative prototyping cycle.

Chapter Five presents the implementation of the situation-based testing approach in the form of the simulation-based InSitu Toolset. The components of the Toolset are discussed, including the relationships between components and the implementation of the Model Framework for the Toolset. The realisation of the Test Process, through the Toolset, is described including an illustration of the role of the Model Framework in the Test Process. Finally, the performance of the Toolset is presented to demonstrate the implementation is suitable for conducting the evaluation of the situation-based testing approach.

Chapter Six describes the evaluation work undertaken to validate the contribution of this thesis. The discussion of a real-world case study demonstrates that InSitu Toolset can correctly identify real-world inappropriate situations in a sensor-driven system. A case study of a hypothetical system for a real-world environment demonstrates an example of a more complex test specification, both in terms of the environment and inappropriate situations. The results from a user trial are presented and analysed to provide an objective assessment of the suitability of the situation-based testing approach for assessing indoor action-based ubicomp systems.

Chapter Seven presents the conclusions of the thesis. This includes key findings relating to the major and minor contributions of the work, as well as a description of the successful completion of the research objectives. The thesis concludes with a discussion of possible future work, both technical development and further research.

Chapter 2

Background

2.1 Introduction

This chapter builds on the motivation section to address the background issues which informed the direction of this thesis. Chapter 1 discussed the need for indoor ubicomp systems to be well-designed in order to maximise the benefit to end-users. A combination of related issues were noted that raise new challenges for testing and evaluating these systems. This chapter elaborates on these challenges.

In the first part of this chapter, the general field of ubiquitous computing (ubicomp) is discussed in order to narrow the focus of this thesis to the specific subset of ubicomp systems that are most affected by these challenges. The following section identifies and describes the challenges in testing and evaluating this subset of ubicomp systems, with particular emphasis on those challenges most relevant to the motivational issues already outlined. The chapter begins with a brief discussion on terminology to differentiate between ubiquitous computing, pervasive computing and context-aware computing.

2.2 Terminology

The term *ubiquitous computing*[1] (ubicomp) was coined by Mark Weiser [119], at PARC, to describe context-driven systems that are designed to meet end-user needs and which are heavily embedded in the user's environment. Although the terms ubiquitous computing and *pervasive computing* are often used interchangeably, a literature survey by Ronzani [93] uncovered subtle differences between the two. Pervasive computing is a term that appeared four years after ubicomp, predominantly in relation to the company Novell and in particular to refer to Novell's business strategy of widespread network connection for information users, i.e. focusing on networked devices and the distribution of information to people.

Lyytinen and Yoo's [74] discussion of ubiquitous computing provides a similar view of the field. In their paper the authors provide an illustrative chart that maps the fields of ubiquitous computing, pervasive computing, *mobile computing* and *desktop computing* in terms of their level of embeddedness and mobility. The authors' key finding is that while ubiquitous computing demonstrates high levels of both mobility and embeddedness, pervasive computing and mobile computing are each only extensive in one dimension, i.e. respectively embeddedness and mobility. Lyytinen and Yoo show that advances in the ubiquitous computing realm must build on the advances from both mobile and pervasive computing.

Another term of significant relevance is context-aware computing, which is credited to Schilit et al. [97, 99]. These researchers were particularly interested in mobile, distributed computing, under the direct influence of Mark Weiser's work on ubiquitous computing. Schilit and Theimer described the challenge of the user's changing circumstances and the need for systems to respond to this. They created the term context-aware computing to describe software that could adapt to changes in the location of people and objects. Later contributions from researchers such as Dey and Abowd [36] generalised this definition by lessening the emphasis on location

[1]The term ubiquitous computing was actually first used by Steve Jobs three years prior to Mark Weiser however for a different purpose. Jobs was marketing the Apple II desktop computer as an educational resource (a ubiquitous computing resource) that could be used *everywhere* on campus. [93]

information. Dey and Abowd defined context aware computing as:

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task.

The final term introduced in this section is Ambient Intelligence (AmI), a term that originated in Philips [93, 58]. Many researchers describe AmI as the convergence of many of the concepts central to ubicomp, pervasive computing and context-awareness [46, 44]. However, there are conflicting views about the extent to which AmI is distinct from ubicomp. As an example of this, reports from the Information Society Technologies Advisory Group in 1999 and 2003 clash in their assertions. In 1999 this group noted that ubicomp and pervasive computing are enablers of AmI, however in a separate report, from 2003, the group suggested that AmI was not clearly differentiated from its predecessors, and that further effort could help clarify its exact nature.

Due to the similarity between ubicomp and AmI, this thesis will draw on research from both fields, however the term ubicomp will predominantly be used because it is more firmly established in the literature.

This thesis will use these terms according to the following definitions:

Mobile computing refers to mobile devices that use the wireless network to connect to resources and services, and that are portable.

Pervasive computing refers to physical environments richly embedded with networked sensors, actuators and computing devices to support distribution of information and services.

Context-aware computing uses context or situational information to provide relevant information and/or services to the user, where relevancy depends on the user's task or situation.

Ubiquitous computing refers to environments that feature mobile computing, pervasive computing and context-aware computing, in order to support systems that react or adapt in response to changing conditions and situations in the environment, with the over-arching aim of meeting end-user needs. Ubiquitous computing is identified as the main focus of this thesis.

2.3 Design Issues for Ubicomp Systems

Weiser's opening statement of his seminal paper *The Computer for the 21st Century* [119] places heavy emphasis on the link between successful technology and the ability of the technology to become invisible. Weiser's vision for ubiquitous computing is heavily user-centric in favour of systems that provide background support in order to free the end-user from 'using' technology. Weiser closes this paper with the statement:

> "Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking a walk in the woods."

Milner [76] describes this idea as users being embedded in the ubicomp system. Pederson and Surie [88] use the term egocentric to indicate the significant paradigm shift that this aspect of ubicomp introduces. The use of the term egocentric also signifies that the human, body and mind are at the centre of interaction in the ubicomp world.

Fundamental to Weiser's vision is physical integration of ubicomp systems in the user's environment, a key characteristic identified and discussed by Kindberg and Fox [65]. To achieve this shift away from the desktop requires that systems are embedded in the physical environment with the result that user interfaces can be distributed in the environment. Embedded sensor technology is a key enabler of this by providing dynamic input that is not solely from the user's mouse and keyboard. This distributed, embedded input source is matched by embedded actuator technology. The combination of these technologies enables systems to sense end-user needs and respond to them by proactively actuating the physical environment. The result is a user interface, that crosses the boundary of the physical environment and the digital environment of the computer system. This user interface is not only distributed in the physical-digital environment, but it can also be invisible, i.e. indistinguishable to the user from the physical environment.

It is evident from this description that the state of a ubicomp deployment is impacted by elements of both the physical and digital environments. It can be considered that two entities exist in a ubicomp environment with the capability to proactively alter the state of the environment. The first are the users who can change both their own personal state, e.g. their position, as well as the state of mobile, fixed and embedded devices in the environment. The second is the ubicomp system which can perform actions and instructions on networked and actuatable devices and services in the environment. Ultimately the state of the ubicomp environment, at any given time, is the summative effect of user activity, system behaviour and the physical environment.

One of the most significant changes introduced by this aspect of the ubicomp paradigm is the *dynamic spatio-temporal dependency* that exists between users, the system and the environment, which can significantly impact the effectiveness and appropriateness of ubicomp system behaviour. This is brought about by the combination of pervasive, mobile and context-aware computing, that together enable ubicomp. There are two specific relationships that are key in this:

1. Pervasive and context-driven computing together introduce an inextricable link between the physical environment and a computing system.
2. Mobile and context-driven computing together introduce tight spatio-temporal dependencies between user activity and system behaviour.

As a result, ubicomp systems must be designed to accommodate both the physical environment and natural user activity and normal work-flows. Physical environments are generally difficult to adjust to any significant extent, without requiring building or demolition work. Users are the centre of the ubicomp environment, and so disrupting their behaviour breaks the ubicomp ideal. Consolvo et al. [30] illustrate this in their discussion on experiences designing the Labscape system. In their paper, Consolvo et al. state that systems must be designed with consideration for the target environment and end-users, particularly to ensure that the ubicomp system does not disrupt the end-user's natural work-flow.

For this reason, the term *design considerations* is defined for use in this thesis, in reference to testing and evaluation of ubicomp systems:

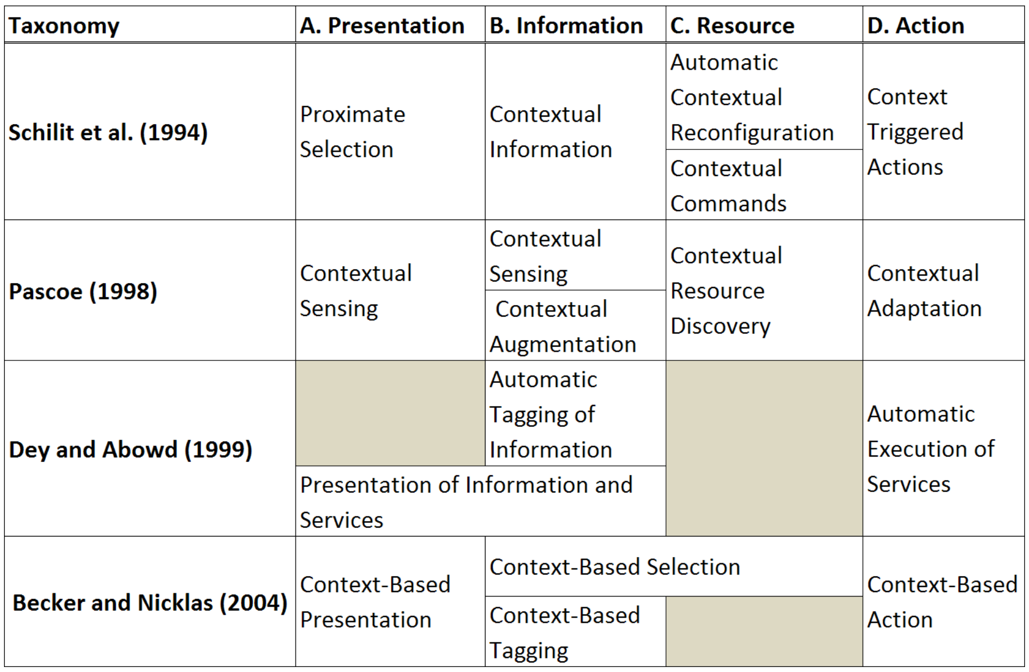
Design considerations refer to entities or aspects of the environment that impact the state of the environment, but that cannot or should not be forced to change in order to resolve problematic behaviour, i.e. the physical structure of the building and the natural activity or work patterns of users.

2.3.1 Categories of Context Aware Systems

Ubicomp systems feature context awareness so they can adjust their configuration, presentation and actions according to context relating to users, entities and the physical environment, and can function without explicit user instruction [5, 86, 7]. Dey and Abowd discuss the characterisation of context extensively in their papers [4, 35]. The resulting well accepted definition of context refers to situations defined by *people*, *places* and *objects*:

> Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application.

There have been some attempts to categorise the functionality of context-aware systems, notably by Schilit et al. [98], Pascoe [86], Dey and Abowd [36], and Becker and Nicklas [16], who's taxonomies all share elements in common. For the purpose of this thesis, these taxonomies are summarised using four categories to reflect the commonalities that they share: (A) Presentation of Information and Services, (B) Information Annotation and Selection; (C) Discovery, Configuration and Selection of Resources and Services; (D) Action, Instruction and Actuation, as summarised in Figure 2.1. This categorisation is used to narrow the scope of this thesis by identifying and defining the target subgroup of context aware systems that are most impacted by the previously identified testing and evaluation challenges for ubicomp systems.



Taxonomy	A. Presentation	B. Information	C. Resource	D. Action
Schilit et al. (1994)	Proximate Selection	Contextual Information	Automatic Contextual Reconfiguration	Context Triggered Actions
			Contextual Commands	
Pascoe (1998)	Contextual Sensing	Contextual Sensing	Contextual Resource Discovery	Contextual Adaptation
		Contextual Augmentation		
Dey and Abowd (1999)	▨	Automatic Tagging of Information	▨	Automatic Execution of Services
	Presentation of Information and Services		▨	
Becker and Nicklas (2004)	Context-Based Presentation	Context-Based Selection		Context-Based Action
		Context-Based Tagging	▨	

Figure 2.1: Categorisation of Context-Aware Systems

A. ***Presentation* of Information and Services:** This category focuses on the issues surrounding user and device interfaces, and appropriate presentation and formatting of information to best support explicit interaction and/or task-centric user actions. Becker and Nicklas [16] provide the example of a navigation system which might adapt its view based on the speed of travel. This is primarily a concern for device interfaces, including mobile, large-screen, shared, private and public displays. As such, this is of lower relevance to the more environment-centric concern of the spatio-temporal relationship that exists between users, entities and locations. For this reason, this category has been ruled out of the scope of this thesis.

B. ***Information* Annotation and Selection:** This category focuses on management of information and appropriate selection of information, relevant to the current contextual situation. In parallel to the advances in sensor technology, context management systems have emerged to gather, manage, evaluate and disseminate context information, making the context aware computing paradigm more feasible [17]. This category is largely concerned with finding and filtering information for end-users, as discussed by Brown and Jones [18]. The issue of finding and filtering information, although necessary to support context-aware systems, is not central to investigating the appropriateness of behaviour exhibited at the distributed interface of ubicomp systems. As discussed by Yeung et al. [122], this is generally handled by the frameworks and middleware that support context-aware system development. For this reason, this category has been ruled out of the scope of this thesis.

C. *Resource* Discovery and Configuration: This category relates to appropriate use of resources and services when availability changes dynamically in the environment. For example, appropriate selection of wireless connectivity to provide the best quality of service to the end-user. This is an issue that is generally managed below the application level, as mentioned by Rakotonirainy and Groves [92]. As such, this is not part of the general application-level design problem, and is considered outside the scope of this thesis.

D. *Action*, Instruction and Actuation: This category relates to appropriate actions, invocations and actuations performed by a ubicomp application in the environment and is closely linked to the system's decision logic. Cook and Das [31] define the decision layer of a smart building application, as the layer responsible for controlling action in the physical layer. The physical layer refers to the actuators and device controllers which can trigger and change the state of the world. This is central to the scope of this thesis because it directly impacts on the spatio-temporal relationship that exists between users and the physical environment. For brevity in this thesis, this category will be referred to as *action-based* ubicomp systems.

2.3.2 Summary

This section provided a discussion on ubiquitous computing, but placed emphasis on the issues directly related to the interface between physical and digital environments. This section also looked at dependencies this introduces between users, physical space, system behaviour and the temporal domain. Finally, the term *design considerations* was defined to refer to entities or aspects of the environment that impact the state of the environment, but which cannot or should not be forced to change in order to resolve problematic behaviour.

The second part of this section presented four categories of context-aware systems: Presentation of Information and Services; Information Annotation and Selection; Discovery, Configuration and Selection of Resources and Services; and Action, Instruction and Actuation. These four categories were informed by previously published taxonomies. The relationships of each of the categories to other taxonomies was presented in Figure 2.1. The latter of these categories, referred to as action-based ubicomp systems, was identified as the focus of this thesis. The next section of this chapter looks at the challenges relating to testing and evaluation of action-based ubicomp systems.

2.4 Challenges for Ubicomp Testing and Evaluation

As noted by Satyanarayanan [96], there are new challenges introduced by ubicomp that have no obvious mapping to previous paradigms. Similarly, Davies et al. [34] note that new approaches are required to address the new dimensions of complexity that have been introduced by ubicomp. This discussion, on the challenges surrounding testing and evaluating ubicomp system behaviour, focuses on the issues most relevant to action-based ubicomp systems. Further, this discussion specifically focuses on those challenges that are directly relevant to the issues central to the motivation for this thesis, i.e. (a) the tight spatio-temporal dependency between physical and digital environments, (b) loose coupling between inputs, outputs, cause and effect, and (c) spontaneity of user activity and dynamism in users' environments.

In freeing users from the desktop, designers are faced with the challenge of creating invisible, distributed interfaces that must respond appropriately to spontaneous user interaction, both explicit and implicit. Testing and evaluating these systems is particularly challenging because it is difficult for the designer to trace the exact system behaviour, experienced by multiple individuals, under various circumstances. Banavar and Bernstein [9] note the challenge of accommodating the spontaneous and often unpredictable nature of humans, particularly since ubicomp users can trigger actions in both devices and the physical environment. In a recent publication[2], García-Valverde et al. [46] itemised a list of design and testing challenges for ubicomp systems which includes the high variety of scenarios, heterogeneous devices, different sources of information, changing contexts, unpredictable and changeable behaviours and adaptation to users.

Ubiquitous computing also shares design and engineering challenges with more mature fields such as networking and distributed systems. However, these issues are not the focus of this thesis for two reasons. Firstly, network simulators are well-established and it has been successfully demonstrated that they can be integrated and reused in ubiquitous computing test-beds. Examples of this include the work by Morla and Davies [78] using the ns2 simulator[3] and the work by Huebscher and McCann [55] using TOSSIM[4]. Secondly, network management, although crucial to enabling a ubicomp environment, is not elemental to the spatio-temporal relationship between users and the environment. Similarly, issues such as service and resource discovery, interoperability, network management, device interfaces, privacy and security are all of significant importance, but addressing these concerns does not assist designers in coping with spatio-temporal dependencies during testing, investigating loosely coupled causal factors, or anticipating the

[2]Year of publication: 2010.

[3]ns2, Network Simulator (download): http://www.isi.edu/nsnam/ns/ (Accessed 20th September 2010).

[4]TOSSIM, Download as part of TinyOS: http://webs.cs.berkeley.edu/tos/download.html (Accessed 20th September 2010).

unexpected in the physical-digital environment.

The following sections will first discuss some long-standing issues associated with testing and evaluating ubicomp systems, including scalability, heterogeneity of context sources and heterogeneity of deployment environments. Following this, challenges more directly related to testing the effectiveness of exhibited system behaviour will be addressed. These include the appropriateness of the effect of system behaviour in the physical environment, the problem of anticipating the unexpected, and the difficulty tracing loosely coupled cause and effect.

2.4.1 Scalability

The first obvious challenge, for ubicomp system testing, is the scale of the distributed computing environment. Sitou and Spanfelner [106] maintain that to ensure acceptance of ubicomp applications, tests must be possible at scales that support systematic investigation of all aspects of contexts and their change over time. Similarly, Lyardet et al. [72] identified that a central challenge for these tools is to enable designers and developers to rapidly explore large scale environments and extensive scenarios that are infeasible to recreate in the real world due to the effort and complexity involved.

Abowd [2] notes four aspects relating to scale of ubicomp environments: physical space, number of people, number of devices and length of time.

- **Physical space** refers to the deployment environment in which a ubicomp application will operate. Scale is not limited to floor area, also important are the quantity and types of zone that make up the total area, as well as the number of levels in indoor environments, i.e. floors in a building.
- Scale in terms of **people**, can be challenging when faced with large numbers of mobile users. It can be difficult to recruit large numbers of volunteers when running a trial with a large scale mobile ubicomp system, particularly due to the issues of privacy and intrusiveness often associated with sensors and

environment monitoring. Additionally, any trials involving real users introduce the overhead of arranging ethical approval for the study. Finally, tracking mobile users in large environments becomes linearly more difficult with each added user.

- The third aspect of scale is that testers need to be able to deploy large quantities of **devices**. Often researchers are limited in the scope of a test by a finite amount of available resources.
- Finally, scalable **time** (duration) is necessary during testing because many ubicomp systems are designed to run in the background, continuously responding to the changing environment. Additionally, a user's circumstances are not defined by their surroundings alone, time has a significant impact on end-users, whether it is the physical difference between darkness at night and light during the day, or the passing of time and its impact on diaries and schedules, or the relative time difference between meals and the need for refreshment. Potts et al. [89, 90] were among the first to write about the tight relationship between time and location, and the impact of this on ubicomp system design.

Designers need tools which can scale to the appropriate dimensions of physical space, number of users and devices involved, and the length of time of a test. Testing conducted using live deployments, although offering high fidelity, is ultimately limited in terms of finite physical space. On the other hand, in the same way that modelling and simulation has proven beneficial to both network engineers and civil engineers, it has also been demonstrated as useful for the purpose of testing ubiquitous computing systems, e.g. AmISim [46], UbiREAL [80], QuakeSim [21] and UbiWise [13].

2.4.2 Heterogeneity

Heterogeneity is often cited directly in relation to context generation and devices. Context-awareness has evolved so that there are few bounds on what can be considered context [100]. Advances in hardware and processors have resulted in smaller, more powerful, sensor motes, which means that context can be generated in types and quantities previously not freely available [101, 112]. Some examples include: location, orientation, velocity, time, device configuration, device capabilities, user preferences, temperature, humidity, luminosity, physiological state and emotional state [35, 4, 49, 52, 97, 94, 101]. Additionally, permutations and combinations of this information can support derivations of implicit knowledge, e.g. an application that has information about time and location might be able to infer when a user is on a coffee break.

Satyanarayanan [96] talks at length about the issues related to diversity in context information. This information can differ in terms of source, format, freshness and complexity. The source can be sensed (location, orientation) or non-sensed (user profile, device specification). Data formatting differs across sources, e.g. according to vendor, as discussed by Casola et al. [23] in their work on a common data model. Context sources also vary in freshness, as discussed by Henricksen et al. [51]. Static information can be considered 'fresh' all the time since it never changes. Dynamic information changes at different rates, e.g. the relationship between co-workers may exist for years, however a user's location information changes regularly through the day.

Designers will need access to diverse sources of context in order to test a variety of ubicomp systems. This can also be beneficial during the design cycle because it may not be immediately apparent which context sources are most suitable for a specific deployment configuration. Availability of heterogeneous sources enables the designer to test different configurations.

As distinct from the previous paragraphs, heterogeneity can also refer to diversity of ubicomp deployment environments, e.g. home [121], office [115] or scientific laboratory [30]. This can be due to significant or subtle differences between purpose built structures. For example, both hospital and hotels can be characterised as containing many beds. However, hospitals are likely to also have a lot of medical equipment, whereas hotels are more likely to contain a lot of leisure facilities.

It is unlikely that designers will be limited to developing one system for one particularly deployment environment. It is more likely that designers will develop many systems, and probably for many different types of buildings. Designers may also face the challenge of developing one system that must be capable of operating appropriately within many different deployment environments. Gaining access to deployments for testing is not always possible, particularly if environments are already operational with users occupying the space on a daily basis. However, testing ubicomp systems in unsuitable environments can generate misleading results. Consolvo et al. [30] discuss the importance of evaluating ubicomp systems in appropriate settings with appropriate test-users.

In the same way that network simulators can assist designers to test different network configurations, ubicomp simulation tools can assist ubicomp designers to test different physical configurations and so address the challenge of heterogeneous environments for testing ubicomp systems.

2.4.3 Appropriateness of System Behaviour

The term unwanted behaviour was first defined by Fahrmair et al. [39] as a new form of problematic behaviour which arises for a certain combination of user, situation and environment. Unwanted behaviours are exhibited system behaviours, that are either perceivable by the end-user or result in an inappropriate situation that is apparent to the end-user. The challenge lies in the issue that these are not defined as technical errors and often only result for a certain combination of user, situation and environment. It is a failure though because under these conditions the system does not meet the end-user's needs. The result can be that the system imposes an unnatural pattern of behaviour on the end-user [30] or can leave the user in a physically or psychologically uncomfortable situation [110].

The challenge for the designer is that inappropriate situations are not limited to manifesting within the digital environment. Actuation technology enables inappropriate situations to also exist in the physical environment. As a result, the appropriateness of exhibited behaviour can be subject to the summative outcome of user activity in the physical environment and actuations stemming from the digital environment.

In a live test-bed, when a designer is conducting multi-user testing, the designer cannot physically be in enough places in order to monitor the effects of system behaviour as experienced by each individual user. An aspect of user observation approaches, e.g. Wizard of Oz testing [69], is that the designer shadows individual users. This can provide an opportunity to observe users' experiences, however there are practical issues of scale, specifically in terms of the number of test-observers required. Alternative approaches have used video recordings of live environments, however again scalability has been identified as a limiting factor in practical terms. Video recordings generally require annotation in order to make use of the information contained in the recordings. Consolvo et al. [30] reported on their experience of this process, during which they found it required eighty-five hours of manual annotation to process eighteen hours of video footage.

Ultimately, during testing, the challenge of determining the appropriateness of system behaviour stems from the physical-digital boundary that exists between the situation experienced by the end-user in the physical environment, and the digital environment of the ubicomp system. This is exacerbated by the issues of scale and the distributed nature of both the physical and digital environments.

2.4.4 Unanticipated Situations

In a discussion on Robin Milner's paper on ubicomp [76], Chalmers [107] addresses the issues of the unanticipated and unexpected. According to the ubicomp paradigm, users are not trapped in the confines of procedural tasks with the purpose of reaching an end goal. The spatio-temporal freedom introduced by mobility and embedded sensors means that computing is *living* with users and not just operated by users. As a result, there is wider scope for user behaviour to impact on system output. Davies et al. [34] also note the difficulty in trying to anticipate user behaviour and user acceptance in advance in order to deliver well-designed systems.

Scenarios are a well established medium to describe how a computer system might be used. For desktop applications the user generally follows a sequence of steps to achieve their goal, e.g. to open a document, start a print job or visit a website. Scenario-based approaches are useful, in this case, for capturing how a user might interact with the system. There are a clear set of steps from start to finish with a clearly defined goal. However, scenarios are inherently rigid in their procedural definition. Tang et al. [111] note that prediction-based design approaches will be more complex and often ineffective when used for ubicomp systems because of the unknown factors.

Chalmers asserts that although it is possible to put forward definitions of correctness for ubicomp applications, it is not yet possible to incorporate the unanticipated. Although Chalmers does not explicitly state it, the inability to account for the unanticipated during testing means that it is difficult to guarantee completeness in testing. The designer cannot be expected to anticipate every situation that will arise and write a scenario to describe it. A research challenge exists in terms of finding approaches that will assist designers to deliver robust applications in the face of the broader input-output scope.

It is proposed here that the spontaneous interaction that is characteristic of ubiquitous computing, as discussed in the literature [8, 9, 65], could be tested using situations rather than predefined scenarios. In fact, situation was used as an early synonym for context, as discussed by Dey [35]. The reactive and adaptive nature of ubicomp applications means that it is more difficult to define a goal and the sequence of steps that lead to it. Additionally, work by Henricksen and Indulska [51], Nishikawa et al. [80] and Clear et al. [28] have made contributions towards formalising the specification of situations. However, despite the trend towards discussion of contextual situations in the literature, few application design tools have incorporated situation-based testing as well as scenario-based testing.

It is worth noting at this point a distinction in two aspects of unanticipated situations, these are *unanticipated situation specifications* and *unanticipated* *situation instances.* A situation specification is a general definition of a situation, for example using logical operators to combine variables that represent contextual attributes. A situation instance is an occurrence of a situation specification at run-time for a specific user, location or other environment state. For the purpose of discussion in this thesis, and specifically in reference to Fahrmair et al.'s definition of unwanted behaviour [39], it is the latter of these that is referred to when the term unanticipated situation is used.

2.4.5 Traceability of Causal Relationships

Closely linked to the challenge of testing the appropriateness of system behaviour, is the difficulty in trying to trace the causal factors leading to specific inappropriate situations. Huebscher and McCann mention this challenge specifically for context aware systems that react to the environment [55], noting that tracing areas of poor performance is often difficult. Tang et al. [111] mention traceability as an open issue even when limited to the digital environment, through analysis of logged device events and interactions.

As previously mentioned, situations that arise in the physical environment are the summative effect of user activity in the physical environment and system behaviour originating in the digital environment. The physical environment is not fully connected or integrated with the computing system, and so designers cannot automatically trace relationships between cause and effect across this boundary. It is the complexity of relationships between user activity, spatial and temporal relationships, and the situations that arise as a result of exhibited system behaviour, combined with the loose coupling between physical and digital environments, that make traceability difficult in ubicomp deployments.

Traceability for ubicomp systems is often cited in relation to privacy. Weber et al. [116] refer to it as "attributing events and actions to those *who* cause them". However, during testing, when attempting to uncover problematic and unwanted behaviour, the aim is not to ascribe blame by identifying who caused a problem.

Instead, Weber et al.'s definition is generalised and traceability is referred to as *attributing events and actions to* ***that*** *which caused them.* Events and actions refer to behaviours exhibited by a system at run-time.

Tools are needed that can analyse the causal relationships between physical and digital events so that a full set of causal factors can be established even in the face of the challenging conditions of ubicomp deployment environments.

2.4.6 Summary of Challenges

This discussion on the challenges faced by ubicomp designers during testing and evaluation of ubicomp systems focused on the subset of systems that are action-based. Action-based systems perform actions based on a user's current contextual situation but without requiring explicit input or instruction from the end-user. These systems are also capable of performing actions in the physical environment through the use of distributed actuators. For these reasons, this discussion on challenges focused on the general, long-standing issues of scale and heterogeneity, which are directly linked to implicit inputs and outputs that are distributed in the physical environment. Scale refers to the quantity of devices and users, the amount of physical space and the duration of a ubicomp system test. Heterogeneity was discussed in terms of the types of devices and context sources in physical environments, as well as the different types of deployment environments in which a ubicomp system can be expected to operate.

Additionally, challenges that are more specific to testing and evaluating action-based systems included the difficulty in determining the appropriateness of exhibited system behaviour, anticipating the many situations a system will encounter, and tracing the causal factors behind inappropriate situations. In terms of identifying inappropriate situations, it is difficult for the designer to monitor the distributed environment, particularly as tests scale to larger environments and greater quantities of users which in turn increase the complexity of testing. The mobile, embedded style of ubicomp has freed users from the desktop providing more freedom for user activity to impact the effectiveness of system behaviour. This less-bounded scope however, makes is difficult for the designer to anticipate the many situations that the system will encounter post-deployment. Finally, the loose coupling of inputs and outcomes, combined with their existence across the physical-digital boundary, makes it difficult for the designer to trace the causal factors leading to specific outcomes.

2.5 Chapter Conclusions and Summary

This chapter began with a discussion of the key areas closely related to ubiquitous computing, including mobile computing, pervasive computing, context-aware computing and ambient computing. Each term was grounded in a background review of the literature and was defined with reference to how these terms will be used in this thesis. The focus of this thesis is primarily on ubiquitous computing, which was defined as: *ubiquitous computing refers to environments that feature mobile computing, pervasive computing and context-aware computing, in order to support systems that react or adapt in response to changing conditions and situations in the environment, with the over-arching aim of meeting end-user needs.*

This was followed by a discussion on design issues in ubicomp systems, with emphasis placed on the issues central to the motivation and research question for this thesis, i.e. those surrounding the strong dependency between users, physical space, system behaviour, the temporal domain, and the appropriateness of situations arising in the environment. This part of the discussion also investigated categories of context-aware systems and defined the category of action-based ubicomp systems, which is the main focus of this thesis.

The third and final section of this chapter looked at the challenges related to testing and evaluation of action-based ubiquitous computing systems. These included scalability, heterogeneity, unanticipated situations, appropriateness of system behaviour, and traceability of causal factors. Scalability and heterogeneity are long recognised challenges associated with designing ubicomp deployment environments. Simulation has repeatedly been demonstrated as a useful tool in addressing these issues, for example in AmISim [46], UbiREAL [80], QuakeSim [21] and UbiWise [13].

The challenge of unanticipated situations stems from the combination of context-awareness and mobile users, which together mean that ubiquitous computing applications are more exposed to the spontaneous and sometimes unpredictable nature of user behaviour, than other computing paradigms have been. In light of this, it was proposed in this chapter, that a situation-based testing approach could provide a potential alternative to scenarios when testing the reactive and adaptive behaviour of ubicomp systems.

Assessing the appropriateness of system behaviour is challenging because the effects are not always bounded or tightly coupled to the digital environment. The designer must assess the appropriateness of situations that can encompass both physical and digital environment state. An additional challenge lies in the distribution of the users and devices affected by system behaviour, throughout a physical deployment space. Traceability of causal factors is challenging because of the loose coupling between cause and effect across the physical-digital boundary.

2.5.1 Chapter Summary

In summary, this chapter narrowed the scope for this thesis to action-based ubicomp systems, with particular focus assessing the appropriateness of exhibited system behaviour. The next chapter investigates the state of the art in testing and evaluation tools which have made contributions to this area.

Chapter 3

State of the Art

3.1 Introduction

Tools and approaches for testing and evaluating ubiquitous computing systems remain active research issues, as noted by Tang et al. [111]. For this reason, a broad set of tools, suitable for testing action-based systems, are included in this review. The tools highlight the benefits and limitations of various approaches, as well as illustrating some of the current open issues, particularly for early stage testing of action-based ubicomp systems.

A comparison framework has been devised as an instrument for comparing tools included in this survey. This framework is defined in order to highlight the contributions, benefits and limitations of each tool, specifically in relation to testing and evaluating action-based ubicomp systems.

In support of the survey and prior to commencing the survey, a review of models to support these tools is discussed. This review looks at models for situation specification with a view to supporting the situation-based testing approach. Following this, the review looks at models that can represent the design considerations, which were defined in the last chapter, including user activity and the physical environment.

This chapter begins with the review of modelling approaches that can support ubiquitous computing testing and evaluation tools. This is followed by a definition and discussion of the comparison framework. The final sections of the chapter present the survey of tools, test-beds and other approaches, along with a tabulation of the comparison framework and an accompanying analysis.

3.2 Models for Situation-Based Testing

This section discusses modelling approaches suitable for supporting a situation-based testing approach. The first part of this section looks at situation specification to support situation-based testing. In the remaining subsections, the discussion addresses the issue of design considerations, which were defined in the last chapter as *the entities or aspects of the environment that impact the state of the environment, but that cannot or should not be forced to change in order to resolve problematic behaviour, i.e. the physical structure of the building and the natural activity or work-flow of users.* To address the design considerations both location modelling and activity modelling will be discussed.

3.2.1 Situation Modelling

Dey was among the first to refer to situational abstraction [35]. He described it as a level above widgets, interpreters and aggregators, i.e. in current terms a level above the context management system. Dey's motivation in defining the term *situation* was to support context-aware designers to focus on the 'heart of the design process'. He proposed that a tool which focused at the situation level would free the designer of lower-level concerns, so the designer could focus on the features, reactions and adaptations for the ubicomp system.

Bettini et al. [17] provide a diagram which illustrates Dey's references to the situation level. For clarity, their diagram has been reproduced here in Figure 3.1. The diagram illustrates the three layers of abstraction for sensor data and context, which Bettini et al. define. At the base of this triangle is low-level context information, which is data in its least processed form, direct from the source, e.g. sensor outputs. The middle layer is characterised by some semantic interpretation, while still formatted in such a way that the data is composable and reusable. Finally, the top layer, situation relationships, is most closely aligned with Dey's description of the situation level.

Bettini et al.'s definition of the situation level refers to situational relationships as a collection of situations. Dey refers to a situation as a collection of states. Although the language is different, the fundamental meaning is the same. Both are describing a highly abstracted level of context which is heavily focused at the application layer.

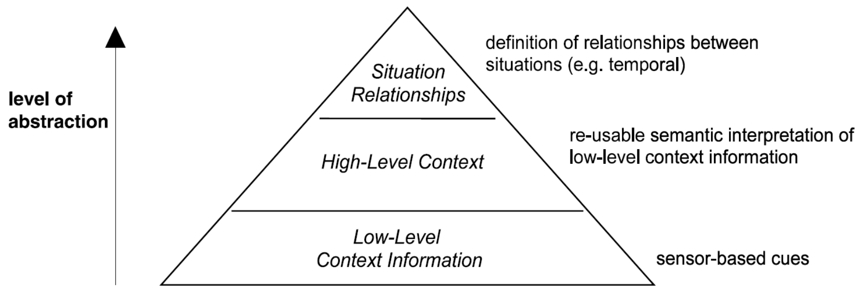

level of abstraction
Situation Relationships
High-Level Context
Low-Level Context Information
definition of relationships between situations (e.g. temporal)
re-usable semantic interpretation of low-level context information
sensor-based cues

Figure 3.1: Bettini et al.'s diagram of the layers of context [17].

Henricksen and Indulska [50] were among the first to formalise Dey's early definition of situation abstraction in their work on the Context Modelling Language (CML). CML provides a formal example of a situation specification model, which uses predicate logic to describe the relationships between fact types (information) from the deployment environment. CML provides constructs to describe context for the purpose of prototyping and fine-tuning context-aware applications, and so can model metatdata as well as contextual information. Although this means that CML extends beyond situation specification, it is fundamentally relevant to situation specification.

CML is based around information tuples, which aggregate information through the use of logical expressions. The main fact types defined by CML are *Person*, *Device*, *Activity* and *Location*. CML also defines some fact types for communication resources in a ubicomp environment, however this issue was ruled out of scope in the background chapter, so within the bounds of this thesis, these fact types are not discussed. In terms of relationships CML defines *located at* (i.e. containment or physical coordinates), *located near* (i.e. proximity), *engaged in* (i.e. a user's relationship to an activity) and *permitted to use* (i.e. resource permissions). CML also makes use of logical relationships including $\wedge$ (and), $\vee$ (or), $\neg$ (not). This combination enables CML to address both logical and spatial relationships in situation specifications. Henricksen and Indulska define a situation (S) as:

$S(v_1, \ldots, v_n) : \phi$
where
$\{v_1, \ldots, v_n\} =$ *Set of free variables*
$\phi =$ *Logical expression*

Clear et al. [28, 27] have completed work on situation specification to support their work on SituVis, a visualisation tool for situational analysis. SituVis provides intuitive views of situations specifications, using Parallel Coordinate Visualisation, which plots discrete pieces of information on a set of parallel vertical axes, each axis representing a type of context. As part of this work, Clear et al. developed a formal definition of a situation specification:

> A situation specification consists of one or more assertions about context that are conjoined using the logical operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$). Assertions may comprise further domain-specific expressions on context, given that the required semantics are available.

This provides the formal language required in order to chart situational data. Additionally, it takes steps beyond previous efforts by contributing towards formally differentiating between interpreted context and a situation specification. The work by Clear et al. is not limited to situation specification. This work also looks at the relationships that can occur between situations, i.e. in terms of overlap between situations or situations that subsume other situations. In this way SituVis enables evaluation and analysis of situations with a view to uncovering potential conflicts, and so affords designers an opportunity to gain better understanding of the situation space.

Nishikawa et al. [80] use a computation tree logic (CTL) approach to situation specification, in their tool UbiREAL. Their aim is to validate a system implementation using a specification defined in CTL. Using this approach, Nishikawa et al. use multiple situation specifications together to define a service specification. To achieve this, it is necessary to create a formal situation definition that can be used with the CTL language. The authors define the state of the environment (U) as a tuple (R, D), where R is the set of all rooms and D the set of all devices, as follows:

$U = (R, D)$

where:

$R = \{r_1, \ldots, r_n\}$

$D = \{d_1, \ldots, d_n\}$

Within these definitions, $r_j \in R$ has attributes: position (pos), shape and size ($base$), capacity (cap), temperature ($temp$), humidity ($humid$), heat capacity ($heatcap$), illumination ($illum$), and acoustic volume (vol). Attributes are denoted as $r_j.temp$, using temperature as an example. $d_j \in D$ has attributes: room (r), position in a room (pos), and state ($state$). Attributes for devices are denoted in the same way, for example $d_j.r$ denotes the room attribute of device d_j, i.e. the room in which the device is deployed.

In addition to specifying the environment, Nishikawa et al. use CTL to define a situational system specification. This specification is rule based, where the set of rules, $AP = \{l_1, \ldots, l_n\}$, define the system. Each rule is comprised of a condition and an action, i.e. $l_i = (c_i, a_i)$. The authors have not provided a full list of operators, however they use examples such as the following as illustrations:

$$(Exist(u_1, r_1) \wedge 28 \leq r_1.temp \wedge 70 \leq r_1.humid, Aircon_1.on(24, 60))$$

This example specifies that when user u_1 is in room r_1, and temperature and humidity are more than 28C and 70% respectively, $Aircon_1$ should be switched to 24C, with humidity at 60%.

Based on this discussion it can be seen that situation models must address the attributes of and relationships between users, entities and locations. This falls in line with Dey's early discussion on situational abstraction in which he recognises these as the basic elements of a situation [35].

3.2.2 Location and Spatial Relations Modelling

Location information was identified as one of the earliest forms of context by Schilit et al. [98], in their paper that describes an approach for *dynamic customisation* of mobile applications. Their discussion for intelligent adaptation of these services was largely based on location information and spatial relations, motivated by the fact that as a user moves the set of resources available in their surroundings changes. Potts et al. [89,90] were among the first to demonstrate the relevance and importance of location information during the design cycle for applications, in their work to design an online diary scheduling system. More recently, Dias and Beinat [37] showed that enhancing context-based action applications with location information can increase user acceptance for certain types of applications, particularly in the tourism industry.

Often for context aware applications, it is not enough to know the Cartesian coordinates of an end-user. Embedded, mobile, context-aware computing needs to be able to reason about the relationships between users, devices and the physical world. In this respect, it is necessary to differentiate between spatial and location information. Location information can describe the current circumstances of a user or device. Spatial information enables the system to put relative meaning on that location information in order to really benefit the user.

Location Modelling

Broadly speaking, location information can be classified and modelled as physical (geometric) or symbolic information, as discussed by Becker and Dürr [15], Satoh [95], and Bettini et al. [17]. Summarising their discussions on this topic, these types of location information can be described as follows:

Physical location models also referred to as geometric coordinates, represent location as geometric information, i.e. points or areas in a geometrically defined space, for example, the position provided by the Global Positioning System (GPS). Geometric calculations can be performed on physical location information, for example to calculate the distance between two points or containment of a point in a geometric area. For context aware systems, these geometric and geographical models are generally useful for performing the mapping between map data and GPS sensor data.

Symbolic location models also referred to as symbolic coordinates, represent a point or zone by an identifier, often in human readable form and providing semantic information e.g. building name, embedded/fixed device ID or cell ID. Symbolic location models require some knowledge of the relationships between locations in order to infer location information about objects in the environment. For example, a WLAN access point can be defined as being contained (installed) in a specific room, which creates an explicit relationship between the access point object and the physical space. On the other hand, a mobile device connected to this access point has an implicit proximal relationship to that part of the physical building.

Spatial Modelling

Becker and Dürr [15] identify four categories of location model including *set-based*, *hierarchical*, *graph-based*, and *combined graph and set-based*. Set-based models do not define relationships that exist between locations, however both hierarchical and graph-based models can represent this information. Graph-based models are well-suited to navigation tasks, i.e. finding a route between two points. Hierarchical models on the other hand are well suited to representing containment and so naturally lend themselves well to modelling the structure of a building, many examples of which can be found in the literature [57,25,11,95]. These are discussed in the remainder of this section.

Lertlakkhanakul et al. [68] note that this hierarchical containment relationship is also pertinent to user perception. Occupants of a building do not perceive the space in terms of the mathematical morphology, instead human perception of space is hierarchical across places, people and things. This is corroborated in the studies by Lynch [73], which observed that errors in human cognitive maps of the environment are most frequently metrical and rarely topological. Since the focus of this thesis is on indoor environments, this section on spatial modelling focuses on hierarchical and hybrid hierarchical models.

Jiang et al. [57] proposed the AURA Location Identifier (ALI) model, a hybrid location model, which aimed to combine the benefits of both hierarchical modelling and physical location information. ALI features three types of location, *space*, *area* and *point*. A space is a physically demarcated location, e.g. a room or a floor. An area is a virtual space defined for specific technology or applications, e.g. the area covered by a particular wireless access point. A point is defined as the location of an object in the environment, e.g. the position of a user or device. ALI also features a set of operators that can be applied to locations, including *distance*, *contains* and *within*. The distance operator stems from physical location modelling by computing the distance between two points. The contains and within operators stem from hierarchical modelling and can be applied to a hierarchical space tree model, in which each node corresponds to an actual space in the physical environment. Each node in the tree has a set of associated geometrical attributes, which provide additional information about *shape*, *area*, *volume*, *origin* and *spatial orientation*. These attributes mean that the model is not limited to the containment relationship of a pure hierarchical model.

Chen et al. [25] provide one of the early and well-cited formalisms for the hierarchical containment relationship in their ontology for context-aware pervasive computing environments. In their paper, the authors define two subsumption relationships as *spatiallySubsumes* and *isSpatiallySubsumedBy*. These relationships can be applied to places (locations) that are atomic or compound, with the restriction that the cardinality of atomic places for the spatiallySubsumes relationship is zero. Chen et al. acknowledge that the implementation of their model is limited to focus on aspects of the environment that is of interest to their research. For example, they note that they chose to model a room as an atomic location, but acknowledge that other research may require finer granularity. Chen et al.'s model is an example of a pure hierarchical model.

Bandini et al. [10, 11] identified three fundamental spatial relations, which they consider key to ubiquitous computing systems, namely *proximity*, *containment* and *orientation*. They refer to this as their Commonsense Spatial Model to distinguish it from more mathematically based models. In their approach to modelling they specifically aim to address the target physical deployment environment and the components of the ubiquitous computing system. Bandini et al. define proximity as:

> *Two places are said to be proximal if it is possible to go from one to the other without passing through another place.*

Although proximity by distance is not directly modelled by hierarchical models, this definition can be supported. Becker and Dürr [15] noted that hierarchies can be used to compare the distances between positions by considering the smallest spaces in the hierarchy. Objects or positions contained by smaller spaces can be considered closer to each other than positions contained by larger spaces.

Bandini et al. define orientation in terms of four cardinal points, *North*, *South*, *East* and *West*. While proximity and containment are measures of relative position, orientation is different in that it can also be defined as an absolute in terms of the physical space coordinate system. However, regardless of whether a physical or relative measure is used, either approach requires a hybrid variation of the hierarchical model.

3.2.3 Activity Modelling

Ubicomp simulators, particularly those that feature 2D or 3D visualisations of the environment, often provide either user controlled bots [14, 21] or simulation driven bots [59, 105] in the environment. To operate user controlled bots, a human must drive a bot's behaviour, often through standard PC controls. For simulation driven bots, artificial intelligence (AI) or multi-agent based simulations (MABS) are used to generate believable user activity. Re-use of existing simulation engines has generally been acceptable. Examples include the work by Serrano et al. [105] who built Ubik on MASON[1], Jouve et al. [59] who created DiaSim as an extension to Siafu[2], Bylund and Espinoza [20, 21] who created QuakeSim as an extension to QuakeIII Arena[3] and Barton and Vijayaraghavan [13] who built UbiWise on QuakeSim. Although the latter two of these simulators do not implement activity models, they still need to make use of related features such as velocity and collision detection for avatars. The discussion in this section assumes that low-level simulation concerns, such as the physics, navigation and collision detection, are handled by the simulation engine.

Song et al. [109] acknowledge the importance of the spatio-temporal relationship in their modelling approach for tracking the location of complexly moving objects, i.e. those objects that can manifest both random and linear movement patterns. Their model is fundamentally based on a travelling pattern that users are either moving or paused. Moving refers to the time when the user is moving between destinations. Paused refers to the time when the user is at their destination. Song et al. based this on a number of observations including:

- A mobile subscriber mainly switches between two states, stop and move.
- The majority of objects in the real world do not move according to statistical parameters, but instead with intent.

[1]MASON: Multiagent Simulation Toolkit, http://www.cs.gmu.edu/ eclab/projects/mason/ (Accessed 15th November 2010).

[2]Siafu: An Open Source Context Simulator, http://siafusimulator.sourceforge.net/ (Accessed 15th November 2010).

[3]Quake III Arena, http://www.idsoftware.com/games/quake/quake3-arena/ (Accessed 15th November 2010).

- Mobile objects can be grouped by class, which shares common characteristic of maximum speed, e.g. pedestrians, bikes and cars.

- Motion has random and regular parts.

It is the first two items on this list that are most interesting for this thesis, since they are highly relevant to situation or scenario descriptions. The latter two are more relevant to algorithms for sensing technologies which need to efficiently track users in the environment. In implementing activity models for simulation-based ubicomp test-beds and tools, particularly those that address the spatio-temporal relationship, a number of research projects have demonstrated success with high-level models that use 'hotspots' [55] or 'states' [45].

Huebscher and McCann [55] proposed a simulation-based test-bed using a model of contexts to test the context logic of a context-aware application. Their activity model is a high-level description of common activities, e.g. *get out of bed*, *get dressed* and *go out of home*. To create a simulation, the designer associates these activities with hotspots in a 2D map. When an avatar executes an activity, it moves to the associated hotspot and takes a defined time (Δt) to arrive there. This simulated user activity drives context generation suitable for testing context-aware application logic.

García-Valverde et al. [45] make use of the MASON engine which handles the low-level concerns such as simulating velocity and speed in movement. These researchers add a state-based transition model of user behaviour on top of the MASON engine which uses probabilistic models of human behaviour. States are divided between basic states, states with memory and states without memory. The basic state is the primary activity for an agent, i.e. the state that the agent will spend most of their time in during the day. Agents move between the basic state and other states according to a probabilistic model based on the time since a specific state was last entered, i.e. an agent which has just had a break is likely to wait a while before taking another break.

Scourias and Kunz [102] found that when simulating mobile computing environments, fully random mobility models perform poorly compared to models that more accurately represent user activity, such as activity based modelling. This corresponds with Song et al.'s [109] observation that user mobility generally adopts a *move-pause* pattern, and users tend to move with intent, i.e. their behaviour is not random. The study by Scourias and Kunz was motivated by the observation that the profile of mobile computing users had changed since random models were first used. They note that early mobile computing was largely adopted by users who were continuously mobile. However, wider adoption of the technology has meant that it is now used by users who are often stationary, which changes the movement pattern.

Tabak and de Vries [110] note that there is a current trend towards activity based modelling for human behaviour in building usage simulation. Activity based modelling aims to approximate simulations more closely to real user behaviour. Hoes et al. [53], in their results on energy simulations for smart buildings, indicate that user behaviour should be assessed in greater detail, specifically targeting individual buildings and their intended occupants. This fits well with an activity-based modelling approach, because activities are often associated with specific locations in buildings. For example, to post a letter a user needs to go to a mailbox or mailbag, to get a coffee a user needs to go to a coffee machine, or to collect a printout a user needs to go to the printer.

3.2.4 Summary of Modelling Approaches

This section discussed current modelling approaches for situations, environment and activity. Situation modelling is important in order to assist designers and developers to gain greater understanding of the relationships that comprise situations in a ubicomp environment. Situation modelling is seen at the core of work by Henricksen and Indulska [50], and by Clear et al. [28]. Both groups have developed approaches to support specification and exploration of situations. Tools which aim to support exploration of situations, fundamentally need to be able to clearly, formally and logically define situations.

There are two aspects to modelling location information about the physical environment, location modelling and spatial relations modelling. Location information refers to a place and can be represented as either physical coordinates or symbolic information. Spatial relations provide the relationships between places or zones in the environment. These relations are important, not only for describing the static relationships between two or more places, but also the dynamic spatial relationships between users, entities and places.

Finally, activity modelling is an important feature of simulation-based tools for ubicomp, particularly in terms of mobility in the environment. While fully random mobility models perform poorly for simulating mobile computing environments, activity based models which make use of hot-spots or activity states can provide a more realistic simulation. Tabak and de Vries [110] noted the trend towards activity based modelling particularly for building occupant mobility patterns.

3.3 Comparison Framework Definition

The remaining sections of this chapter present a survey of tools and test-beds for action-based ubicomp systems. Prior to commencing the survey, this section defines a comparison framework, which has been created as an instrument for this thesis to compare the contributions made by the tools that are reviewed in the survey. The comparison criteria are based on the findings from the background chapter, which identified a set of challenges related to testing and evaluation of action-based ubicomp systems. These included scalability, heterogeneity, unanticipated situations, appropriateness of system behaviour and traceability of causal factors. Additionally, general design and testing issues such as cost, reusability, extensibility and repeatability also feature in the framework.

The following sections discuss the framework in more detail and identify and define the key criteria. The comparison framework is revisited at the end of the chapter, in a charted format, to summarise the contributions of the tools reviewed in the state of the art survey.

3.3.1 Tool Design

The *Tool Design* category refers to the design and function of each of the reviewed tools. Generally speaking *reusability* and *extensibility* are important traits because they increase the applicability and usefulness of the test environment. *Reusability* of the environment and its resources lowers the amount of investment required, while reusability of complex configurations lowers the amount of time that must be invested. *Extensibility* of the test environment is important since it provides the opportunity to address emerging technologies and is generally beneficial for improving the capabilities of the tool.

Tools that support application testing must provide access to sources of context, either simulated or real. *Context generation* should at a minimum address some aspects of user, entity and location information in order to support thorough ubicomp application testing. Finally, *toolkits* and *testbeds* differentiate the purpose of each tool. Prototyping toolkits facilitate rapid conversion of ideas into interactive system implementations. Test-beds on the other hand provide instruments to facilitate testing prototype implementations.

3.3.2 Deployment Environment

The *Deployment Environment* category refers to the physical deployment environment, simulated or real, used in the design process. *Scalability* refers to the extent of the physical space and the number of devices and users that can be accommodated during testing. *Heterogeneity* refers to diversity both in context sources and in terms of the types of physical environments that the design tool can support. *Configurability* refers to the ease with which devices and physical spaces can be (re-)configured. Environments that are configurable are more flexible and will lend themselves to a wider range of testing. *Fidelity* of the physical environment refers to the visual realism of the physical environment. Finally, *low investment* refers to the resources, man-hours and financial investment required to run a test for each of the approaches.

3.3.3 Testing

The *Testing* category refers to each tool's run-time analysis capabilities. *Repeatability* is useful for resolving problems because it enables the designer to re-run exact sequences of events in order to conduct further inspection. *Scalability* of time refers to the duration of a test that can be conducted. Since ubicomp applications are often required to run continuously, co-existing with users, it is worthwhile to conduct long running tests. Facilitating *spontaneous* or *unanticipated situations* to arise, particularly through user activity, is beneficial because user behaviour is often a primary factor behind unanticipated effects of technology. The two *monitoring* subcategories refer to a tool's ability to assist the designer to monitor all aspects of the environment. *Automatic monitoring* is more desirable for thorough analysis throughout the deployment environment. *Visual monitoring* is beneficial because the designer can use human reasoning to watch for issues that an automatic monitoring tool might not be programmed to detect. Finally, the *spatio-temporal relationship* is included in this category because the temporal relationship only manifests at run-time.

3.3.4 Evaluation

The *Evaluation* category describes issues associated with testing and assessing a ubicomp application. It is also the category that contains most of the identified emerging needs and open issues. *Technical effectiveness* refers to testing a ubicomp deployment to determine that it produces intended outcomes. *Situational appropriateness* refers to investigation of the design of a ubicomp system to determine that it produces outcomes which are appropriate to specific situational instances. *Traceability of causal factors* refers to the ability to trace causal relationships, both physical and digital, in order to determine the factors behind problematic behaviour. *Structured feedback* refers to the need for informative evaluations that can complete iterative prototyping cycles in order to support design refinement.

3.3.5 Summary

This section presented a comparison framework, which has been devised with specific focus on testing action-based ubicomp systems. In real terms, the framework represents a niche area of testing and evaluating ubicomp systems. This is primarily because the complexity and many dimensions of ubicomp systems make it difficult for any single tool to be all-encompassing. Factors which have already been ruled out of scope for the thesis are not included in the framework. The framework is intended to provide criteria to compare the contributions made by each tool in terms of the challenges central to this thesis. It is not intended to provide a ranking of these tools by merit because many of the surveyed tools have additional aspects and features that are outside the scope of this thesis, and so are not addressed by this framework definition.

3.4 Survey of Live Test Environments

Live test environments enable applications to be tested in a real physical deployment that is appropriate to the application being tested. These are high fidelity test environments in which conditions can be approximated with high accuracy while still maintaining some control over tests. It was considered important to include live test environments as part of this survey because testing under these conditions has provided evidence of delivering well designed ubicomp applications, for example in the Labscape project [30].

Three live test environments are reviewed in this section, which provide well-cited examples of both live test-beds and a live test-environment. The Aware Home [62] and Ubiquitous Home [121] provide examples of live test-beds, i.e. ubicomp deployments dedicated to testing and evaluation, and designed for reuse. Labscape [30] on the other hand, provides an example of a live test environment in which a ubicomp system was developed by conducting testing in the target deployment environment.

3.4.1 Aware Home

Description

The Aware Home [62] is a living laboratory project (live test-bed) based at the Georgia Institute of Technology. The Aware Home is a purpose built environment consisting of two identical floors (3 bedrooms, 2 bathrooms) and a basement which focus on a domestic setting. The motivation for including two identical floors is to support parallel prototyping and experimentation, where experiments involve monitoring live-in occupants. Included as part of the design are dropped ceilings, conduit through walls and floors as well as indirect, diffuse lighting with low-sheen floors for working with computer vision technology. The aim of the Aware Home is to investigate sensing technologies that automatically observe inhabitants using a variety of sensing technologies including video, audio, motion and load (weight per floor tile).

The Aware Home has investigated a range of technologies and applications. Location-tracking has been achieved both at room level granularity (*PowerLine Positioning*) and through a self-contained solution for precise indoor location-tracking (*TrackSense*). Applications focus on improving the living conditions and lifestyle of the occupants. For the elderly, applications promote social well-being and independent living such as the Digital Family Portrait which help keep older family members connected with relatives. For parents and families, applications focus on improving efficiency in domestic tasks, family well-being and making home life more enjoyable. The Cook's Collage helps the family 'cook' to resume food preparation after an interruption, e.g. to help with homework or take a phone call.

Analysis

In the literature this project is also referred to as the Aware Home Research Initiative (AHRI) [63] and the Broadband Institute Residential Laboratory [3] which is indicative of the wide range of research challenges that the project aims to address, i.e. ranging from domestic lifestyle to optimal home connectivity. As with all living laboratories the Aware Home is a high fidelity approximation of home life. Access to high fidelity test environments enables these researchers to form deep understanding about how users interact with the environment in order to best inform the design of emerging ubicomp applications. In terms of building size, the Aware Home is well suited to its purpose. The relatively small size of the living quarters would make it easier to perform detailed analysis of activities in the home since it will produce more targeted data sets.

The limitation of the Aware Home is that it is best suited to domestic situations. Researchers interested in outdoor or large-scale deployments, such as those required in office blocks, will need to use an alternative testing approach. Also worthy of mention is the difference in behavioural patterns and environmental control that occupants of a domestic situation have compared to occupants of a shared working environment. Occupants of working environments spend long periods of time there but with less control over how the environment operates. Usually a facility manager controls the policies that dictate how the environment operates and as such it can be expected that the behaviour of occupants in a domestic setting and formal work setting will be different.

The Aware Home is a sophisticated project and test-bed which contributes significantly to deep understanding of user behaviour in domestic environments. However, it has required significant investment, in terms of both time and resources, and despite this, still faces limitations in terms of the diversity of deployment environments that can be approximated using this test-bed. The Aware Home's strengths lie in the fidelity of testing that can be achieved, which in turn means that true unforeseen situations can arise. However, there is a potential limitation in terms of identifying and analysing these situations due to the use of video analysis, and the issues of scalability in testing when using this approach.

3.4.2 Ubiquitous Home

Description

The Ubiquitous Home is another real-life test-bed for home services [121]. It is constructed at the Keihanna Human Info-Communication Research Center building at the National Institute of Information and Communication Technology in Japan. The test-space contains spaces for living, working, cooking, cleaning and sleeping. The technology in the space includes cameras, microphones, floor pressure sensors, infra-red sensors, two RFID systems, accelerometers, speakers and wall-mounted displays. Robots are an additional feature and are treated as appliances by connecting them to the home network so they can be accessed as a home service. Two types of robots are included in the space. The Ubiquitous Computing Home (UCH) is a monitoring robot that observes occupant context but does not interact with occupants. Phyno, on the other hand, is an active or conscious robot with a natural language interface so that occupants can interact with it. The humanoid shape of Phyno and natural language interface is designed so that all family members from age 3 and upwards can interact with Phyno.

The Ubiquitous Home supports a number of applications that have been developed for occupants of the space. These include automatic notification systems, gentle wake-up calls and intelligent entertainment systems. In a paper on daily living [77] the authors propose three objectives for the space that focus on occupants' feelings, behaviour and the usefulness of context-aware systems in sensor based homes. The authors report initial findings collected from experiments in which five families have lived in the Ubiquitous Home for a duration of two weeks each. The key findings were: (1) that occupants are initially very aware of sensors but after three days adapted to living with sensor-driven applications; (2) occupants reported frustration trying to live with Phyno when the robot could not understand their instructions; (3) occupants reported finding it difficult to form a mental classification of Phyno, i.e. human, pet or robot;(4) occupants found it psychologically easier to be watched by Phyno than by an anonymous background entity in the environment.

As well as investigating the applications that support these living environments, the ubiquitous home is conducting research into the management of information using a distributed environment action database system. The apartment has its own Network Operating Center (NOC). The Ubiquitous Home includes spaces in both the ceiling and floor as well as behind-the-scenes corridors that experimenters can access for machine installation and cabling.

Analysis

The Ubiquitous Home is a similar project to the Aware Home, both in terms of the research objectives and the execution of the project. As such, the benefits and limitations of the test-bed are similar to those already discussed for the Aware Home. Physically the two test environments are very similar. The Aware Home is slightly larger including more bedrooms and bathrooms than the Ubiquitous Home. Both test-beds have custom designs to facilitate deployment and installation of sensor technology.

The primary difference between the two projects is the investigation conducted in the Ubiquitous Home into the personal relationship between occupants and the 'home computer'. The active or passive nature of context-aware systems in the future may well impact on the acceptance of these technologies. Occupants that feel helped rather than watched are more likely to be accepting of this presence in their home. The Ubiquitous Home has a valuable contribution to make in this area, including the comparison between passive, active and humanoid robots, and the understanding that is gained through the experiences of occupants living with these robots.

Overall, the Ubiquitous Home provides a significant contribution in terms of gaining deep understanding about how users behave in smart homes, which is similar to the Aware Home. Ubiquitous Home is differentiated by the mother-child metaphor that it has adopted, under which occupants are looked after by robots, affording analysis of the relationship between users and context-aware robots.

3.4.3 A Ubicomp Environment Study: Labscape

Description

Labscape [30] is a custom ubiquitous computing deployment and so differs from the previous examples in that the environment is not reusable for testing different prototypes. The authors, Consolvo et al., are particularly interested in methods of evaluation that can be used on ubicomp deployments to confirm that the technology is not disruptive to a user's natural work patterns. The objective of the project is to use a domain specific live deployment as a medium to demonstrate the strengths and weaknesses of different user study techniques when applied to ubicomp systems. The Labscape application is designed to support work flow in a cell biology laboratory.

The authors make use of Contextual Field Research (CFR) (observing users) and Intensive Interviewing (questioning users) to capture requirements for the system and inform the design of Labscape. More interesting though is the Lag Sequential Analysis (LSA) approach used to assess the deployed system. LSA gathers quantitative data by observing users as they perform routine activities to determine ordering of tasks and relationships between tasks. Data can be recorded using pen and paper, however coded video capture can also be used. Video footage capture must subsequently be coded to represent the activities performed during the experiment and their temporal duration and sequence. This study collected 18 hours of video data from 10 experiments. Forty five hours and thirty minutes were spent training five coders to analyse the footage, and eighty five hours and thirty minutes were spent coding the video.

Analysis

The significant benefit of this style of design are that experiments and assessment are conducted in the user's environment using real activities rather than contrived activities. This results in a tailor made system that is not disruptive and achieves the biologists goal of minimising the number of movements required by reducing the number of movements compared to the previous method of completing their work.

The authors were able to show that a highly authentic study was able to produce a well-designed system that meets the needs of the target user group.

The authors also draw a comparison between Labscape and another live deployment by HP Labs called the CoolTown Exploratorium [64] [42], a ubicomp science museum project. Consolvo et al. note their observation on the impact of different application domains on the implementation of the ubicomp system. Since ubicomp is still relatively immature the research community has yet to learn the impact of differences in terms of scale and purpose when it comes to ubicomp environments. Different applications, cultures and social scenes will impact on the implementation and appropriateness of ubicomp environments. In comparing Labscape to CoolTown the authors note that for Labscape's biologists the value is in the record of the experimental experience, however for CoolTown's visitors the value is in the experience itself. The result of this difference led to the development of an implicit interface for CoolTown and an explicit interface for Labscape. Access to real users and their environment is a significant benefit in determining an appropriate design for user interaction.

A significant disadvantage of the LSA approach is the investment required to extract results, particularly in terms of man hours. This limitation would become problematic for a large scale experiment, especially as the number of users increases. Effectively the amount of coding required is a multiple of the number of users in an experiment, since users can conduct tasks in parallel. Also there is the possibility that in multi-user experiments, users could unwittingly obscure one another in the path of the camera's view leading to incomplete data sets.

There are also some risks associated with this approach. As with many observation methods there is the problem that users may alter their behaviour because they know they are being watched. A second risk is that the final design of the ubicomp system could be highly tailored to the test environment. The risk is that the application could be over tailored for the test user group and might not be easily reused in other similar work environments. The authors mention that, at the time of writing, they were surveying a wider user-group with a view to redesigning the Labscape user interface.

Overall, the Labscape project made a significant contribution in terms of demonstrating the role which user-centric evaluations can play in delivering applications that achieve the goal of being non-disruptive and that compliment the natural work-flow of end-users.

3.4.4 Summary Findings for Live Test Environments

Predominant in the reporting for live test environments, is the deep understanding that these environments afford about how users will behave and interact with ubicomp environments. In the Ubiquitous Home [121], this was explicitly mentioned in reference to the occupants preference for a humanoid presence in the apartment, rather than an all-seeing big-brother style observer, to assist with managing the home. In Labscape [30], the authors relate that their choice of an explicit user interface was based on their user study findings about the biologists' work-flow. This is central to the contribution that comes from live test-beds, which is to inform best practice in the design of ubicomp systems.

A limitation of these environments is the cost and investment required to configure a suitable deployment. It is difficult to determine exact costs for a live test-bed since no two are alike and costs are generally not reported in the literature. However, to demonstrate the scale of cost that can be involved in live testing, a Ubisense[4] UWB Location Research Package consisting of four sensors and ten tags, is listed at a price of £17,500 on the Inition website[5]. Although not all sensor hardware is this expensive, sensors are not the only cost for these environments. The construction or purchase cost of a deployment space is also a significant part of the total expenditure. The house used in the Aware Home project was a custom build for the research.

A second limitation is the time and effort required to configure a sensor network. Although advances are being made to deliver self-configuring networks,

[4]http://www.ubisense.net (accessed 15th November 2010).

[5]http://www.inition.co.uk/inition/pdf/mocap_ubisense_locationresearchpackage.pdf (accessed 17th April 2011).

improvements to the speed and ease at which these networks can be deployed is still an active research problem [41]. Ubisense in particular must be physically mounted in the environment and requires careful calibration to achieve accurate location tracking. Patel et al. [87] report on their experience and note that often several configurations must be tested before a suitable setup can be found. Researchers have also reported other problems that hamper sensor network deployment including, bugs in the operation systems for the sensor network [120] and hardware not working as planned or expected [66].

Designers may also need access to large-scale test environments, for example in the case of deployments in office blocks, hospitals or airports, which are not feasible to recreate in the real world. There is also the challenge of finding consenting users to take part in heavily observed experiments. Consideration needs to be given to ethics and privacy. García-Valverde et al. [46] make the point that real environments can be inappropriate for testing emergency situations. Additionally, in the longer term for mass produced systems, it is unlikely that developers will have access to high fidelity test environments throughout the development cycle.

In summary, although user testing is still considered essential in order to gain deep understanding and produce well-behaved systems, lower fidelity tools have the potential to address some of the limitations of live test-beds, particularly early in the design cycle. Often during early stage testing designers can benefit from being free of the low-level concerns required to deploy and integrate with sensor and actuator networks. The next section discusses the contribution that simulation-based test-beds make to the field.

3.5 Survey of Simulation-Based Test-Beds

Simulation is already an established and accepted tool for testing and evaluation in the field of networks and distributed systems, e.g. J-Sim [114], ns-2[6]

[6]Information Sciences Institute, The University of Southern California. ns-2 The Network Simulator, http://www.isi.edu/nsnam/ns/ (accessed 15th November 2010).

and OPNET[7]. Ubiquitous computing shares common concerns with these fields including the distributed nature of the infrastructure and the issue of mobile users. Simulation offers benefits such as scalability, flexibility, extensibility, re-useability and repeatability to help overcome these challenges. These benefits provide assistance in addressing some of the long standing hurdles for ubiquitous computing developers, i.e. cost and effort. Simulation requires less space and equipment and can allow for easier experimental configuration and manipulation of distributed test environments. Simulation is extensible and configurable, and so can address heterogeneity of context sources and physical environments. Finally, simulation enables early prototypes to be tested without the risk of real world repercussions should the prototype system fail.

This section reviews simulation-based ubicomp test-beds, including descriptions of individual test-beds and analytical discussions of their benefits, limitations and contributions. The test-beds reviewed in this section provide examples of older well-cited tools as well as newer active research that is addressing current open issues.

3.5.1 QuakeSim

Description

QuakeSim, a very early ubicomp simulator, made it possible to test and demonstrate ubicomp applications without a real deployment or requiring test-users to move around a real test environment. The test-bed is a modification of the Quake III Arena computer game. By altering how objects in the game engine behave the authors, Bylund and Espinoza, enabled QuakeSim to generate simulated context data about the position of each user in the environment. Location information in the game engine is converted to longitude, latitude and altitude coordinates which is particularly relevant if the test-environment is a real world place.

[7]OPNET.com, OPNET Technologies, Inc. http://www.opnet.com (accessed 15th November 2010).

QuakeSim retained the interactive nature of the game engine that allows test-users to move about the environment. The example application called GeoNotes makes use of this interactive element for testing and demonstration purposes [38]. GeoNotes is a location enabled messaging or tagging system that lets users annotate their location with additional information using a virtual Post-It note. During testing and demonstration of GeoNotes, the system was configured to receive location information from QuakeSim via the Context Toolkit [8] [94].

In this work, Bylund and Espinoza specified the following requirements for a development and demonstration tool that provides context simulation:

1. It should provide a realistic simulation of real world environments, complete with people and other complex objects (for example buildings, animals and water).
2. Multiple users, represented by avatars, should be able to share an environment and interact with each other.
3. It should be possible to create and equip the simulator with new environments, avatars and objects.
4. Tools for building environments, avatars and objects should be available.
5. Sensors that register information such as the position and altitude of individual users should be simulated.

Analysis

Bylund and Espinoza identified the challenge that distributed context sources posed for developers of context-aware services. They noted the ‘extreme’ difficulty that arises from trying to test and demonstrate location-enabled services that are under development. They recognised that by using a simulation approach, functioning

[8]The Context Toolkit is a prototyping tool which simplifies the task of integrating a ubicomp application with sensor nodes and is discussed later in this chapter.

services could be developed effectively for the real world but which would be supplied with a simulated stream of context information.

The rendering engine of Quake III Arena provides what the authors refer to as 'semi-realistic' surroundings for the test-users during an experiment. In a separate paper about the GeoNotes systems, Espinoza et al. [38] refer to a simulation model of Södermalm in Stockholm. Although the model is not discussed in depth and no screenshots are presented, this reported achievement would indicate that Quake III Arena modelling tools are capable of replicating some degree of realism.

The adoption of longitude, latitude and altitude would indicate that QuakeSim is targeted at outdoor environments since this is more akin to technologies like the Global Positioning System (GPS) which does not work indoors. However, the authors do not explicitly mention anything about this in their publications. The authors also discuss the potential for implementing actuatable devices such as lights and doors in the environment. However, they also discuss support for inputs to the QuakeSim engine as part of their future work, so it is unclear if this was achieved [20, 21].

The requirements presented by Bylund and Espinoza are heavily tailored toward location-enabled mobile services. Interactive, shared test environments are beneficial for addressing spontaneous interaction in ubiquitous computing and as such are important requirements. Extensibility and reusability are also important so that an extensive range of ubicomp applications can be tested. However the specificity of singling out altitude and position as important pieces of context could be considered unnecessarily limiting as a requirement for a ubicomp demonstration tool.

QuakeSim made a significant contribution in terms of facilitating multi-user testing of distributed, mobile systems in relatively large-scale physical environments, particularly in comparison to the live test-beds already discussed.

3.5.2 UbiWise

Description

Following directly on from QuakeSim, UbiWise emerged [13, 14]. Barton and Vijayaraghavan learned of QuakeSim at a UbiTools workshop[9]and decided to extend it further. UbiWise modified the Quake III Arena code so that weapons were replaced with virtual handheld devices to enable testing of hardware and low-level software, e.g. device protocols. This modification of the weapons system is referred to as UbiSim. UbiSim provides a first person 3D view of the simulated environment and its devices, including the device that is held by the test-user.

The second component of UbiWise is the Wireless Infrastructure Simulation Environment (WISE). WISE uses Java's Swing Toolkit to provide a 2D view of the control surface on a physical device. Users interact with the device using mouse clicks to press buttons on the device in the 2D view. UbiWise is the integration of UbiSim with WISE. UbiSim provides the 3D view of the physical world. WISE provides the separate 2D device view.

One of the test-cases discussed for UbiWise includes three device types, a digital camera, a wireless PDA and a set of digital picture frames. The test-case uses these devices to demonstrate how UbiWise can investigate both resource discovery and data transfer. In the UbiSim view of the environment, the user sees three picture frames physically mounted on a virtual wall. The users also sees a partial view of the device that their avatar is carrying. In the WISE view, the user sees the digital screens of all shared devices and all of their own devices. UbiWise updates the views across all versions of these devices.

The authors describe their efforts to accommodate the perspective of three types of stakeholder in the design of UbiWise. The test-user can interact with a simulation by using the devices in WISE and changing their location in UbiSim. The researcher can configure a simulation. The developer can extend UbiWise to include additional devices and protocols.

[9]UbiTools 2002, Atlanta, Georgia

In this work, Barton and Vijayaraghavan specified requirements for a ubiquitous computing simulator, as summarised below:

1. Experiment with new sensors, whether hand held or environmental, without building and deploying them.
2. Aggregate device functions without connecting real devices.
3. Develop new service and device discovery protocols without implementing these protocols in multiple mobile devices.
4. Observe how users might react to new devices and services before we can fully realise them.
5. Explore the integration of hand held devices and Internet services without the huge cost and large teams needed to realise this integration.
6. There should be a balance between fidelity and simplicity but the environment provided must be close enough to reality to validate ideas.

Analysis

At the time that Ubiwise was developed 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 provided a simulation-based test-bed to break this circle of dependence. Appropriately, the test-cases focus on resource discovery and data transfer, issues identified earlier in this chapter as part of the general middleware problem. UbiWise is unique in the field in its combination of 3D and 2D simulation to support device control and low-level protocol development. Environment simulation in ubicomp test-beds is often reserved for testing aspects of these systems that focus on mobility issues.

The requirements set out in this work focus on issues relating to testing and managing hardware and low-level resources in the environment. The most general requirements that Barton and Vijayaraghavan identified included extensibility, minimising effort and cost for researchers, and repeatability, however even some of these were described in device specific terms.

Overall, UbiWise made a significant contribution to the field at the time by enabling services to be tested using devices and protocols that were available only in virtual form, i.e. prior to realisation of the real world counterparts.

3.5.3 UbiREAL

Description

UbiREAL is a ubicomp environment simulator or smartspace simulator as the authors refer to it, developed by Nishikawa et al. [80]. UbiREAL incorporates simulators for physical quantities, network simulation and a 3D visualisation mechanism. This enables the simulator to facilitate deployment of virtual devices in a 3D space, simulation of communication between devices from MAC level to application level and to simulate changes in the physical state of the environment, e.g. temperature change. The focus of UbiREAL's evaluations are invisible physical quantities such as temperature, humidity, electricity and radio as well as audible quantities such as acoustic volume.

The most relevant aspect of UbiREAL with respect to this thesis, is the approach towards systematic testing of the correctness of a ubicomp system. The authors have approached this challenge by providing a formal model for system specification. A smartspace U is defined as a tuple $U = (R, D)$, where R is the set of rooms and D is the set of devices. This is similar to the set-based location models mentioned earlier in this chapter. The service specification is defined using a set of rules $AP = l_1, ..., l_l$, where each rule l is comprised of a condition and an action. The example service specification that the authors provide is an air-conditioning system. If a room exists at a certain temperature that is defined as hot by a user, then switch on the air-conditioning unit.

To test an implementation of the system specification, a tester must either manually specify routes for mobile objects through the smartspace, or define a test specification which contains a sequence of test cases or scenarios that are automatically executed. These sequences are tested against a set of propositions, P, which if they hold true, infers that the system is correct. The authors fundamentally aim to determine correctness of a system specification by testing its implementation to determine if the following conditions hold:

1. For every rule $l = (c, a) \in AP$ and every state s of U, if condition c holds for state s, then action a is executed.
2. Every proposition $p \in P$ holds.

Due to the large state spaces that often must be tested for ubicomp systems, the authors advocate testing either using a predefined set of samples or at predefined time intervals.

The authors present performance results for sequence execution and simulation visualisation. The authors do not appear to have published example output or results from a defined set of propositions. The authors also do not appear to have published performance results for test execution.

Analysis

The major contribution of this work is an approach for formal specification of smartspace applications, along with an approach for testing correctness of these specifications. Additional strengths include the extensive simulation capabilities that feature in the work, both in terms of physical environment phenomena and network connectivity between devices. Also worth noting is that UbiREAL is one of few tools which attempt to address a full prototyping cycle, from system specification through to testing.

However, the use of a set-based location model is a limiting factor in terms of the reasoning that can be conducted about the spatial relations of the ubicomp environment. Moreover, UbiREAL as a tool suffers from a lack of generality. The service specification must be defined in specific terms. The examples provided by the authors refer to individual users and devices. It is also difficult to assess the expressiveness of the tool because the user, device and room models are not formally documented in order to illustrate their attributes.

In terms of testing, a limitation of the work, which the authors acknowledge, is that they can only validate correctness for the inputs that are known to have been tested. This is because of the challenges surrounding the scale of testing required in order to address the full state space. The authors advocate eliminating test-cases that are unlikely to occur, however adopting an approach such as this does not fit well for testers who may wish to investigate unanticipated and outlier test cases. Additionally, because the inputs are tested against a system specification, the tests are effectively locked into the assumptions made in the system design. This leaves less room and flexibility to investigate the key challenges of unanticipated behaviour and unexpected situations.

The authors refer to the use of temporal logic, such as Computational Tree Logic, which they describe as being used to intuitively determine the state to which a system will transit. However, the details on how this is used to investigate correctness are insufficient to draw conclusions about the contribution of this particular aspect of the work. The authors also claim that inconsistencies in system specifications should be detected by UbiREAL, however the description of this is short and lacking detail. It appears that detection of these inconsistencies is through observation and visual monitoring of the simulated smart space by the tester.

Overall, the contribution of this work is notable since few tools have addressed the issue of correctness and validation for ubiquitous computing systems and smart spaces.

3.5.4 DiaSuite and DiaSim

Description

DiaSim [59, 19, 24] is part of DiaSuite [60, 61], a tool suite which aims to address the full development life cycle for pervasive computing systems, including implementation, testing and live deployment. Of the tools in the suite, DiaSim is the most relevant to this thesis. DiaSim is a parameterized simulator which means it is flexible in terms of the domains that it can be used to test. The parameters are drawn from the underlying layers of the tool suite, part of which is a taxonomy defined by a domain expert using a custom design language, and which a custom compiler uses to automatically produce a domain-specific programming framework. This underlying support means that testing can be transparent for a pervasive computing application prototype, regardless of whether it is being tested on a live, hybrid or simulated environment.

DiaSim simulates a 2D representation of a pervasive computing environment, comprised of polygon shaped regions which model walls and areas. The authors mention a hierarchical environment model, however presentation of this model was not found in the papers reviewed [60, 61, 59, 19, 24]. The environment can be populated with sensors and actuators. Sensors are activated by stimuli, e.g. temperature, light. Stimuli are simulated using simple models which approximate their intensity by assuming they are uniform in a region.

The simulation engine is connected to a rendering engine which produces a 2D visualisation of the environment. The authors also refer to this as a monitoring engine, although currently it is only capable of visual monitoring and visual debugging. It does however accept live user interaction in order to modify a simulation on the fly, e.g. to add a stimulus. DiaSim produces a simulation log and supports browsing the raw data in this log to help determine the events leading to an error. However, automatic error detection is not mentioned as a feature of DiaSim.

Analysis

The DiaSuite toolset has a number of sophisticated features including transparent testing across simulated, hybrid and live deployment environments. DiaSim appears to be a newer addition to this body of work and as such is less mature, however the goals and direction of the work closely align with current open issues, in particular thorough testing and debugging of pervasive computing applications.

The limitations of DiaSim appear to lie with the monitoring engine which has mainly been reported as a rendering or visualisation engine. The monitoring engine produces the 2D view of the environment that allows the tester to visually monitor how scenarios unfold in the environment. The strength of the rendering engine is that it can use logs to replay problematic sequences of events. It is unclear, however, how the tester is notified of problematic behaviour. There is no indication that any automatic notification of problems has been implemented.

This seems to be the major limiting factor of DiaSim, particularly because the authors are targeting large, complex environments where many users and devices are interacting. The authors note simulations which feature 200 users and discuss scenarios which could scale to 900 users. The authors also discuss a scenario which involves a three story building with an area of 13,500m^2. The visual monitoring and debugging approach, or even manually tracing raw data in log files, have the potential to place significant cognitive load on the tester for certain simulation configurations.

Overall, DiaSim makes a significant contribution by enabling ubicomp systems to be specified at a very high level and subsequently allowing the code base to be tested in both the real and simulated test environments, without requiring any modifications.

3.5.5 Ubik and AmISim

Description

AmISim is a multi-agent based simulator (MABS) which García-Valverde et al. [46, 45] and Serrano et al. [105, 104] have created with a view to addressing 'all features' relating to AmI, and integrating them into a common framework. The high-level aim is to create a design and development framework capable of testing, validating and verifying AmI environments. The authors describe their intention to make use of existing forensic analysis techniques for MABS [103] to support debugging in AmI applications. The AmISim architecture takes a layered approach to support separation between four key concerns (in this order from the bottom layer up): the environment model, the agent (user) model, the context model and the adaptation (AmI service/application) model. The tiered architectural approach is used to support hybrid testing, potentially allowing the simulated environment to be replaced with a real test-environment.

The environment and agent models are implemented using MASON[10], a Java-based fast discrete event multi-agent simulation library core. The authors use the name Ubik to refer to the combination of AmI-specific models and MASON. Ubik uses a 2D model of physical space which can be comprised of rooms, windows, doors and stairs, as well as sensors and actuators which are active throughout an experiment. The sensors listed by the authors in an example of an emergency fire evacuation include RFID, sensors for fire intensity and sensors to detect open doors and windows [46]. Agents are visualised as 2D circles with directional indicators.

Individual users are modelled as agents, each with their own physical and behavioural characteristics, e.g. role in an organisation, velocity and level of panic. This agent model was discussed as the beginning of this chapter as an example of an activity model. In the paper by Serrano et al. [105], the authors describe AmISim's use of a state-based model to represent activities, and probabilistic transitions to determine agents movement between activities, for an everyday scenario. In the

[10]MASON, http://www.cs.gmu.edu/ eclab/projects/mason/ (accessed 15th November 2010).

paper by García-Valverde et al. [46], the authors describe their use of random behaviour in agents combined with attractors to specific events based on proximity, for emergency scenario simulations.

The context model makes use of the Open Context Platform (OCP) [79], a middleware which provides support for managing contextual information. OCP provides the functionality of a context management system, including gathering, merging, interpreting, reasoning and storing information. OCP works by classifying situations of interest and triggering relevant information and actions as appropriate.

The purpose of the adaptation model is to simulate the services and applications offered to users of an AmI environment. The model is comprised of two parts, the context model API and the adaptation logic. The API provides an interface between OCP and the AmI application logic, and is responsible for extracting and formatting contextual information. The adaptation logic is responsible for generating appropriate actuation instructions which are sent to the environment model via OCP. The authors list a number of techniques which this model can make use of including SWRL[11], reinforcement learning and machine learning.

Incorporated alongside MASON and Ubik is an Analyzer which provides the features for investigating simulation results. The details provided for this are brief, however the authors mention that any agent method starting with 'get' or model method that returns a property can be stored in a relational database (RDB), e.g. creation/destruction of agents and agent interaction. The authors mention that events of interest must be registered, however the details for this are not covered.

Analysis

This work appears to be relatively new in the field with the earliest publication on AmISim, found during this literature survey, dating from 2009. There are few experimental results presented in the papers reviewed here [46, 45, 105, 104], so it is difficult to ascertain the exact capabilities of AmISim. The authors repeatedly refer

[11]SWRL: A Semantic Web Rule Language Combining OWL and RuleML http://www.w3.org/Submission/SWRL/ (accessed 15th November 2010).

to testing, validation, verification, debugging and forensic analysis, however without detailed descriptions of how these have been realised or the results a designer can expect to generate using the tool, it is difficult to determine which of these objectives the researchers have achieved. Further publications, that will clearly define the extent and limitations of AmISim's models are still needed in order to make a solid judgement on this work. However, the research challenge that has been identified is very worthwhile, and the proposed objective can be expected to make a significant contribution to the fields of both ubicomp and AmI.

Results presented by the authors include tests to scale their simulations, which have achieved a building of 200 floors and 200,000 agents, however there is no indication whether this is a 3D model or a 2D model similar to that illustrated in their papers. A gap in their current research is the application of AmISim to real world scenarios, which the authors acknowledge. The authors do not discuss the fidelity of the building evacuation simulations, compared to the actual behaviour of real users, which is of significant importance because the authors state that their objective is to validate applications for the real-world.

The primary benefits which seem to emerge from this work are as follows. The authors discuss the configurability and extensibility of the technical architecture to address varying scales and types of both physical environment, devices and users. The authors consider the querying power of an RDB for performing data-mining on logged simulation events to be a considerable contribution, referring to it as a 'breakthrough'. It is true that, in general, debugging, monitoring and analysis of the effectiveness of ubicomp and AmI systems is currently poorly addressed for these fields, however it is difficult to determine how great is the contribution of AmISim in this respect without more details on the capabilities of the simulator and *Analyzer*.

Overall, AmISim currently provides benefits to the designer through the use of a sophisticated MABS approach, matched with powerful context management services, for extremely large scale simulations of physical environments. The authors also mention the potential for hybrid testing, for which context management services will provide additional benefit in terms of easily switching an application between real and simulated test environments.

3.5.6 Summary Findings for Simulation-Based Test-Beds

The overwhelming benefit of using simulation based test-beds is the level of scale that can be achieved in terms of users, devices, physical deployment space and duration of test simulations. For automated simulations, i.e. those that use agents or artificial intelligence, the tester can also achieve scale in terms of long running experiments. Simulation based test-beds are useful for testing early stage concepts in a low-risk, low-investment environment at scales that are often infeasible in the real world. Additionally, experiments can be more easily repeated, or replayed in the case of DiaSim, since they are not affected by the noise that is an issue in live deployments.

However, simulations are limited in the fidelity that they can achieve, both in terms of the physical environment model and sensed context generation. Often test-beds must make a tradeoff between maintaining simplicity which is beneficial for early stage testing and achieving higher fidelity that is required for testing to ensure robustness in the final application. As such, it is not expected that they will offer a replacement or alternative to the deep understanding gained through live testing.

The focus of this thesis is on action-based ubicomp systems, and so requirements drawn from seminal simulation test-beds reflect this. The motivating factor for the creation of UbiWise was testing devices and protocols, with less emphasis on testing ubicomp systems or applications. As a result, UbiWise is less relevant to this work and only a subset of UbiWise's requirements are used to inform the design in this thesis. These are combined with the more relevant requirements from QuakeSim. Their influence on this thesis will be discussed in more detail in Chapter 4, however in short, the relevant requirements include the issues of heterogeneity, user-centric evaluation, cost-effectiveness and environment realism.

UbiWise and QuakeSim used simulation for the purpose of testing during exploratory research. However, a trend among newer tools, is a move towards simulation for the purpose of testing and error detection, e.g. UbiREAL's approach to testing system specification or AmISim's proposal for forensic analysis of ubicomp system behaviour. This adds to the requirements set out by UbiWise and QuakeSim to include the need for structured testing and evaluation of ubicomp systems.

3.6 Survey of Prototyping Tools

Incremental and iterative development (IID) has repeatedly been identified as beneficial to successful development of complex systems [67,32,81,48], a trait which both Tang et al [111] and Davies et al. [34] note as relevant to ubicomp systems. An evolutionary prototyping approach, a form of IID, produces the appearance of stability, incrementally building on tested prototype stages but without investing the resources of a full deployment [48]. Rapid prototyping and iterative design cycles offer a method to support exploratory research, a contribution that has repeatedly been mentioned as beneficial to ubiquitous computing, for example by Li et al. [69], Huebscher and McCann [55], Davies et al. [34] and Tang et al. [111].

Li et al. [69] identify three key benefits for ubicomp as: (i) lowering the barrier to entry; (ii) speeding up design cycles; (iii) making it easier to capture feedback early in the design cycle. Davies et al. [34] provide a similar discussion, referring to lower costs and the possibility to explore and trial ideas. Huebscher and McCann [55] also make reference to incremental and iterative approaches (IID), by mentioning the capability to incrementally build and test context logic in a controlled environment.

Tang et al. [111] provide a thorough discussion on the issue of user-centric, iterative rapid-prototyping for ubicomp. In their paper, the authors provide a diagrammatical representation, which illustrates the iterative cycle between design, prototyping and evaluation, in relation to the many forms of prototyping. The authors illustrate how, during successive prototyping cycles, a developer can progress through the tools including sketches, paper prototypes, interactive prototypes, computer prototypes and production ready prototypes. Tang et al.'s diagram is reproduced in Figure 3.2 due to its significant relevance to this thesis.

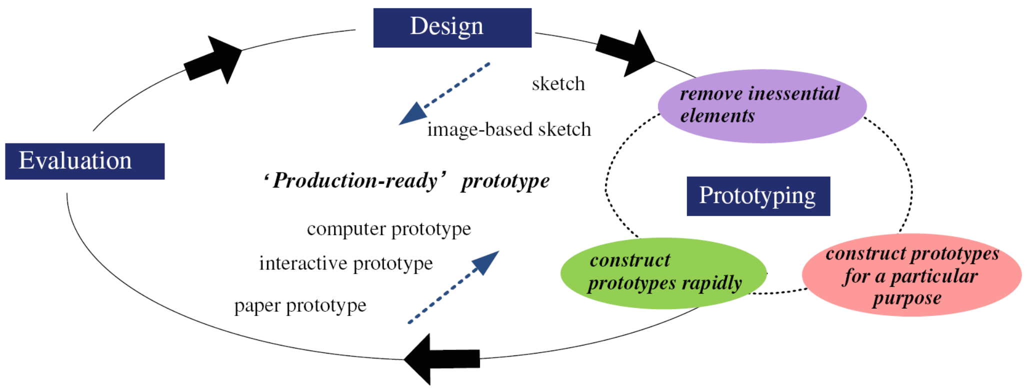

Design
sketch
image-based sketch
remove inessential elements
Evaluation
'Production-ready' prototype
Prototyping
computer prototype
interactive prototype
paper prototype
construct prototypes rapidly
construct prototypes for a particular purpose

Figure 3.2: Tang et al.'s [111] diagram of user centred iterative prototyping approaches for ubicomp.

Prototyping tools are included as part of this survey because part of the motivation of this thesis is to provide support for the evaluation stage of an iterative prototyping cycle. This section discusses the contributions that have been made by three examples of prototyping tools and techniques in the field. The Context Toolkit [94] provides an example of one of the earlier and well-cited prototyping toolkits which produces computer prototypes. Topiary [69] provides a newer and well-cited example of a prototyping tool which produces interactive implementations that can be evaluated using Wizard of Oz studies. Finally, PALPlates [22] is an example of how paper prototyping can be used for distributed applications.

3.6.1 The Context Toolkit

Description

One of the earliest prototyping tools is the *Context Toolkit* [94] which emerged in the late 1990's. The Context Toolkit aimed to make it easier to build context-enabled applications. The design of the toolkit drew inspiration from the success of widgets used in the process of rapidly prototyping graphical user interfaces (GUIs). In the same way that GUI widgets hide underlying functionality from the UI designer, context widgets were designed to be reusable and to manage the specifics of underlying hardware and middleware for the ubicomp developer. For example, an *IdentityPresence* widget would manage detection of user presence and provide call-backs to the context-aware application when updates to the location status occurred.

This effectively allows a context-enabled application to subscribe to certain types of context. A drag-and-drop approach enabled designers to rapidly pull together context sources to support the prototype application. In this paper the authors list six widget types: IdentityPresence, Activity, NamePresence, PhoneUse, MachineUse and GroupURLPresence. The three applications discussed which were developed using this tool focus on shared displays for information presentation.

In later work by Bylund and Espinoza [21] the QuakeSim was integrated as an additional context widget for the Context Toolkit. This enabled the Context Toolkit to supply simulated location information and real context information to prototype applications. Additionally, by integrating QuakeSim as a context widget, designers are able to easily switch between real and simulated context data simply by changing the widget used during a test.

Analysis

In their paper on this work the authors, Salber et al., identified some of the key challenges faced by developers working with ubiquitous computing systems. Context aware systems acquire their input from what were considered at the time unconventional sources, i.e. sensors. Sensor data often needs to be abstracted, i.e. into context, in order to be beneficial to the context aware system. Additionally, context is sourced from multiple distributed, heterogeneous and dynamic sources. The Context Toolkit was developed to remove some of the labour involved in this process, and to enable the developer to focus on the application rather than the underlying technology.

Salber et al.'s approach promoted separation of concerns between context sensing and application semantics. This offered support for rapid prototyping of context-aware applications and was among the earliest tool-kits to address this need. This has benefits in the early design of an application when designers wish to focus on the semantics of the application and not re-implementing the underlying information sources. Low-investment prototyping is also beneficial for exploratory research and quickly testing ideas, an issue which is repeatedly mentioned in the literature as central to ubicomp development due to the relative immaturity of the field [69, 33, 111].

The reason for including the Context Toolkit in this review is its close relationship with the simulation test-beds already discussed, as illustrated in Figure 3.3. The integration of the Context Toolkit with a simulation context source immediately makes it more accessible for testing early stage prototypes. As previously discussed, Bylund and Espinoza developed QuakeSim [21, 20] to test and demonstrate context aware services using a 3D simulator to generate location coordinate information. The Context Toolkit on the other hand encapsulated real context information, from sensors, in context widgets. In the integration of these tools, QuakeSim acted as a source of simulated location context to the Context Toolkit. The combination of these tools provided a more powerful test-bed than each had been independently. Developers could perform quick, lower fidelity tests using simulated context. However, using the context supply from real-sensors, developers had the facility to perform usability testing and full-scale deployment tests.

Other similar tools have been developed since the Context Toolkit, in particular VisualRDK [117]. This latter tool is not included here due to space constraints and because progress on its planned integration with a 3D simulation tool, for the purpose of testing, was not found in the current literature. However, it is worthy of note because it produces two types of prototypes, a production-ready version and a debugging version. A developer can examine the control flows of the non-distributed, debugging version on a local development machine to inspect program flow and data structures. This supports investigation of the technical implementation of an application, however VisualRDK does not appear to address the more user-centric issue of appropriateness.

The Context Toolkit is a seminal tool, which provided a major contribution to

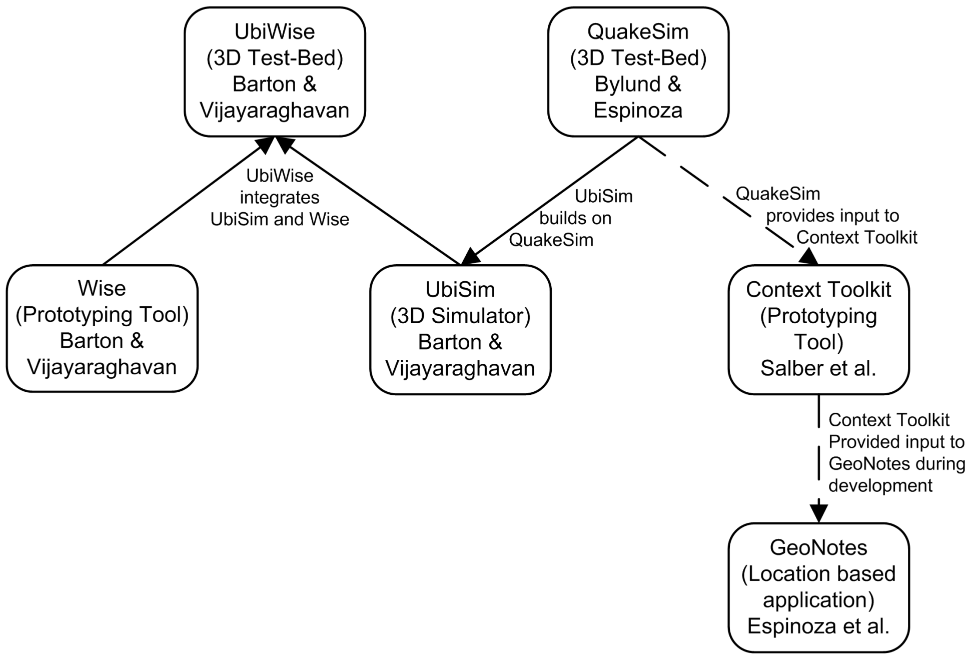


Figure 3.3: Relationship between Context-Toolkit, QuakeSim and UbiWise.

the field by enabling rapid high-level prototyping of ubicomp applications. The extension by Bylund and Espinoza added further value to the Context Toolkit by extending the range of context sources available to prototype context aware applications, as well as supporting an early form of hybrid ubicomp test-bed.

3.6.2 Topiary

Description

Topiary [69] is a graphical tool for rapid prototyping of both ubiquitous computing test scenarios and ubiquitous computing 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 simulation, called the *Active Map*, where users are moved around in much the same style 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 uses a Wizard of Oz approach in which the tester follows the test-user in the real world. By observing the user and the situations the user encounters, the Wizard (tester) can play the role of the prototype system and generates the behaviour of the real system.

Active Map can be used to model both indoor and outdoor environments. Designers use it to model the spatial relationships between people, places and things, e.g. Alice is near Bob or Bob is in the Gym. Topiary supports two basic relationships in both static and dynamic capacities: presence (in, out, enters, exits) and proximity (near, far, moves near, moves far). The Scenario Producer tool allows the designer to generate a collection of location relationships in order to define a scenario. The final component of Topiary is the Test Workspace which manages both the End-User UI and the Wizard UI. Topiary tests can run off location information generated from defined scenarios or using live location data where available. To demonstrate live testing, Topiary has been integrated with PlaceLab [56], a live test-bed at Massachusetts Institute of Technology (MIT), similar to the Aware Home and Ubiquitous Home which were already discussed.

PlaceLab is not discussed in depth in this review because many of its research objectives overlap with the previously reviewed live test-beds. PlaceLab also places significant emphasis on capturing large reusable datasets, with less emphasis on testing and experimentation directly in the living space, the latter of which is considered more relevant to this thesis.

Analysis

Topiary focuses on providing three main benefits to interaction designers. Firstly, it lowers the investment required to develop an operational prototype that can be tested with real users. Secondly, this naturally speeds up the development cycle meaning that early iterative cycles can be completed more quickly. Finally, Topiary allows user feedback to be captured while prototypes are still in a low-risk and highly flexible state. These contributions are all in keeping with findings from Abowd's Classroom2000 project [1]. In this discussion, Abowd asserts that ubicomp prototypes should be developed quickly, tested and explored quickly, so that lessons can be learned quickly.

Topiary's creators claim that the tool is the first for prototyping location enhanced applications. Their main contribution in this respect is that they facilitate interaction designers to create high-level designs rather than customise or 'cobble together' existing technology. Topiary provides a tool that allows designers to visually relate to an overview of the environment including the relationships between people, places and things.

Topiary is particularly smart in making the most of its 2D view. It could be expected that in using a 2D view it would be difficult to prototype scenarios for more than one floor of a building. However Topiary allows for composite scenarios and provides the facility to define scenarios on different pages, which are effectively separate bitmaps. Although not explicitly mentioned in their paper, this should assist in overcoming the limitation faced by many 2D simulators when attempting to model 3D environments.

The obvious limitation of Topiary is the scale of testing. The Wizard of Oz style approach used during the testing phase is limited by the observation capabilities of the Wizard. For location enabled applications the Wizard needs to follow an individual test-user in the environment which means that one Wizard is required per test-user. This is a significant investment of man hours for large scale testing both in terms of the experiments run as well as the training required for new Wizards.

The second limitation is the spatial relations model, which is described as basic by the authors. Bandini et al. defined containment, proximity and orientation as the minimal set of spatial relations in their Commonsense Spatial Reasoning Model [11]. Topiary only satisfies two of these, containment and proximity. It is not expected that extending to include orientation would be a difficult task, however Topiary only features spatial and to a certain degree temporal context; additional forms of context are desirable in a ubicomp prototyping tool.

Overall, Topiary addresses its objectives well and makes a strong contribution to early stage prototyping of location tracking systems for interaction designers.

3.6.3 Paper Prototypes: PALplates

Description

Paper prototyping is a low-investment, user-interface design and evaluation technique often used for developing graphical user interfaces (GUI). The approach involves drawing all proposed views of the GUI and its dialog boxes on sheets of paper. Similar to the Wizard of Oz approach, the tester must play the part of the system by manually changing the interface in response to the test-users actions.

In this work with PALplates the authors, Carter and Mankoff [22], apply the paper prototyping approach to a ubicomp system featuring a distributed interface. The PALplaces system is designed to support office workers in everyday activity by providing relevant information at appropriate locations using public displays distributed throughout the office building. Office workers were also able to interact with these displays to send notification messages to colleagues, the example used in the paper is to notify someone in maintenance that a printer is almost out of paper.

The system was implemented by combining notebooks and post-it notes. The notebooks played the role of the mounted display units, also called the PALplates. The post-it notes played the role of messages or notifications in the network and as such they could be transferred to other PALplates. Researchers played the role of the network and their responsibility was to respond to user requests by scheduling meetings, fetching supplies and delivering messages. A network update ran once each day.

Analysis

The obvious significant benefit of the paper prototype approach is the low investment required before a trial of an idea. Designers can turn very early stage concepts into a functioning prototype. Although the underlying technology has not been implemented, test-users are still able to experience how the application might operate in a real deployment. The tester benefits by observing how users interact with the prototype and from the facility to capture feedback from the test-users very early in the design cycle. Carter and Mankoff observed a link between location and usage patterns for individual PALplates. For example, the authors give an example of unique functionality that occurred on the kitchen PALplate, that lead to social interaction through discussion on the PALplate.

In both the PALplates project and in similar work by Liu et al. [71], scalability of paper prototypes was identified as a potential limitation when prototyping ubicomp systems. As both the distribution of the sensor network and the quantity of data being transferred between devices increase, the manpower needed to emulate system behaviour can become substantial and difficult to maintain.

Overall, the paper prototyping experiment provided a useful insight to inform further development of the PALplates system, specifically the impact of location on how PALplates were used. For example, the kitchen PALplate generated social interaction through discussion, whereas the hallway PALplate was used to book the meeting room. More generally, paper prototyping offers the potential to inform about how users will interact with ubicomp systems.

3.6.4 Summary Findings for Prototyping Approaches

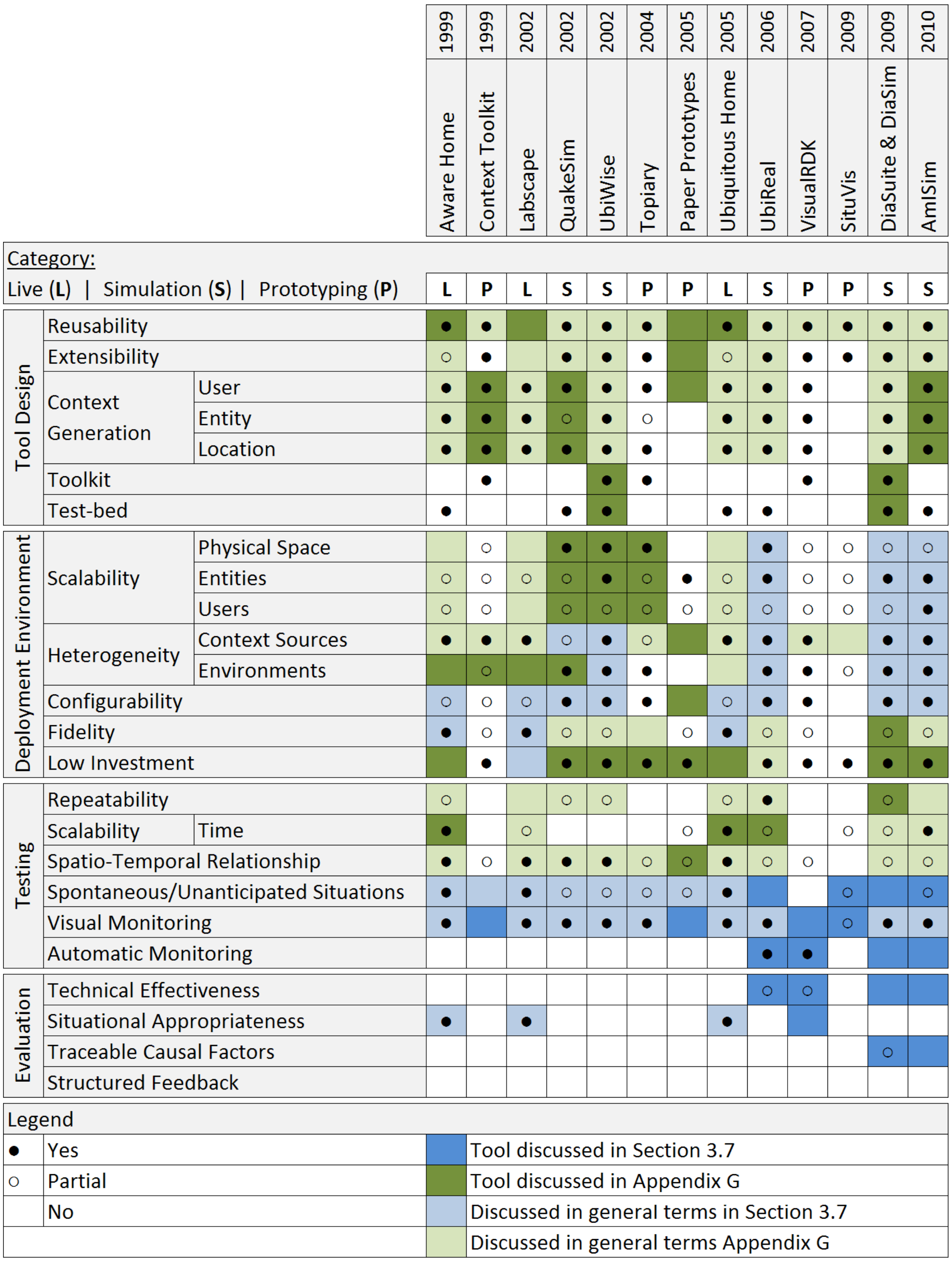
Prototyping tools can provide significant benefit in terms of freeing the developer from the low-level concerns required during a live deployment. The tools presented in this section also demonstrate that these low-fidelity systems still enable the tester to gain some understanding about how users will interact with a ubicomp system. Carter et al. [22] were able to gain insight into how location impacts device usage. Li et al. [69] demonstrated how they could use a Wizard of Oz approach for testing mobile systems in the wild. The Context Toolkit on the other hand made use of QuakeSim to test the GeoNotes system in a virtual version of a real world location using accurate simulated coordinates so that researchers could test a mobile system without leaving their desk.

Rapid-prototyping is particularly beneficial in the early stages of design because, as Li et al. [69] point out, it lowers the barrier to entry. Evolutionary or iterative prototyping allows complex systems to be built in small validated steps, with each iteration providing feedback to inform design refinement. Although there have been many examples of prototyping tools for ubicomp systems [94, 69, 22], for the most part these have been lacking structured testing and evaluation approaches, which are necessary to complete iterative cycles. VisualRDK [117] goes some way to addressing this issue by providing an approach for debugging systems, however this is limited to investigating technical effectiveness of an implementation and is less suited to assessing user-centric issues. In the longer term the field will benefit from structured methods of providing testing and evaluation feedback, which can inform design refinement, as part of an iterative prototyping cycle.

3.7 Comparison Framework Analysis

This section revisits the comparison framework defined earlier in this chapter to provide a charted comparison of the tools reviewed in this chapter. The charted comparison framework, shown in Figure 3.4, illustrates the four main sections of the framework, which include tool design, deployment environment, testing and evaluation. Due to the density of information presented in the charted comparison framework, it is not possible to discuss each tool individually for all characteristics defined in the framework.

For brevity, this section presents a condensed discussion of the comparison framework analysis for the surveyed tools. The full analysis is included for reference in Appendix G. The shading of squares in Figure 3.4 indicates whether tools were discussed in specific terms, general terms or not at all, for each of the comparison criteria in the framework. Darker squares illustrate where tools have been singled out for discussion in relation to a specific comparison criterion, in either Section 3.7 or in Appendix G. Paler squares illustrate that a tool has been discussed in general terms, e.g. broadly in terms of the category to which it belongs.



			Aware Home	Context Toolkit	Labscape	QuakeSim	UbiWise	Topiary	Paper Prototypes	Ubiquitous Home	UbiReal	VisualRDK	SituVis	DiaSuite & DiaSim	AmISim
			1999	1999	2002	2002	2002	2004	2005	2005	2006	2007	2009	2009	2010
Category: Live (**L**) | Simulation (**S**) | Prototyping (**P**)			**L**	**P**	**L**	**S**	**S**	**P**	**P**	**L**	**S**	**P**	**P**	**S**	**S**
Tool Design	Reusability		●	●		●	●	●		●	●	●	●	●	●
	Extensibility		○	●		●	●	●		○	●	●	●	●	●
	Context Generation	User	●	●	●	●	●	●		●	●	●		●	●
		Entity	●	●	●	○	●	○		●	●	●		●	●
		Location	●	●	●	●	●	●		●	●	●		●	●
	Toolkit			●			●	●				●		●	
	Test-bed		●			●	●			●	●			●	●
Deployment Environment	Scalability	Physical Space		○		●	●	●			●	○	○	○	○
		Entities	○	○	○	○	●	○	●	○	●	○	○	●	●
		Users	○	○		○	○	○	○	○	○	○	○	○	●
	Heterogeneity	Context Sources	●	●	●	○	●	○		●	●	●		●	●
		Environments		○		●	●	●			●	●	○	●	●
	Configurability		○	○	○	●	●	●		○	●	●		●	●
	Fidelity		●	○	●	○	○		○	●	○	○		○	○
	Low Investment			●		●	●	●	●		●	●	●	●	●
Testing	Repeatability		○			○	○			○	●			○	
	Scalability	Time	●		○				○	●	○		○	○	●
	Spatio-Temporal Relationship		●	○	●	●	●	○	○	●	○	○		○	○
	Spontaneous/Unanticipated Situations		●		●	○	○	○	○	●			○		○
	Visual Monitoring		●		●	●	●	●		●	●		○	●	●
	Automatic Monitoring										●	●			
Evaluation	Technical Effectiveness										○	○			
	Situational Appropriateness		●		●					●					
	Traceable Causal Factors													○	
	Structured Feedback														

Legend			
●	Yes	🟦	Tool discussed in Section 3.7
○	Partial	🟩	Tool discussed in Appendix G
	No	▧	Discussed in general terms in Section 3.7
		▨	Discussed in general terms Appendix G

Figure 3.4: Comparison Framework Chart.

3.7.1 Comparison Framework Criteria Analysis

The following subsections focus on discussing individual comparison criteria from that framework, particularly focusing on criteria that are of central interest in this thesis as well as being relevant to open issues in the field.

Visual Monitoring

Many of the reviewed simulation and prototyping tools provide visualisations of the deployment environment. These allow the designer to visually monitor the effect of the ubicomp system in the physical environment. The exceptions are Paper Prototypes, Context Toolkit and VisualRDK [117], which all have in common that their prototypes run in live physical buildings. Live environments can achieve visual monitoring by using cameras to record activity in the environment. The disadvantage of this is the significant man hours that must be invested in order to process the recordings and extract meaningful data. SituVis is listed as partially providing visual monitoring for testing purposes because it provides intuitive views of the relationships and potential conflicts between situations.

Automatic Monitoring

Less has been achieved in terms of supporting designers with tools that automatically monitor ubicomp system behaviour. Of the surveyed tools, UbiREAL has made a notable contribution by enabling designers to define a system specification and check the execution of this logic in a simulated environment. VisualRDK makes a notable contribution by enabling developers to debug specially compiled localised versions of pervasive computing prototypes. DiaSim and AmISim, both more recent work, state that their ambitions also lie in this area, however neither have produced publications with evidence of significant advances in this area yet.

Spontaneous/Unanticipated Situations

Live environments perform best in this category, primarily because the complexity of the human condition and spontaneous action cannot currently be truly simulated. However, models which simulate key aspects of human behaviour can provide some of the dynamism of human behaviour, for example the models in AmISim [46]. Simulation or prototyping tools driven by real users also provide some opportunity for unanticipated situations to arise, particularly if the test-user is a target end-user or at least is independent of the design process. DiaSim is not listed as featuring spontaneous user activity because the details available on its user model are sparse. UbiREAL is also not listed as supporting spontaneous user activity because it only allows a tester to define static procedural routes for a simulated bot. SituVis [27] is listed as partially supporting discovery of unanticipated situations, because although SituVis does not support run-time testing, it provides an opportunity to uncover unexpected situational conflicts, which might otherwise not be foreseen.

Technical Effectiveness

UbiREAL provides a significant contribution in this subcategory by enabling designers to define a formal service specification which can in turn be tested for correctness in a simulated deployment environment. DiaSim is also making progress in this category with its parameterized approach to system specification, which in the longer term is expected to support debugging for ubiquitous computing systems. The researchers behind AmISim aim to provide forensic analysis as a method of testing, debugging and analysing ubiquitous computing systems, however it appears that this remains part of their future work.

Situational Appropriateness

Live test-beds provide researchers with the opportunity to perform relatively long-term testing and evaluation of ubicomp systems. The close approximation of these test-beds to real living conditions provides the opportunity for test-users to uncover unintended consequences of system behaviour. These conditions also provide the opportunity to uncover unusual or outlier situations that arise infrequently. The strength of this is the deep understanding and learning that is gained about these environments. However, the level of investment required for these test-beds is prohibitive to using them for mass development. Designers can benefit from tools which can evaluate the appropriateness of system behaviour using a more affordable approach.

Traceable Causal Factors

Among the reviewed tools, DiaSim comes closest to providing traceability of events leading to situations of interest. This is achieved through recorded log files, which can be replayed or manually examined, in order to identify the sequences of events leading to a specific situation. The limitation of DiaSim is that the tool does not provide automatic monitoring. The designer must visually monitor the simulated environment in order to identify problems, which has the potential to place significant cognitive load on a designer and may not be as thorough as an automated monitoring approach.

3.7.2 Comparison Framework Analysis Findings

The comparison framework analysis uncovered that there is little to differentiate individual tools in the categories of Tool Design or Deployment Environment. Generally, simulation-based test-beds tend to perform better in terms of scalability, configurability and to a certain extent heterogeneity, simply because both devices and the physical environment can be modified more easily and at relatively little cost. Live environments are realistically only configurable in terms of the devices and network deployment. Live test-beds also tend to have the highest costs, particularly those that are custom built and include the expense of the physical building.

However, live environments produce the highest fidelity experiments because test-users can interact with the system in an authentic setting. This provides the opportunity to gain deep understanding about how these environments will operate, and make informed decisions about best practice for ubicomp design. At the same time though, the development cycle for individual systems also needs more accessible tools that can be used regularly, repeatedly, and with lower investment. Simulation-based test-beds have the necessary characteristics to be well-suited for frequent testing, particularly in the early-mid stages of the development cycle.

In terms of testing ubiquitous computing systems, the comparison framework analysis uncovered that few tools address the issue of automatically monitoring system behaviour at run-time. UbiREAL and VisualRDK provide the only examples among the reviewed tools. UbiREAL does this by adopting a situation-based service specification and requirements testing approach. The aim of UbiREAL is to support end-users to prototype applications, for their own domestic environments, and visually demonstrate how a prototype will behave for specific situations at run-time. VisualRDK makes a contribution to automatic monitoring by supporting developers to debug specially compiled, localised versions of their code. Although both of these tools make contributions to this issue, both also place more emphasis on technical effectiveness, than on the more user-centric issue of appropriateness.

It was also uncovered that no tool provides automatic monitoring while also engineering the opportunity for spontaneous or unanticipated situations to arise. UbiREAL comes closest with its situation-based approach to system specification. However, UbiREAL's approach is limited due to a lack of generality and over-specification of test cases, which make it difficult to test for the unanticipated.

Evaluation has generally been poorly addressed by early stage and low investment tools. It is probable that the trend for lower investment tools is only emerging now because earlier tools needed to first focus on supporting designers to more easily prototype ubicomp systems. The Context Toolkit was seminal in this respect, and more recently Topiary has also made a significant contribution in terms of enabling developers to rapidly create ubicomp systems.

Live environments have been demonstrated to provide reliable results in terms of evaluation, however they have not yet been demonstrated as a practical solution for mass testing of systems on a large scale. Limitations in terms of cost and lack of flexibility, make them inaccessible to many projects. UbiREAL and VisualRDK provide some degree of evaluation for technical effectiveness. UbiREAL enables a tester to determine that for specific inputs the system behaves as intended. VisualRDK supports the tester to examine the program flow and data structures of a ubicomp system at run-time.

In recent years, a trend has emerged of simulation and prototyping that is placing more emphasis on addressing the issue of testing and evaluation of ubicomp systems earlier in the development cycle and requiring lower investment. Many of these tools are looking to more formal, systematic and automated approaches to testing. UbiREAL made a notable contribution by enabling end-users to define a formal, testable, situation-based system specification. VisualRDK supports a designer to debug the program flow and data structures of a ubicomp system at run-time, using a specially compiled localised version of the system. This addresses technical effectiveness but not the more user-centric issue of situational appropriateness, the latter of which is more central to the focus of this thesis.

DiaSim provides an element of traceability by enabling the designer to manually examine log files which can uncover sequences of events leading to specific situations. This is more than most tools report, however it is still insufficient for thorough examination of a system's behaviour, particularly because DiaSim does not support automatic monitoring for problem detection. The large quantities of data that ubicomp systems can produce has the potential to make this a tedious and laborious task. AmISim aims to support forensic analysis as a method of testing, debugging and analysing ubicomp systems, however results for this are yet to be published.

3.8 Chapter Conclusions and Summary

This chapter began by presenting a survey of modelling approaches appropriate for supporting a situation-based testing approach. The chapter also presented a survey of ubicomp tools that have made contributions to the area of testing and evaluating action-based ubicomp systems. A comparison framework, based on the challenges identified in Chapter 2, was devised as an instrument to compare the surveyed tools.

Based on the review of tools, there is promise in simulation-based approaches and in the use of iterative and incremental development cycles. It is apparent however, that although much work has been conducted towards addressing the prototyping phase of an iterative development cycle, there has been less emphasis on the testing and evaluation stages required to complete each cycle, particularly for lower investment testing. There is a need for methods and approaches to testing that address the novel issues introduced by ubiquitous computing. In particular gaps and open issues remain surrounding (i) monitoring spatio-temporal relationships throughout large scale environments, (ii) assessing the appropriateness of situational outcomes that result from the implementation of a particular design, (iii) traceability of causal factors across the physical-digital boundary, and (iv) provision of structured feedback to the designer to inform subsequent iterations in the design cycle.

3.8.1 Summary of Findings

This chapter began by looking at the state of the art in modelling that could be used to support a situation-based testing approach. Models by Henricksen and Indulska [51], Nishikawa et al. [80] and more recently Clear et al. [27], provide a basis for a situation specification model. The section on modelling also look at models to address the design considerations that were identified and defined in the background chapter as user activity and the physical environment.

Tabak and de Vries [110] noted a current trend towards activity based modelling, particularly for building usage simulation. Scourias and Kunz [102] found that activity based models performed better than fully random mobility models, which they attribute to changes that they have observed in the types of end-users making use of mobile systems. Activity-based models can provide more targeted testing, and recent examples can be found in work by García-Valverde [45] and Tabak and de Vries [110]. In terms of physical environment modelling, specifically spatial relations, hierarchical models were found to naturally lend themselves to modelling the containment relationship of the internal structure of buildings. Hybrid-hierarchical models have the advantage of accommodating additional information about physical location information, and other spatial relations. The combination of a situation model, with a hybrid hierarchical model and an activity-based model, has the potential to support situation specification as well as addressing the design considerations defined in Chapter 2.

The comparison framework analysis uncovered that there was little to differentiate individual tools, of similar classifications, within the categories of Tool Design and Deployment Environment. However, generally simulation-based tools are better in terms of extensibility, scalability, heterogeneity, configurability and lower investment. Live environments are generally better for high fidelity, accurate approximations of physical deployments.

In terms of the Testing and Evaluation categories of the comparison framework, it can be seen that evaluation is an issue mostly addressed by live environments and by some of the more recent research on design tools. Live environments have generally provided the most detailed and thorough user-centric evaluations of ubicomp systems, particularly for late-stage prototypes. However, the limitations of scale and significant investment required, make live testing impractical and inaccessible to many research projects, particularly for early-stage testing or exploratory research. There is a need for alternative tools and approaches.

It was found that there is no tool that combines automatic monitoring with the opportunity for spontaneous and unanticipated situations to arise. This requires a degree of generality in test specification, not seen in the current tools. Generality is required because it is difficult to script specific tests and scenarios for unanticipated outcomes.

Overall, the charted comparison framework illustrates that many aspects of testing and evaluation remain open issues for ubicomp systems. In particular, the issue of realising approaches for evaluating user-centric issues such as appropriateness of system behaviour, through a structured traceable approaches, that are lower-investment and so suited to early stage testing.

Chapter 4

Design

4.1 Introduction

The hypothesis of this thesis is that a *situation-based testing approach* could identify inappropriate situations in a simulated deployment environment. Inappropriate situations were characterised as those situations that arise due to the effects of unwanted behaviour in a ubicomp deployment. Fahrmair et al. [39] stated that unwanted behaviour is not a technical problem in a ubicomp system, but instead arises due to a specific combination of user activity and environment, social and task context.

The state of the art review showed that while much emphasis has been placed on the design and prototyping phases of iterative development cycles for ubicomp systems, less has been achieved for the evaluation phase. Due to the scale of ubicomp deployments, both in terms of run-time duration and physical space, designers will benefit from tools that can automatically monitor and test the global state of a deployment environment. While UbiREAL [80] and VisualRDK [117] have gone some way to addressing evaluation of technical effectiveness, little evidence was found of tools which can test more user-centric issues, automatically and systematically, with the exception of high investment live testing.

This chapter presents the requirements and design for the situation-based testing approach, which are informed by the findings from the state of the art. The chapter concludes with an analysis of the design that relates back to the research objectives and technical requirements, and that discusses the novelty of the approach.

4.2 Situation-Based Testing Approach

The situation-based testing approach focuses on assessing the state of a physical deployment, as opposed to examining execution of a ubicomp system. As such it must identify inappropriate situations independent of system behaviour, user activity, and environment configuration and layout. The state of the art review illustrated the benefits of simulation in terms of scalability, configurability, extensibility and reusability. However, simulation provides an additional advantage for the situation-based testing approach because the state of a simulated ubicomp deployment can be captured, including both physical and digital artefacts. This provides the opportunity to test the summative effect of user and system behaviour in the environment, in order to assess the appropriateness of system behaviour, based on actual effects and outcomes.

In Chapter 1, research objective **O2** set out the objective to formalise the situation-based testing approach in a test specification framework. The InSitu Model Framework has been designed to address this and is presented in this chapter. Research objective **O3** set out the objective to define a systematic process of testing that supports a feedback loop. The InSitu Test Process has been designed to address this and is presented at the end of this chapter. The Model Framework and Test Process together define the situation-based testing approach. These are defined independent of technology and so can be realised in various implementations.

Research objective **O4** set out the objective to realise the situation-based testing approach in a simulation-based implementation. This is addressed by a toolset implementation. An overview of the InSitu Toolset is presented in this chapter, but a detailed discussion on the Toolset is deferred to Chapter 5. The Toolset provides the mechanism to evaluate the underlying test approach.

4.3 Influences from State of the Art

This section discusses the influences from the state of the art that were instrumental in realising the InSitu Model Framework, Test Process and Toolset. These influences led to a set of technical requirements that must be satisfied in order to realise the situation-based testing approach. For the purpose of discussion, these are divided into two sections, the first focuses on the Model Framework, and the second set focuses on the Test Process and Toolset.

4.3.1 Influences on the InSitu Model Framework

At the core of all of the situation-based approaches discussed in the state the art review, is a situation specification model. Situations are abstractions of context and can be defined through aggregation of contextual information. Situation specification enables situations to be formally defined through logical and spatial relationships between elements of the physical deployment space. Examples can be found in UbiREAL [80], CML [50] and SituVis [27].

Of these examples, UbiREAL provides the only example of a situation-based testing approach, however a limitation of the UbiREAL approach is over-specification of test-cases. This is prohibitive for testing that aims to uncover unanticipated outcomes arising in the deployment environment. Situation specification must be generalised in order to accommodate testing for the unanticipated.

This leads to the following requirement:

> **Requirement R1:** A situation specification model must feature in the Model Framework, which supports generalised definitions, and which can specify the logical and spatial relationships between elements in the deployment environment.

Requirement **R1** addresses the issue of writing situation specifications that are capable of testing the unanticipated and unexpected. However, it is also necessary that unanticipated situations have the opportunity to arise. Chalmers [107] suggests that model-driven simulations of users and computers can assist with the examination of ubicomp systems, particularly for emerging or unanticipated situations. Tabak and de Vries [110] are more specific in noting the current trend in activity based modelling for building usage simulation.

Activity-based approaches reflect the scheduling of activities in time and space. Routes taken through a building are a result of the activities that occupants need or want to perform, and the locations associated with those activities. Activity based modelling provides a more accurate approximation of human behaviour than purely random models [102], without requiring complete fidelity and so can be a lower investment approach. The inclusion of an activity-based model addresses one of the *design considerations* defined in the background chapter, user activity. An activity-based model should be included in the model framework to drive simulations that approximate the natural work-flows or activities of occupants.

The second design consideration is the physical environment and in particular its spatial relations. Spatial relationships are central to situation specification and so a situation-based testing approach must be able to reason about this aspect of a building. This is primarily because often the structure of the physical environment cannot be changed, except in terms of the operational use of the environment, and so a system must be designed to behave appropriately for the target environment.

Hierarchical models best represent the natural containment relationship in the environment, and there are many examples in the literature [25, 95]. Hybrid hierarchical models are more flexible by accommodating extra spatial relationships, as well as physical location coordinates, and there are also a number of examples in the literature [57, 10].

Proximity regularly appears in the literature in relation to sensing technologies, due to the relevance of a user's immediate surroundings. Henricksen and Indulska [50] include it in CML, along with containment, as a key spatial relationship for situation specification. Bandini et al. [10] identify three key spatial relations for ubiquitous computing as containment, proximity and orientation.

These findings lead to the following requirement:

> **Requirement R2:** An activity model must be included to drive simulation of user activity; a model of the built environment must be included to promote relevancy of testing by representing the target deployment environment.

The situation specification model, from requirement **R1**, must be able to test situation specifications using an up to date view of the state of the environment. A benefit of simulation noted in the last section is the ability to capture both simulated state and sensed information. An environment state model is necessary that can represent a unified view of this information, and which includes the main elements of situations and the spatial relationships in the environment.

This leads to the following requirement:

> **Requirement R3:** A testable model of environment state must be defined and maintained.

4.3.2 Influences on the InSitu Test Process and Toolset

Simulation-based ubicomp test-beds generally support prototype systems to remain independent from the test-bed platform. To enable this, test-beds need to provide sources of context information to the prototype under test. Papers on the seminal simulation-based ubicomp test-beds, UbiWise [13, 14] and QuakeSim [21], set out requirements for these type of test-beds.

The requirements from QuakeSim focus on the need for realistic, configurable and extensible deployment environment simulations, featuring multi-user interactions and simulated sensors to generate location information. The requirements from UbiWise focus on user-centric assessment of ubicomp technology prior to a product-ready prototype, testing at lower costs, and providing sufficiently realistic simulated deployment environments.

Consolidating the requirements put forward by QuakeSim and UbiWise, the following requirement is put forward for a simulation-based ubicomp testbed:

Requirement R4: A simulation based test-bed for ubicomp should be configurable, extensible, reusable and scalable, with relatively low investment and should support heterogeneous context generation.

Core to the situation-based testing approach is the necessity to maintain separation between sensed context and environment state. A ubicomp system should only receive a sensed view of the environment based on the information generated by sensors embedded in the deployment environment. However, it is necessary for the situation-based testing approach that a state view of the environment is available, which represents the actual conditions that exist in the deployment environment.

Simulated sensed context should only be supplied to the ubicomp system in order to drive system behaviour. State information is supplied only to the testing engine and makes it possible to test the effect of ubicomp system behaviour by examining the situations that arise in the deployment environment.

This leads to the following requirement:

Requirement R5: Separation between simulated sensed context and environment state information must be maintained.

The challenge in assessing the effectiveness and appropriateness of system behaviour in ubiquitous computing environments lies in monitoring the distributed nature of their effect. Therefore, monitoring ubicomp deployments must be performed at a global level, i.e. across the whole space. Examples of visual monitoring tools were found in the literature, in which the designer can observe simulations unfold and watch for problems to arise, e.g. DiaSim [59]. However, visual monitoring is not particularly suitable for thorough testing. Testing is a tedious computational task, better suited to a computer than to a human. Video recording of live test-beds provides another option for monitoring ubicomp deployments, however the time investment required to process the recordings is prohibitive to both rapid iteration and low investment testing.

Also, as already noted in this section, in order to test for unanticipated and unexpected outcomes a monitoring engine must be able to apply abstract test definitions. Test definitions should be represented using a situation specification model so they can be tested against the state of the environment.

This leads to the following requirement:

> **Requirement R6:** A monitoring engine must be able to test environment state, which can be distributed in the ubicomp space, to automatically identify instances of inappropriate situations, without prior knowledge of the sequences and scenarios that will arise at run-time.

Structured feedback is particularly useful as part of an iterative prototyping cycle in which successive iterations produce incremental, testable changes, as discussed by Tang et al. [111]. Evaluation feedback, particularly for a prototyping cycle, is largely an open issue for ubiquitous computing and one that is actively being researched, e.g. the DiaSim [59] and AmISim [46] projects.

On identification of situational instances, the designer needs to be able to trace the causal factors. Generalisation of Weber et al.'s definition [116] resulted in a definition of traceability as: *attributing events and actions to that which caused them.* According to this definition, traceability should support identification of both the user activity and system activity that can effect change in the state of the environment.

This can in turn be provided as structured feedback. Although the user should not be held accountable for inappropriate situations arising in the environment, user activity that is instrumental to these outcomes should be used to inform the design refinement process.

This leads to the following requirement:

> **Requirement R7:** Structured feedback must be generated, which can be analysed and traced for the purpose of investigating the causal factors leading to specific situations.

4.4 Overview of the InSitu Design

The *InSitu Model Framework* provides a formal model of test specification for the situation-based testing approach, in line with research objective **O2**. The *InSitu Test Process* defines a systematic method of testing, which includes a feedback loop to support evaluation and design refinement, in line with research objective **O3**. Together the InSitu Model Framework and InSitu Test Process define the situation-based testing approach. A toolset implementation of the framework and process is required to realise the situation-based testing approach.

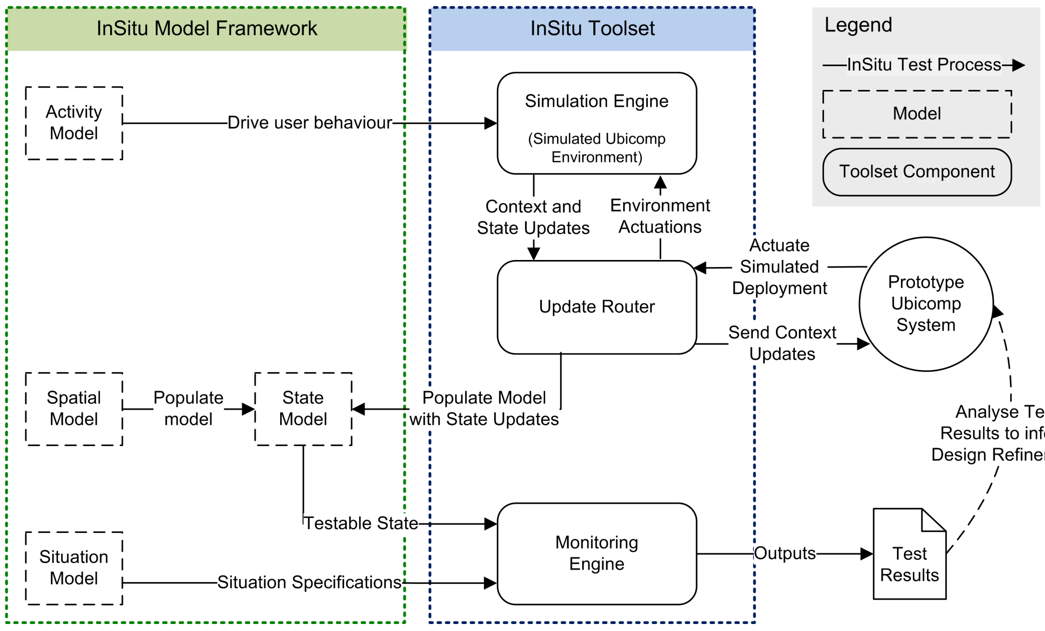


Figure 4.1: Overview of InSitu Model Framework, Test Process and Toolset.

Figure 4.1 illustrates an overview of the InSitu Model Framework, InSitu Test Process, and the relationships between the models, process and *InSitu Toolset* components. The InSitu Model Framework and Toolset are each bounded by dashed lines, and each are respectively labelled. The InSitu Test Process is defined by the arrows between the models and components in the diagram.

The *Activity Model* is used to drive user behaviour in a simulated deployment. The *Simulation Engine* generates sensed and state update messages to drive testing. The *Spatial Model* and state update messages combine to form the *State Model*, which is a testable representation of the state of a simulated deployment environment.

The *Situation Model* and State Model are reconciled in the *Monitoring Engine* to examine the state of the environment for inappropriate situations. Test results are analysed to assess the extent of occurrences of inappropriate situations during a simulation, and to inform design refinement of the prototype ubicomp system.

The remaining sections of this chapter describe the design of individual models in the InSitu Model Framework and the design of the InSitu Test Process.

4.5 InSitu Model Framework

The InSitu Model Framework provides the test specification framework necessary to address objective **O2**. The *Activity Model* and *Spatial Model* have been designed to describe the design considerations, which include user behaviour and layout of the physical deployment environment. The inclusion of these models in the Model Framework improves the relevance of tests to the intended deployment configuration. The *Situation Model* supports situation specification that is generalised and so can be used to test unanticipated outcomes at run-time. Finally, the *State Model* is used to maintain an up-to-date view of the environment state which can be tested at run-time using the Situation Model.

4.5.1 Spatial Model

Becker and Dürr's [15] survey of modelling approaches discussed the suitability of hybrid hierarchical models for representing the structure of buildings. This section discusses a hybrid hierarchical model that provides the facility to model both physical and operational aspects of a building. Common elements of hierarchical models are floors and rooms, as was previously discussed in relation to Satoh's model [95]. For many 2D simulators which look at only one floor at a time this can be sufficient. However, modelling multi-floor buildings requires both horizontal and vertical zones.

A zone called *vertical-spaces* was introduced to model stair-wells which are important

Zones		
Zone	Type	Description
Building	Physical	Root of the model for the internal layout of a building.
Vertical-Space	Physical	Zone which spans more than one floor, e.g. stairwells and atriums.
Floor	Physical	Floor in a building.
Room	Physical	Room in a building.
Space	Operational	Zone which contains two or more rooms.
Area	Operational	Zone which does not contain any other zones.

Table 4.1: Zones in the Environment Model

because they act as the connection between floors. Vertical-spaces are also useful for modelling atriums, a popular feature in modern buildings, which provide natural light through an open column in the centre of a building.

Another limitation of many spatial models is that they focus on describing only physical aspects of a building. However, operational usage of a building often results in semi-permanent zones, many of which do not have tangible boundaries but can be perceived by occupants and ubicomp systems. For example, in modern offices large work spaces are often shared by many occupants, each individually occupying a unit of space approximately the size of a desk. To accommodate this, an operational zone was introduced to the hierarchical model called an *area*. The building, floor, room and vertical-space types all share in common that they are all bounded by walls. An area is different in this respect. Areas are not physically defined by walls, instead they are small symbolic zones, that are located inside rooms.

The final type of zone added to the hierarchy is called a *space*. *Spaces* are larger than *areas* and cover two or more other zones in the hierarchical model. Spaces are particularly aimed at addressing the issue of internal rooms or inter-connected rooms. The way in which these rooms are used on a day-to-day basis is also an

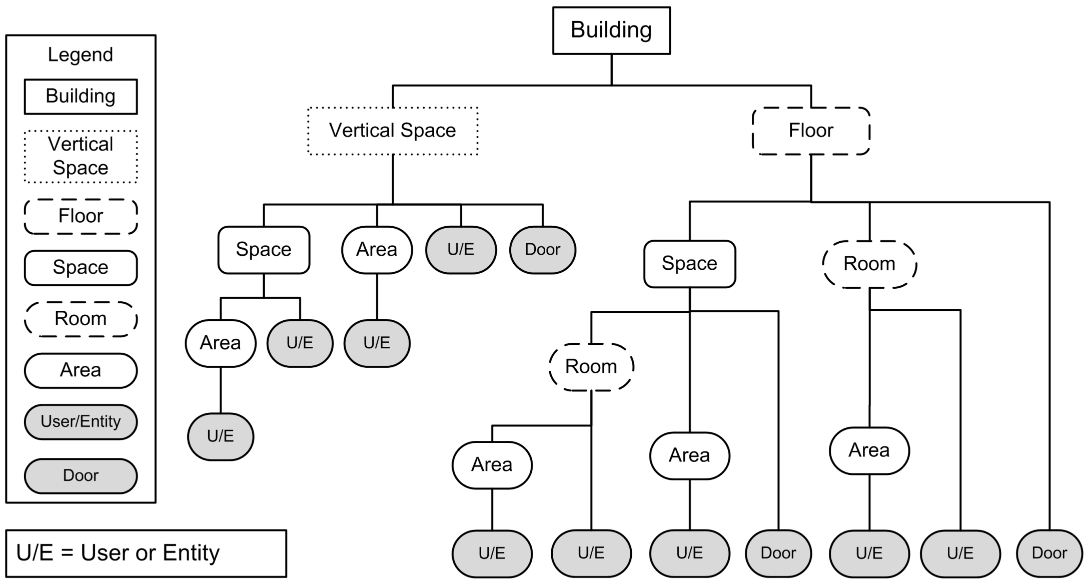


Figure 4.2: Environment Model: Hierarchy of Zones.

operational feature of a building. In day to day operation, if the doors between interconnected rooms are generally left open then the rooms effectively operate as one space, subject to social norms observed by the occupants.

Table 4.1 lists the zones included in the Spatial Model. These are the building, vertical-spaces, floors, rooms, spaces and areas. An intuitive illustration of the hierarchical relationships between these zones is illustrated in Figure 4.2. For example, a building node can directly contain vertical spaces or floors. These in turn act as containers for other nodes, such as spaces, rooms, areas, doors, users and entities. Vertical-spaces do not contain rooms, but they can contain interesting areas, e.g. an exit point to a floor on a stairwell, as well as containing entities, e.g. a lift. Floors can contain rooms, and can also contain areas. Figure 4.3 defines, more formally, the restrictions on the containment relationship between zones.

Proximity is addressed using granularities of zones in the environment model hierarchy. Bandini et al. [10, 11] define proximity as:

> *Two places are said to be proximal if it is possible to go from one to the other without passing through another place.*

Adjusting this slightly for users, entities and zones, this model defines proximity as:

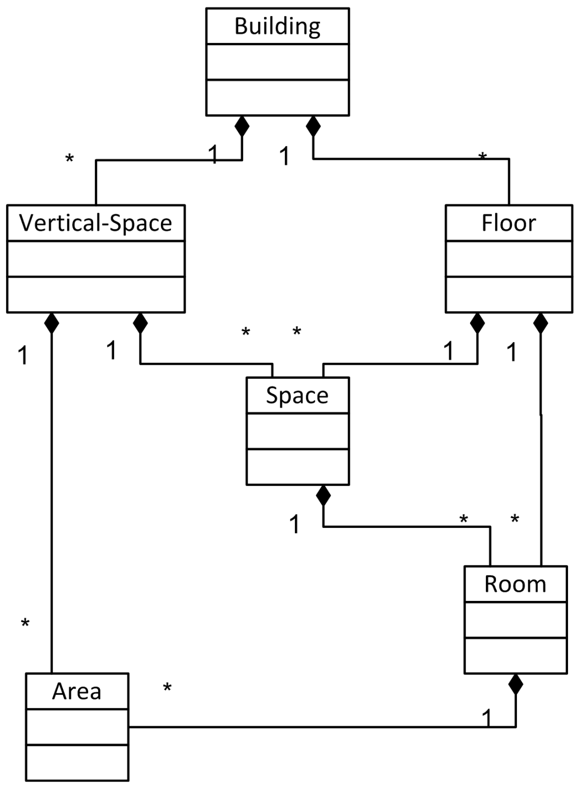


Figure 4.3: Restrictions on containment relationship for zones.

Entities and users are said to be proximal if it is possible to go from one to the other without passing through another zone.

Becker and Dürr [15] noted this point similarly, in their discussion on hierarchical models. It is possible to compare the distance between positions by considering the smallest zones in the hierarchy. The hierarchical spatial model presented in this section becomes a hybrid hierarchical model when supplemented with information from the State Model. The resulting combination features the spatial relations of containment, proximity, orientation, as well as physical coordinates from the environment. The State Model also provides attributes about zones to differentiate between different zone types, for example a coffee-room, office or corridor. At run-time the model exists in the zone states of the State Model.

4.5.2 State Model

This section describes the State Model, which is used to represent the state of the environment at a particular point in time. The UML diagram in Figure 4.4 illustrates the attributes of the main elements of the model, which include users, entities, doors and zones. Doors are included as a distinct element in this model, because although they could be classified as entities, they have the unique position of directly impacting the routes that users travel through a building, and as such impact how spatio-temporal relationships evolve. Table 4.2 provides definitions for all of the attributes used in the State Model.

Attributes of the State Model can be static, dynamic or semi-permanent. Static attributes are unchanging, e.g. user ID, and so only need to be populated with data once. Dynamic attributes are subject to change with time, e.g. user location, and so need to be updatable. For simulation purposes, initial values should be provided for all dynamic elements in order to reflect the starting configuration for the simulated test-environment, and so that false positives do not occur during testing. Semi-permanent attributes are static for long durations and do not change during a test simulation, however, in the real-world these attributes would change over time, e.g. a user's position of employment.

Figure 4.4 also illustrates the supported relationships between elements of the model. Elements can be contained by zones, elements can be near to other elements, and zones can be contained by other zones. The restrictions on the containment relationship between zones was already discussed.

User State

User state information describes an individual user at a given point in time. The state of a user is defined in terms of a unique identifier (ID), the user's role, a symbolic location, physical coordinates, orientation, the user's field of view and a timestamp for the information. The value for symbolic location is the zone ID for the zone that a user is in at a given time.

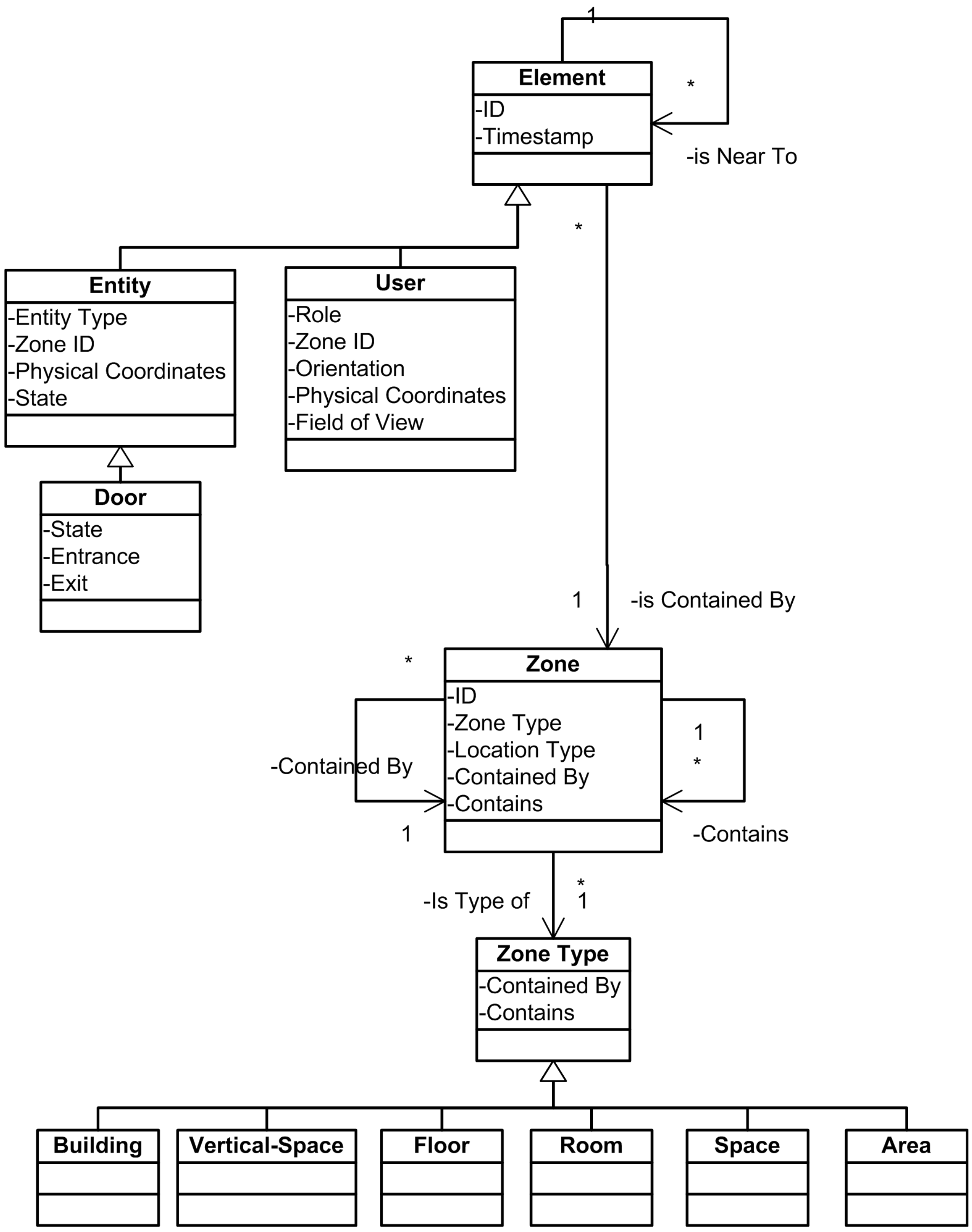


Figure 4.4: Environment State Model including Spatial Zones.

Environment State Model Attributes		
Attribute	Cat	Description
Contained By	S	The parent location of a zone.
Contains	S	The child locations of a zone.
Entity Type	S	Type of entity e.g. motion sensor, proximity sensor, door, light.
Entrance	S	Describes a directional entry point to a zone.
Exit	S	Describes a directional exit point from a zone.
Field of View	D	Set of users and entities in a user's field of view.
ID	S	A unique identifier.
Location Type	SP	The purpose of a zone, e.g. coffee room, meeting room, office.
Orientation	D	Orientation of a user.
Physical Coordinates	D	Coordinate location information.
Role	SP	Role of a user in the building.
State	D	State of an entity or door, e.g. on/off, open/closed.
Timestamp	D	The time at which the last update was received and recorded for this user.
Zone ID	D	Symbolic location identifier.
Zone Type	S	Spatial element type, e.g. area, room or floor.

Table 4.2: Environment State Model Attributes. Categories (Cat): S = Static, D = Dynamic, SP = Semi-Permanent.

Entity State

Entities can represent any device, sensor, actuator, or object in the deployment environment. This supports situation specifications, which will be discussed in the section on the Situation Model, to include both physical and digital objects in the environment in situation specifications. The attributes used to describe entities, as shown in Figure 4.4 are identity, type, symbolic location, physical coordinates, entity state, and a timestamp. As is the case for user state, the symbolic location of an entity must correspond with the ID for a location.

Door State

Doors are generally not contained by any single room since they are connectors between zones and so they are contained by floors or by spaces. All doors in the model have an entry and exit point. This enables the model to define which side of the door is contained by the parent location (entry), and which side of the door is contained by the child location (exit). In the case of ambiguous containment relationship between locations, e.g. for conjoined offices, where the parent-child relationship cannot be defined, the entry and exit points are arbitrarily assigned.

Zone State

The Spatial Model defined the relationships between zones in the physical environment. Zone state on the other hand defines the attributes of an individual location. Zone state is defined in terms of ID, purpose, zone type, parent location and child locations. Parent location and child location attributes provide access to nodes above and below this location in the Spatial Model hierarchy.

4.5.3 Situation Model

The state of the art discussion on situation modelling described work by Henricksen and Indulska [50], Nishikawa et al. [80] and Clear et al. [27]. This section builds on these situations models with the aim of providing a more generalised approach to situation-based testing than Nishikawa et al., by adopting a model more akin to the work by Henricksen and Indulska [50]. The Situation Model presented here enables elements of the State Model to be aggregated using logical and spatial relationships.

Situation Specification

This Situation Model generally observes Clear et al.'s [27] definition of a situation, with a minor amendment to remove the limitations on logical relationships to fall more in line with Henricksen and Indulska's [50] use of existential and universal operators. The resulting definition is as follows:

> A situation specification consists of one or more assertions about context that are conjoined using logical operators. Assertions may comprise of further domain-specific attributes, given that the required semantics are available.

The State Model, which maintains the global state of the environment (G), provides the source of available information for this situation-based testing approach. Based on this, a situation (S) can be defined in terms of users, entities, zones, and their attributes, at some point in time, t, as follows:

$S(t) = \{U(t), E(t), L(t)\}$

where

$S =$ A situation specification.
$U = \{u_1, \ldots, u_n\}$ The set of users that exist in the physical environment.
$E = \{e_1, \ldots, e_n\}$ The set of entities that exist in the physical environment.
$L = \{l_1, \ldots, l_n\}$ The set of zones that exist in the physical environment.

Following the work by Henricksen and Indulska [50], situation specifications combine logical connectives AND, OR and NOT, as well as the existential quantifier *exists*, and the universal quantifier *for all*. Table 4.3 defines the notation used in the Situation Model.

Operator	Explanation
$\exists$	Exists
$\forall$	For All
$\nexists$	Not Exists
$\neg$	Not
$\wedge$	And
$\vee$	Or
$\in$	Is a Member Of.
$\notin$	Is Not a Member Of.
$\longleftarrow$	Assignment

Table 4.3: Notation for logical operators of the Situation Model.

Finally, these situation specifications are defined for the purpose of testing the state of the environment at specific points in time. The purpose of testing environment state is to determine if behaviour exhibited in the environment has resulted in inappropriate outcomes. For this reason, test specification follows the format of Nishikawa et al. [80], which uses an *if ... then* rule-based structure. To support traceability, test specifications must log records of their findings, which are a subset ($O(t)$) of the global state ($G(t)$), and are not limited to the situation state information $S(t)$. A test specification can be defined as:

if exists
$S(t) = \{U(t), E(t), L(t)\}$
then
$\mathit{Output} \;\; (O(t) \subset G(t))$

Situation Model Example

The following logic provides an example of how a situation might be defined.

if
$\exists$ (U(type = "visitor") $\longleftarrow$ v)
$\wedge$ $\exists$ (L((id = v.roomLocation) $\wedge$ (type = "office")))
$\wedge$ $\nexists$ (U(((type = "staff") $\vee$ (type = "student")) $\wedge$ (roomLocation = v.roomLocation)))
then
Output (*v.roomLocation, v.userID, Situation ID, unique Alert ID, Timestamp*)

This example situation specification, featured in this test, reads as follows:

A visitor EXISTS in the building, assigned the variable name *v*

AND a location EXISTS which is the same as *v*'s location AND the location is an office

AND there DOES NOT EXIST a staff member or student in the same location.

4.5.4 Activity Model

This section discusses the Activity Model which drives user activity and which focuses on large granular activities in a simulated deployment environment. This Activity Model bounds random behaviour of simulated occupants in the spatio-temporal dimension. The Activity Model is a state-transition based model, similar to that used by García-Valverde et al. [45]. A simulated user travels to some location, or "hotspot" to use Huebscher and McCann's term [55], and stays there for some duration while involved in an activity, e.g. attending a meeting. The spatio-temporal bounds define limits on reasonable occupant behaviour within which randomness can be applied.

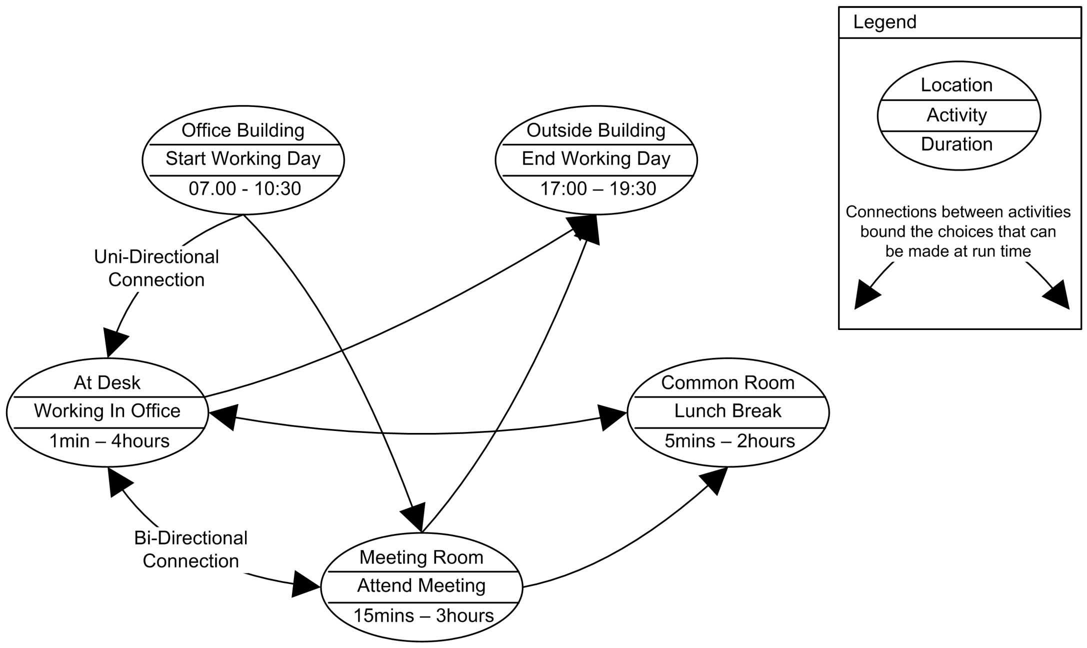


Figure 4.5: Example Activity Model including example relationships between activities.

Padovitz et al. [84] discuss this idea of boundaries using the term situation subspaces. These subspaces define boundaries in the larger context space where situations can arise. They provide the example of the physical limitations on the speed of the human body, both in terms of walking and running. Using these definitions, Padovitz et al. can demarcate areas of the context space where situations arise, and minimise their search which is relevant to the issue of state spaces which can quickly explode in size.

The purpose of the boundary points in this Activity Model is to enable a simulation to test the bounds of reasonable behaviour and to augment the designer's ability to anticipate possible test scenarios. Figure 4.5 shows a directed graph illustrating some example activities that a user might participate in and the possible options for sequences of activities. Each oval includes information about the type of activity (Activity), the location where the activity will occur (Location), and the bounded duration of the activity (Duration). Table 4.4 defines these attributes of the Activity Model. The directional connections in the graph represent the dynamic sequence of activities that can be generated during a simulation.

Activity Model Attributes	
Attribute	Description
Activity	The name of the activity.
Location	The location of an activity.
Minimum Duration	The minimum amount of time an occupant spends at an activity. For example, the time spent getting coffee might be determined by the time it takes to boil the kettle.
Maximum Duration	The maximum amount of time an occupant spends at an activity before moving to another activity. For example, most users can only work for a limited time without the need for refreshment. This is a limitation of the human condition.
Next Activity	The list of possible next activities that a user can perform, one of which is randomly selected.

Table 4.4: Attributes of the Activity Model

Three activity types were identified for multi-user environments that feature both individual and group activities. These are individual activities, shared activities, and group activities. The basic difference between these activities is that individual and shared activities only have one participant at a time. Group activities can have one or more participants at the same time. The three activity types are more clearly defined as:

Individual Activity: Individual activities have one participant and are owned by one occupant. For example, in an office environment occupants often have their own desk.

Shared Activity: Shared activities engage one participant at a time, but can engage many occupants over time. For example, a water dispenser is a shared resource in a work environment, so many users will fill their glasses during the course of a working day, however this activity will be done one at a time, assuming only one tap is available.

Group Activity: These activities arise when one or more participants take part in the same activity physically and temporally co-located, e.g. attending a meeting.

4.5.5 InSitu Model Framework Summary

This section presented the design for the InSitu Model Framework, which consists of the Spatial Model, State Model, Situation Model and Activity Model. The Spatial Model defines the hierarchical representation of the internal layout of a building. The realisation of this model in the State Model, through the parent child relationship, combined with physical coordinates and orientation, provides a hybrid hierarchical model. The State Model defines a testable representation of the state of the environment at any given point in time. The Situation Model defines test situation specifications, for inappropriate situations, which can be used to identify instances of situations at run-time. Situation specifications are an aggregation of environment elements using logical operators and spatial relationships.

4.6 InSitu Test Process

At the beginning of this chapter, Figure 4.1 illustrated the division of concerns between the InSitu Model Framework, Test Process and Toolset. The Toolset components, which are used at run-time, include a Simulation Engine, a Monitoring Engine and an Update Router. The Simulation Engine is responsible for generating both state and sensed context updates. The Monitoring Engine is responsible for generating the State Model at run-time, and testing it against the Situation Model. The Update Router is responsible for maintaining the separation between sensed context and state updates, to ensure that the ubicomp system only receives a sensed view of the environment. This section discusses the underlying process of the situation-based testing approach, that must be realised in the implementation of the InSitu Toolset to support this approach.

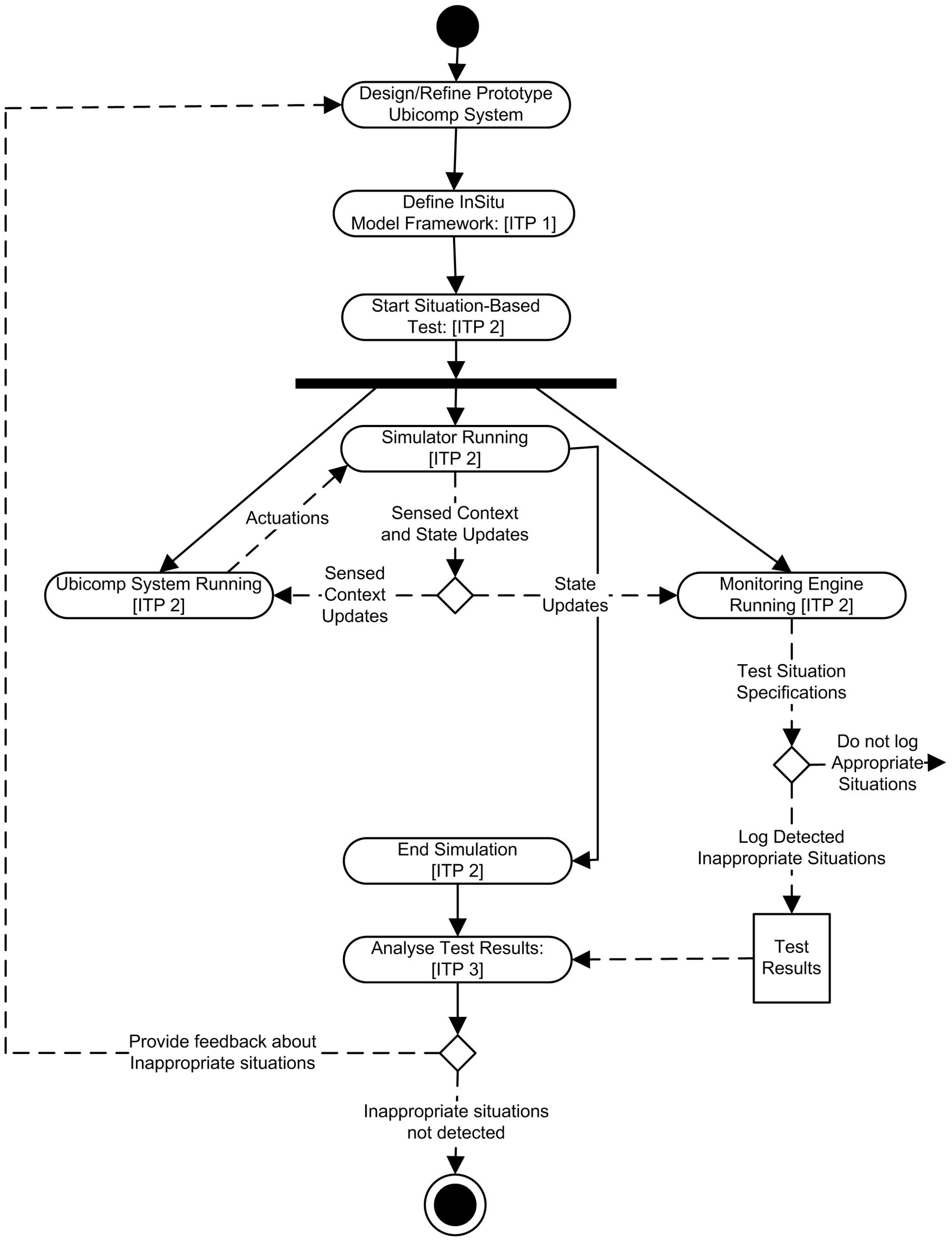

Design/Refine Prototype Ubicomp System
Define InSitu Model Framework: [ITP 1]
Start Situation-Based Test: [ITP 2]
Simulator Running [ITP 2]
Actuations
Sensed Context and State Updates
Ubicomp System Running [ITP 2]
Sensed Context Updates
State Updates
Monitoring Engine Running [ITP 2]
Test Situation Specifications
Do not log Appropriate Situations
Log Detected Inappropriate Situations
End Simulation [ITP 2]
Analyse Test Results: [ITP 3]
Test Results
Provide feedback about Inappropriate situations
Inappropriate situations not detected

Figure 4.6: InSitu Test Process.

Figure 4.6 illustrates the InSitu Test Process. The situation-based testing process begins with the designer designing and building a computer-based ubicomp system prototype. These are the first two phases of an iterative prototyping cycle, as defined by Tang et al. [111]. When this is complete, the designer can begin the testing and evaluation phase. The steps in the InSitu Test Process have been coded in Figure 4.6 for ease of reference, i.e. ITP 1, ITP 2 and ITP 3. These codes are reused later in the thesis, in Chapter 5 to illustrate the relationship between this process design and its implementation. The codes are also used in Appendix D to illustrate the relationship between the end-user instructions and this process definition.

The first step (ITP 1) of the evaluation process is to define the Spatial Model, the Situation Model and the Activity Model. The Activity Model needs to be embedded in or linked with the simulated deployment environment in order to drive spontaneous user behaviour at run-time. The Spatial Model should be representative of the simulation model of the physical environment.

The second step (ITP 2) is run-time testing which begins when the ubicomp system, Simulation Engine and Monitoring Engine are running in parallel. The Simulation Engine generates both sensed context and state updates, which are sent to the ubicomp system and Monitoring Engine respectively. The ubicomp system sends actuation messages to the simulated deployment that alter the state the simulated environment. The Monitoring Engine uses the state update messages to dynamically generate the State Model based on the received state updates, and tests it against the Situation Model. If this results in a positive identification of an inappropriate situation, a log of the instance is output to a set of test results.

In the third step (ITP 3), the designer analyses the test results (Alert Report) to evaluate the appropriateness of exhibited system behaviour. An evaluation cycle ends with the designer either looping back to the start to use the findings from the test results to refine the ubicomp system's design, or with the designer exiting the prototyping cycle because the testing approach is no longer uncovering new flaws in the design.

This test process is core to the situation-based testing approach. An implementation of this process is presented in the next chapter.

4.7 Chapter Conclusions and Summary

This chapter presented the design for the InSitu Model Framework and the InSitu Test Process. The requirements set out for this design expanded on objectives **O2** and **O3**, which respectively state the need for a formal framework for test specification, and the need for a systematic test process which features a feedback loop.

The Spatial Model builds on previous hybrid hierarchical models, by providing a formal definition of containment and proximity. Containment in this model is more expressive, in terms of zone types, than the models of Satoh [95], Bandini et al. [11,10] or Jiang et al. [57]. The definition of proximity is a modification of Bandini's definition, to include people and objects, as well as zones. The inclusion of novel operational zones in the Spatial Model aims to reflect the observation that the operational usage of buildings is not totally dictated by the physical structure of a building. Operational zones provide the capacity for the hierarchical model to represent this.

The State Model provides a vocabulary of attributes that can be used in situation specifications. This vocabulary was defined for the purpose of testing action-based ubicomp systems in indoor environments. It is not a definitive vocabulary for the full ubicomp field. However, this is to be expected based on Clear et al.'s [27] definition of situation specification which states that attributes can be domain specific and are subject to availability.

The Situation Model follows the models by Henricksen and Indulska [50], Clear et al. [27] and Nisikawa et al. [80]. It is more generalised that Nishikawa et al.'s approach, which is important, because that was a central limitation identified in the UbiREAL situation-based testing approach. The Situation Model features the logical operators of Henricksen and Indulska and follows Clear et al.'s definition of a situation specification.

The Activity Model has followed the trend of activity-based models, which Tabak and de Vries [110] note as useful for building usage simulation. It has been defined to bound situation subspaces at reasonable limits in the spatio-temporal dimension. This aims to promote realistic simulations that still allow for spontaneous, unanticipated and outlier situations to arise.

The combined features of the Situation, Spatial and State Models address requirement **R1**. The Activity Model, and the combined Spatial and State Models, address requirement **R2**. Although the InSitu Test Process goes some way to addressing requirements **R4** to **R7**, the implementation of the Toolset is needed to fully address these requirements. For this reason, they will be discussed at the end of the next chapter.

The novelty of the InSitu design lies in the support it provides for monitoring the global state of a simulated ubicomp deployment environment, specifically to identify inappropriate situations, without prior knowledge of the scenarios or sequences of activities that will arise at run-time. The Spatial Model is instrumental in this because it informs the Monitoring Engine about the layout of the space, thereby enabling the engine to reason about the spatial relations between elements and zones at run-time. The State Model provides the vocabulary of attributes that can be tested. The discussion on the implementation of this model, in the next chapter, will show how this model is testable by the Monitoring Engine. The Situation Model provides generalised definitions of situations that can be used to identify instances of situations at run-time. This promotes efficiency during test specification, and thoroughness of testing at run-time.

The InSitu Test Process is also core to the novelty of the design. This is largely due to the role of simulation in the testing approach. Simulation affords an opportunity to capture a state view of the deployment environment, something which is not possible in live test environments, even using current state of the art sensing technology. This is key to testing the summative effect of system and user behaviour. The combination of the Simulation Engine with the Update Router enable the Monitoring Engine to maintain a state view of the simulated environment, i.e. the State Model.

Testing this against the Situation Model allows for the actual effect of the system, in the deployment environment, to be assessed.

4.7.1 Summary

This chapter began by providing an overview of the situation-based testing approach. A set of requirements were defined based on the findings and influences from the state of the art. Following this, an overview of InSitu was provided, including the Model Framework, Test Process and Toolset. The Model Framework and Test Process provide a generalised, technology independent, definition of the situation-based testing approach. The Toolset is an implementation which provides the mechanism for evaluating the situation-based testing approach. The Model Framework and Test Process were presented in this chapter. The Toolset is a technical implementation of the approach and is presented in the next chapter.

Chapter 5

Implementation

5.1 Introduction

The previous chapter presented the design for the situation-based testing approach in the form of the InSitu Model Framework and InSitu Test Process. The Model Framework was defined to provide the formal test specification framework, while the Test Process provides the systematic method of testing which supports feedback to the designer. This chapter focuses on the toolset implementation that realises this design. The InSitu Toolset implementation is necessary as a mechanism to evaluate the underlying situation-based testing approach. This chapter discusses the main divisions of the implementation, which are simulation, message routing, situation testing, and results analysis. This chapter also presents a brief discussion on the performance of the Toolset to demonstrate that it is satisfactorily efficient for use as an instrument to evaluate the situation-based testing approach.

5.2 Overview of the InSitu Implementation

Figure 5.1 provides an overview of the InSitu Model Framework, InSitu Test Process, and the relationships between the models, process and InSitu Toolset components. The Model Framework and Toolset are each bounded by the green and blue dotted rectangles, respectively. The InSitu Test Process is defined by the arrows between the models and components in the diagram. The red dashed rectangles indicate the main divisions in the functionality of the InSitu Toolset.

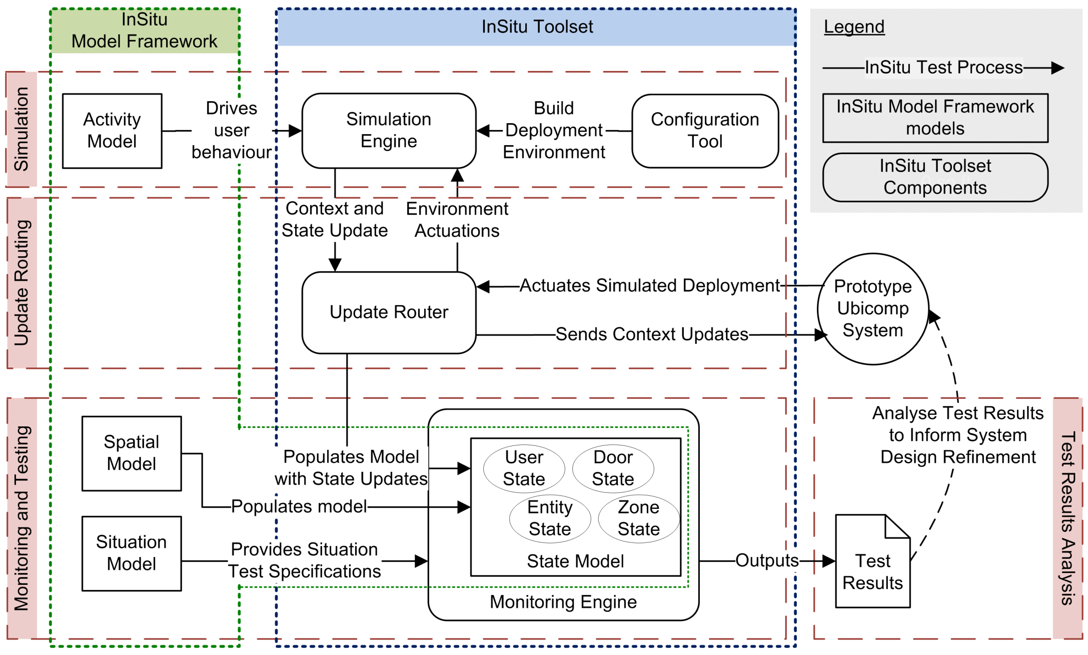


Figure 5.1: Overview of InSitu Implementation. The horizontal, red dashed-line, rectangles, highlight the main divisions of the Toolset.

The *Simulation Engine* is configured with a 3D model of the target deployment environment and draws on the Activity Model to drive bot behaviour. The Simulation Engine generates both sensed and state updates.

The *Update Router* directs messages within the Toolset. Sensed context is directed from the Simulation Engine to the prototype ubicomp system. State messages are directed from the Simulation Engine to the *Monitoring Engine*. Actuation instructions are directed from the prototype ubicomp systems to the Simulation Engine.

The Monitoring Engine uses situation specification test definitions from the Situation Model to examine the state of the environment for inappropriate outcomes. The Monitoring Engine outputs these test results which detail the instances of inappropriate situations that were detected in the simulated deployment environment. Analysis of the test result log provides insight into the extent of inappropriate situations arising in the simulated deployment environment. This is used to inform design refinement for a prototype ubicomp system under test.

The implementation of the Toolset components is discussed over the course of this chapter.

5.3 Deployment Environment Simulation Engine

This section focuses on aspects of the implementation that address requirement **R4**, which was stated in Chapter 4 as *A simulation based test-bed for ubicomp should be configurable, extensible, reusable and scalable, with relatively low investment and should support heterogeneous context generation.* The Activity Model, Simulation Engine and configuration tool, together address this requirement through simulation of the Activity Model, the physical building, mapping zones from the Spatial Model to the building simulation, and finally generation of sensed context and state update messages.

5.3.1 TATUS and Half-Life 2

The Simulation Engine for this Toolset builds on an existing well-cited ubicomp simulator called TATUS [82, 83]. TATUS extended the Half-Life game engine[1] as a tool for simulating the physical environment. Trenholme and Smith [113] provide a survey of the role game engines can play in research projects, and identify the merits of these as research tools. Game engines are particularly beneficial as a basis for building simulation environments, because they have already undergone rigorous testing for performance, reliability and usability. Modern games, including Half-Life and Half-Life 2, are sophisticated, offering realistic environments and advanced artificial intelligence for believable bots, which with standard animations can walk, run, jump, crouch and sit, as well as navigate around collisions and obstacles. Many are provided with software development kits (SDK) enabling developers to build

[1]Half-Life and Half-Life 2 website http://orange.half-life2.com/ (accessed 6th December 2010).

on them. The Half-Life SDKs[2] provide both source code for the game and also a mapping tool called Hammer for building models of the physical world.

TATUS extended Half-Life by introducing simulated sensed context sources, which generate context messages in response to bot or avatar activity. Avatars in TATUS move based on input from a user operating the Half-Life game controls. In addition to context sources, TATUS made two other key modifications to Half-Life. The first provided a TCP/IP connection to send information (sensed context and actuations) in and out of a simulated physical world. The second modification added software objects to the game, which were responsible for directing context messages out of the game, and directing actuation instructions to appropriate simulated physical objects in the game. Through this mechanism, ubicomp systems can receive context updates about the environment, and in response actuate physical objects in a simulated deployment.

Since TATUS was developed, the Half-Life 2 engine was released. Half-Life 2 is a more sophisticated engine, which offers realistic physics, a particle system for environmental effects, and fine-grained control for character animation, including facial expressions. Although not all of these features are used for InSitu, their presence contributes to extensibility of the Simulation Engine. The tools in the SDK for character and facial animation, physical object model creation, and world creation (Hammer) mean that the simulator is also immediately configurable. More recently, Valve the creators of Half-Life 2, have released a plugin[3] for Hammer which supports map creation in Google SketchUp[4]. This not only provides an additional tool for world and object creation, it also provides the potential for reusing models from the Google SketchUp Gallery[5]. The caveat of Gallery models is that some are too complex for game engine rendering, and so not all are candidates for reuse.

[2]Developer pages for Half-Life 2 http://developer.valvesoftware.com/wiki/Main_Page (accessed 6th December 2010).

[3]Valve plugin for exporting Google SketchUp models to VMF or SMD formats. http://developer.valvesoftware.com/wiki/SketchUp_to_Hammer_Export_plugin (accessed 6th December 2010).

[4]Google SketchUp http://sketchup.google.com/ (accessed 6th December 2010).

[5]Google SketchUp Gallery http://sketchup.google.com/intl/en/community/gallery.html (accessed 6th December 2010).

5.3.2 Simulation Overview

Figure 5.2 provides an overview of the implementation of the Simulation Engine featured in the Toolset. The Hammer World Editor is used to construct the model of the physical environment, including positioning of sensed context sources and state update sources, and to perform the task of configuring the Activity Model for a simulation. The map compiled by Hammer is used at run-time to drive a test simulation.

During testing sensed context sources in the environment generate messages appropriate to the type of source. For example, a motion sensor only includes information that indicates presence, it should not include the identity of individuals within its range. State update messages are generated by the Simulation Engine to reflect the actual state of the environment at a particular point in time, as opposed to a sensed view. Actuation instructions are received via the Update Router, which include information to alter the state of the physical environment.

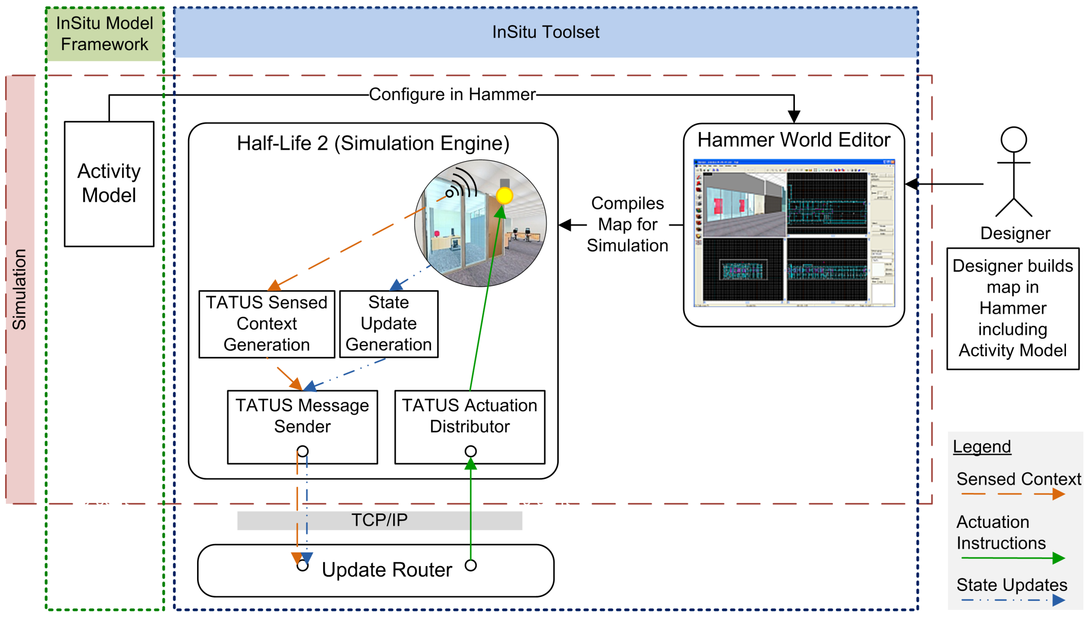


Figure 5.2: Overview of Simulation Component implementation, including its relationship to TATUS, Half-Life 2 and Hammer.

5.3.3 Activity-Based Simulation

In Chapter 4, an activity-based model was introduced that is bounded in the spatio-temporal domain. The Activity Model uses randomness to determine the sequence and duration of activities for a bot. Three standard Half-Life 2 game entities were used to implement this model. A *scripted sequence* is used to define the location and orientation of an activity. For visual effect, scripted sequences can make use of a suitable animation from the Half-Life 2 library of animations. A *logic timer* defines a maximum and minimum duration for a state, at some point during which the timer randomly fires. A *logic case* is a logic decision point, which can be programmed with up to 16 options that can be activated during game-play.

The process of including the Activity Model in a simulation involves manually configuring these game entities in Hammer. As already mentioned, Half-Life 2 provides sophisticated AI-driven navigation for bots, which addresses collision detection and obstacle avoidance. This is used to provide the animation and simulation of bots moving between activities. Hint entities are used to assist bots to navigate more believably, e.g. through doors, or to prevent bots walking in particular areas if necessary.

A limitation of the scripted sequence game entity is that they are not designed to be shared. Each scripted sequence is only intended to be used by the bot explicitly named in its property set, which is a problem for simulating shared or group activities. To overcome this issue, a system of hand shakes was implemented, as a modification to the scripted sequence, so that ownership of an activity can be dynamically changed at run-time.

The randomness of the timers and logic case is an in-built game function, which is contained in a closed source part of the game engine. For this reason it is not possible to ascertain the method used in the random function, however it is assumed that this function does not provide true randomness, but rather a pseudo random number[6].

[6]Comments in the Half-Life 2 source code indicate that the random function uses an MD5 hash and that it takes a seed as a function parameter in order generate a pseudo random number.

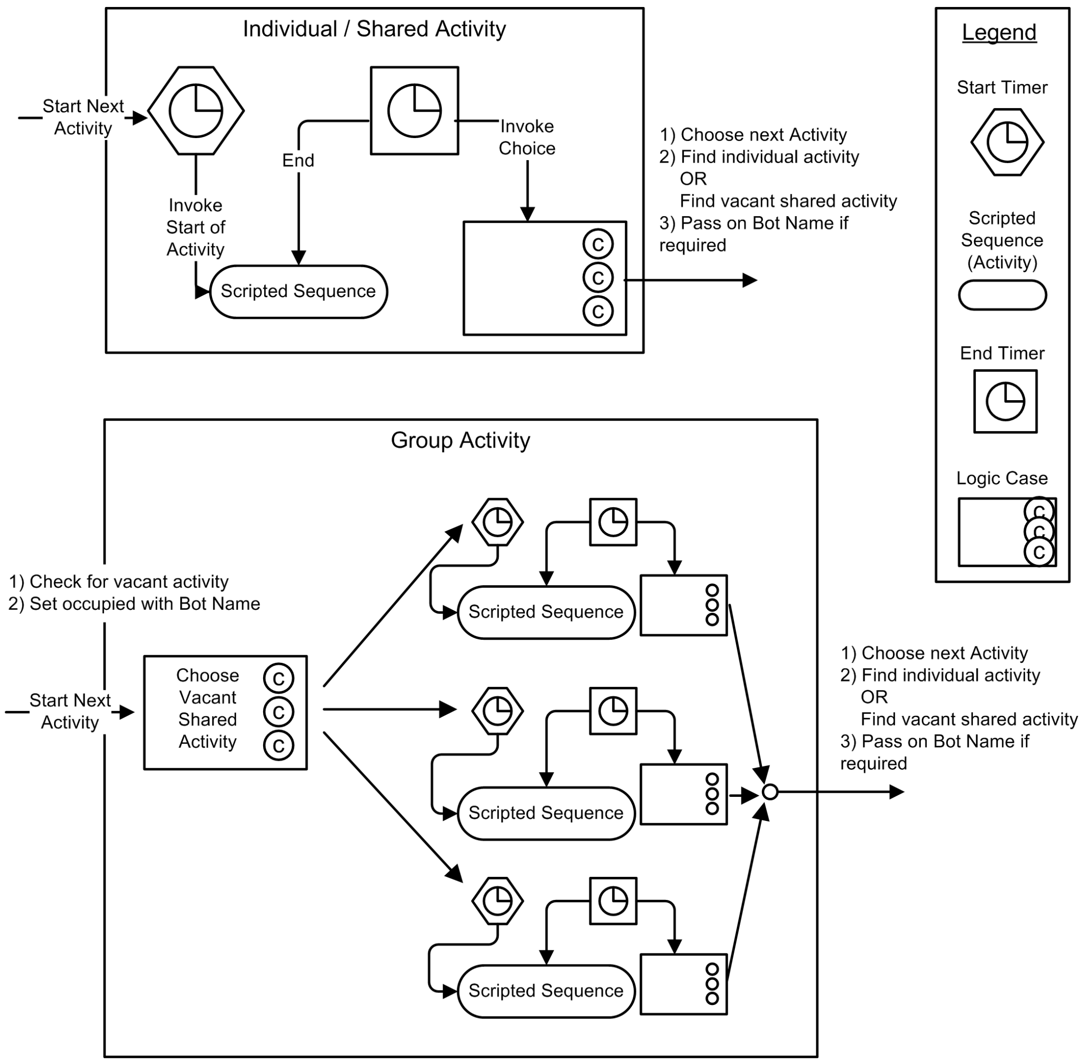

Individual / Shared Activity
Start Next Activity
Invoke Start of Activity
End
Invoke Choice
Scripted Sequence
C
1) Choose next Activity
2) Find individual activity
OR
Find vacant shared activity
3) Pass on Bot Name if required
Legend
Start Timer
Scripted Sequence (Activity)
End Timer
Logic Case
Group Activity
1) Check for vacant activity
2) Set occupied with Bot Name
Start Next Activity
Choose Vacant Shared Activity
Scripted Sequence
1) Choose next Activity
2) Find individual activity
OR
Find vacant shared activity
3) Pass on Bot Name if required

Figure 5.3: Activity Types: Individual/Shared Activity (Top), Group Activity (Bottom)

Activity Types

This section discusses the implementation of the three activity types identified in Section 4.5.4 which included Individual Activity, Shared Activity and Group Activity. Figure 5.3 illustrates the connections between the game entities already discussed which make up the implementation of each of these activity types.

A start timer fires periodically, using short time cycles, until it confirms that a bot has reached its next activity. The start timer then invokes the animation sequence for the activity. The end timer randomly stops the animation at some time between the maximum and minimum durations. The end timer invokes a random selection of one of the pre-programmed options in the logic case. The logic case activates the start timer for the randomly selected activity. Figure 5.3 (top) illustrates this sequence.

Individual Activity: An individual activity is one that is generally owned by an individual occupant, e.g. a desk space. Implementation of individual activities follows the description outlined above.

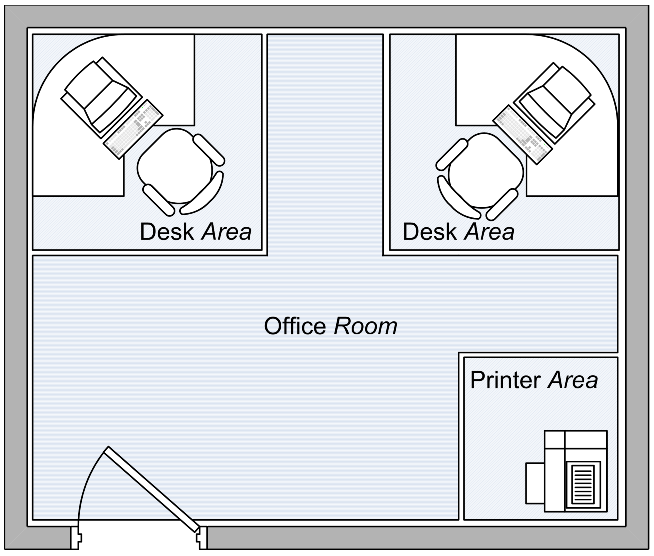
Shared Activity: A shared activity is one that is only occupied by one individual at a time, but can be used by many occupants over time, e.g. a water dispenser. A shared activity is basically the same as an individual activity, except that a check must be completed before the logic case activates the start time, to confirm that the activity is vacant.

Group Activity: A group activity is one in which multiple bots participate simultaneously, e.g. in a meeting room situation. Figure 5.3 (bottom) illustrates that this is implemented as a collection of shared activities.

5.3.4 Environment Zones

Chapter 4 discussed the hierarchical Spatial Model which divides the internal layout of a building into zones. While the Spatial Model provides the containment relationship between zones, it did not provide the physical delineation of the size of individual zones. Figure 5.4 illustrates how this is achieved through the use of Hammer to map out zones on a floor plan. Each zone in the floor plan is a game entity and so can have its own unique identifier. This unique identifier corresponds to the zone's ID, which featured in the Spatial Model.

Zones entities are dual purpose. In addition to demarcating the floor-plan for the hierarchical model, they also act as the state update generators. Their role in this is discussed in the next section.



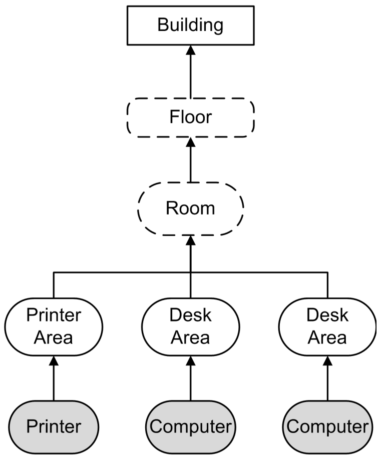


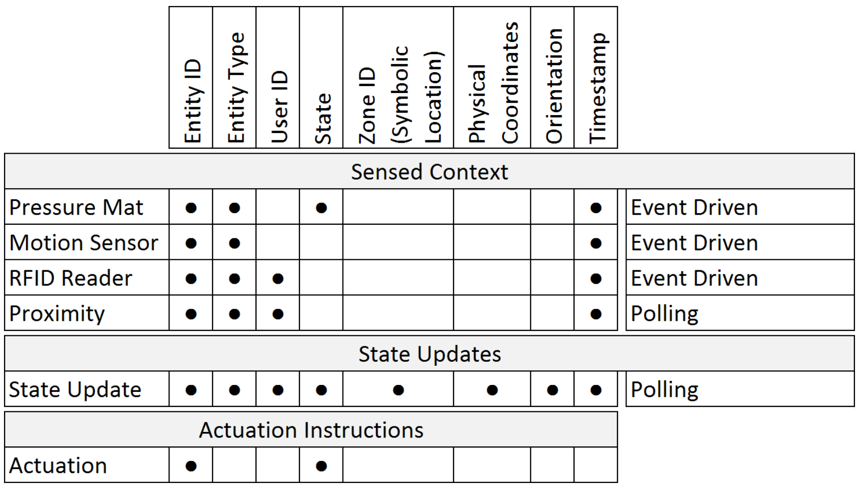
Figure 5.4: State Zones: Layout (left) and Hierarchy (right).

5.3.5 Sensed Context and State Update Generation

The InSitu Toolset extends TATUS by providing support for state update messages, in addition to sensed context messages. State update messages include the necessary information to populate the dynamic attributes of the State Model, as well as unique identifiers to assist reconciling this information.

Context sources, and the messages that they produce, are modelled to generate the type of information that sensed sources will realistically supply. Figure 5.5 illustrates some examples of context value sets used for various information sources. For example, a motion sensor detects presence, but not position or identity, and so its context values include entity ID, entity type and a timestamp. A state update message, on the other hand, requires a full picture of all relevant information, and so includes all of the values listed in this table.

Both sensed and state information sources in the Simulation Engine are based on Half-Life 2 game objects. The SDK code base and the Hammer Map Editor are extensible for this purpose, and so Hammer can be used as the configuration tool for placing and configuring sensed context and state update sources in a simulated deployment environment. Although Hammer provides limited opportunity for fine-grained configuration of sensed context sources, work by McGlinn [75] has



	Entity ID	Entity Type	User ID	State	Zone ID (Symbolic Location)	Physical Coordinates	Orientation	Timestamp	
Sensed Context									
Pressure Mat	●	●		●				●	Event Driven
Motion Sensor	●	●						●	Event Driven
RFID Reader	●	●	●					●	Event Driven
Proximity	●	●	●					●	Polling
State Updates									
State Update	●	●	●	●	●	●	●	●	Polling
Actuation Instructions									
Actuation	●			●					

Figure 5.5: Sensed Context, State Update, and Actuation Information.

progressed this area by providing more control over the configuration of sensor models in the simulator.

State update sources poll the environment periodically. The frequency for this can be set using the Hammer properties dialog box. This setting is at the discretion of the tester or system designer, however a relevant consideration is the duration of an inappropriate situation. Short-lived situations require more frequent updates to ensure they are not missed.

Sensed context and state update messages are encoded in XML and timestamped in the game engine. The XSD schema for these messages is included for reference in Appendix H.

5.3.6 Simulation Engine Analysis

The strengths of the Simulation Engine implementation lie in the use of open source game technology that actively supports modification of the game's code-base. The Half-Life 2 source code is accessible and extensible, with an active developer community and forums. The SDK provides tools to build and configure the physical world as well as bot animation and appearance. The world editor, Hammer, is extensible enabling context source types to be added to it, so it can act as a configuration tool for placement of context sources in the physical environment.

The Simulation Engine is scalable in comparison to live test environments and user controlled simulations. The Half-Life 2 game engine supports up to 2048 entities per simulation, which includes simulated users and entities. As an example of scale, a project separate to this thesis constructed a three storey office building model, which includes 104 rooms comprised of offices, computer labs and seminar rooms, and which is furnished with 520 desks, 352 chairs and 257 replica desktop computers. This model was created in a project separate from the research for this thesis, and features as a tool in the work of Feeney [40] and Quinn [91].

Many aspects of the Simulation Engine are reusable. The maps and bot animations are configurable and reusable, which means that they can be reused in consecutive test cycles. The separation of Model Framework from the Test Process and Toolset, also means that individual models can be reused. The exception is the State Model, but because this is automatically generated at run-time, it does not need to be accessible to the tester or reusable. Finally, the implementation of the test configuration files for the Update Router and the Monitoring Engine, also promotes reusability of test configurations.

Finally, in terms of lower investment, the price of the Half-Life 2 game engine, less than €10.00[7] at the time of writing this thesis, is by comparison to a live deployment very low cost. Configuring a test simulation still requires an investment of man hours, however game entities, including context sources and Activity Model configurations, are positioned using drag and drop interaction. During a user trial, the average time to configure an activity in Hammer, after a learning curve was overcome, was four minutes and seven seconds. An activity is comprised of four game entities, whereas a context source is a single game entity. Based on this it can be expected that, if timed, the task of positioning a context source in a map, would average a time notably less than four minutes.

The most significant investment is the man hours required to construct a physical deployment environment using Hammer. Two buildings modelled for the Simulation

[7]Price listing for Half-Life 2 on Steam http://store.steampowered.com/app/220/ (accessed 10th December 2010).

Engine, as part of other projects, took approximately six weeks each of dedicated man hours to create. The first project modelled the first three floors of a building in Trinity College Dublin, the second project modelled a two storey building located at Cork Institute of Technology[8], Cork.

5.3.7 Simulation Engine Summary

The choice of the Half-Life 2 game engine, as the basis for the Simulation Engine, addresses configurability, extensibility and reusability, in requirement **R4**. The integration of TATUS functionality into Half-Life 2 addresses context generation that is heterogeneous and representative of users, entities and locations. The extension of the context generation functionality to also include state generation using zone entities contributes towards satisfying requirement **R3**. Zone entities are also relevant in terms of realising the hierarchy of the Spatial Model from Chapter 4, and so contribute towards satisfying requirement **R2**. The activity simulations realise the Activity Model from Chapter 4 and contribute towards satisfying requirement **R2**.

[8]Cork Institute of Technology http://www.cit.ie/ (accessed 16th December 2010).

5.4 Routing Sensed Context and State Updates

The discussion in this section focuses on addressing requirement **R5**, which was stated as *separation between simulated sensed context and environment state updates must be maintained.* The Update Router addresses this requirement by maintaining the connections between the ubicomp system, the Simulation Engine, and the Monitoring Engine. It routes sensed context, state updates and actuation messages appropriately across these connections.

5.4.1 TATUS Proxy

A proxy, implemented in Java, was a feature of the TATUS architecture that acted as a middleman between a ubicomp system and the Half-Life game engine. Its role was to support asynchronous communication between these two components so that each could operate independently. This proxy also provided the functionality to establish and maintain TCP/IP network connections between the TATUS Simulator and ubicomp system during testing. This has two specific benefits. Firstly, the simulator can be run on a dedicated machine because the Half-Life game engines are resource intensive. Secondly, the ubicomp developer can run their code base from their own development machine, and do not need to either move their code or install TATUS on their local machine. This distributed network architecture is reused in the implementation of the Update Router for this Toolset.

5.4.2 Update Router

The internal components of the Update Router, and their respective relationships, are illustrated in Figure 5.6. The Router component is responsible for directing sensed, state and actuation messages to their appropriate destinations at run-time, as indicated by the arrows in Figure 5.6. The Manager components are responsible for establishing connections with the Simulation Engine, prototype ubicomp system and Monitoring Engine. The Simulator Manager houses the original TATUS Proxy code to connect to Half-Life. System Managers support ubicomp systems to connect and disconnect at run-time without disrupting a test. This allows for parallel testing of ubicomp systems, which can be expected in real deployments. The Monitoring Engine Manager reuses the System Manager class, but only requires an output connection to route state updates.

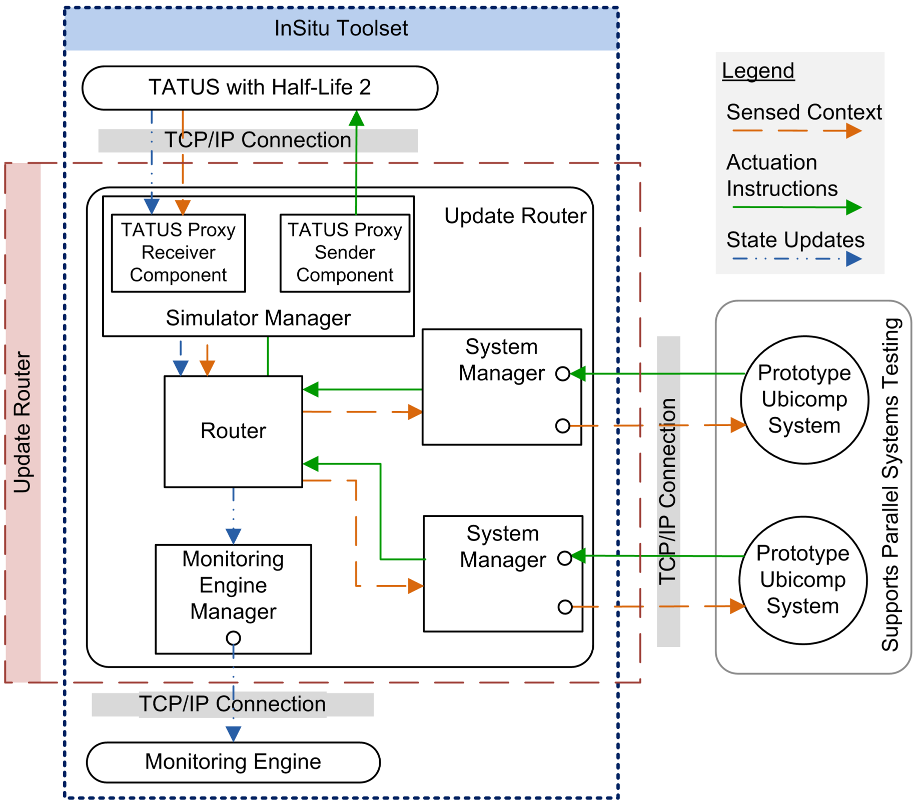


Figure 5.6: Components of the Update Router.

Message routing between the Simulation Engine, ubicomp system and Monitoring Engine uses a unique identifier for individual ubicomp systems and a set of information subscriptions per recipient. Unique identifiers for ubicomp systems are used to support parallel testing, so the router can differentiate between ubicomp systems and the Monitoring Engine. Data subscriptions go further to support parallel testing, by allowing each system to specify the type of information sources that they wish to subscribe to. The ubicomp systems should only ever subscribe to sensed context sources during testing. Similarly, the Monitoring Engine should only subscribe to state update sources during testing. The Monitoring Engine only receives updates about actuation instructions when the actuation has been completed.

The main focus of this discussion on the Update Router was to illustrate that it satisfies requirement **R5**. However, the Update Router has a number of additional features that promote ease of use. The first of these include the use of Java Remote Method Invocation (Java RMI) which enables the Update Router to automatically start the Simulation Engine on behalf of the tester as soon as the Ubicomp Prototype establishes a connection.

A second feature of the Update Router is a test configuration file. The test configuration file supports definition of connection settings, e.g. simulator host and ports, directory for model framework and subscriptions to updates from the router. The test configuration file is defined as an XML file. The XML Schema Definition (XSD) for this file is included in Appendix H.

5.4.3 Summary

The Update Router presented in this section was developed to address requirement **R5** by ensuring that it is possible to separate state and sensed context updates. Although this does not provide a full context management system (CMS), as seen in other ubicomp test-beds, e.g. García-Valverde et al. [46, 45], it provides the necessary functionality to implement this part of the InSitu Test Process, with the guarantee of separation of sensed and state updates.

5.5 Situation Monitoring

The implementation discussion in this section focuses on addressing requirement **R6**, which was stated as *a Monitoring Engine must be able to test environment state, which can be distributed in the ubicomp space, to automatically identify instances of inappropriate situations, without prior knowledge of the sequences and scenarios that will arise at run-time.* A Monitoring Engine has been implemented to address this requirement, together with implementations of the Spatial Model, Situation Model and State Model. The representation of environment state in an information model enables the Monitoring Engine to reason over the global state of the environment. Generality of situation specifications allows them to be dynamically reconciled with global environment state at run-time.

5.5.1 Drools and the Adaptive Engine

Drools[9] is a business rule management system (BRMS) which includes a production rule system. Production rule systems operate by testing a defined condition against the state of the world. The Drools rule engine uses an enhanced implementation of the Rete algorithm [43] in its production rule system. The Rete algorithm improves efficiency by using pattern matching on a network of nodes. This is beneficial in terms of the speed at which rules can be executed.

A limitation of the Rete algorithm is that it is memory intensive and ultimately this is a limiting factor in terms of computing resources. However, the case studies presented later in this thesis do not reach the scale where this becomes problematic, and as such Drools is a suitable technology for the implementation of InSitu used to evaluate the situation-based testing approach. The separation of concerns in Drools, which means the condition is encoded as a declarative rule, and the state of the world is defined as a set of facts, is a further benefit of the technology for implementing InSitu. There is a natural mapping between the separation of rules and facts, and the separation of concerns between the Situation Model and State

[9]Drools website http://www.jboss.org/drools (accessed 6th December 2010).

Model.

The Drools engine used in the Toolset implementation is housed in an eLearning Adaptive Engine [29]. Although not directly related to this work, the Adaptive Engine also uses a model framework and as a result has existing functionality for model management, including loading, reading, writing to, and saving models. In particular the Adaptive Engine has in-built custom functions specifically for reading from Drools Rules and XML models, and writing to XML models. The existing integration of XML in the Adaptive Engine, and also in TATUS, was influential in the choice of XML to represent the Spatial Model, state initialisation file, and the test results, as opposed to an alternative data modelling technique.

5.5.2 State Model Generation

Drools facts use a Java bean implementation and are an extensible part of the Drools rule engine. This made it possible to create custom domain specific facts relevant to the ubicomp deployment environment. The State Model, defined in Chapter 4, models the state of the environment at a particular point in time. A Drools fact vocabulary has been defined, which implements the State Model. The attributes of the vocabulary are accessible through getter and setter bean methods. Most attributes in the State Model can be populated directly from the state updates received from the Simulation Engine.

Static state and initial dynamic state values are read from an XML file containing initialisation values to describe the initial configuration of the deployment environment, as illustrated in Figure 5.7. The Spatial Model is also part of the initialisation process because for the purpose of testing it is assumed that the hierarchy of zones is static.

Dynamic state information, supplied by the Simulation Engine and received via the Update Router, is processed continuously throughout testing by the Update Service and State Model Generator. At run-time, the State Model Generator dynamically updates the State Model, creating new Drools facts if it encounters new users,

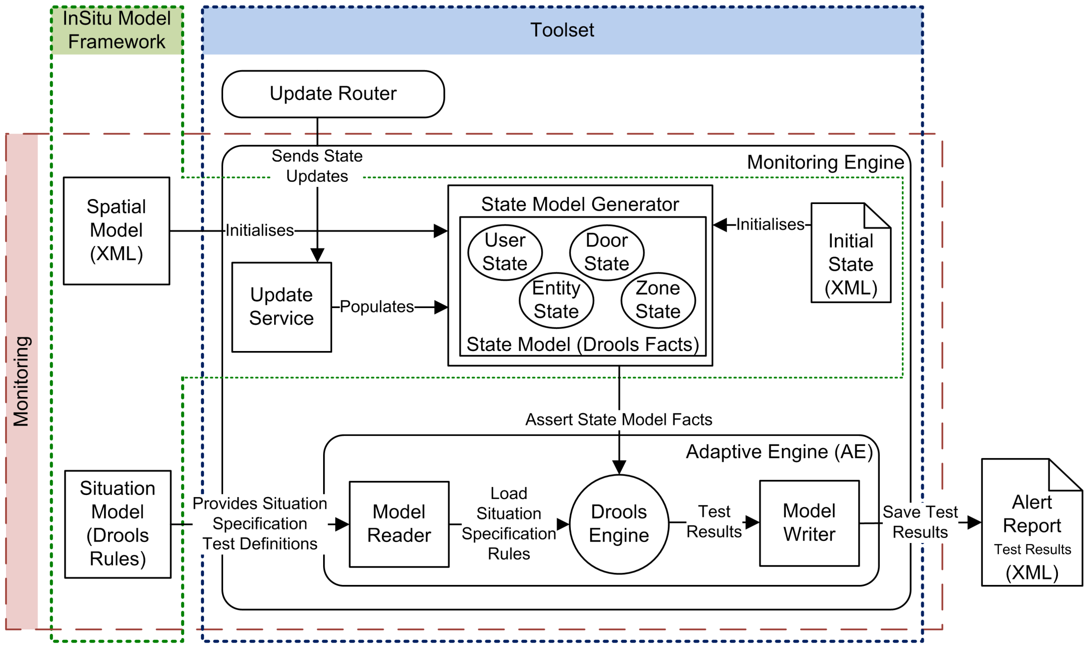


Figure 5.7: Overview of Monitoring Engine Component

entities, doors or locations, however this should not be the case for the latter two of these because the hierarchical model is unchanging during a simulation.

For the purpose of efficiency, due to the role of the Rete algorithm and pattern matching analysis, user and entity facts have been implemented to store the full hierarchical location of the individual or object that they represent. This means that these facts have a record of location at each of the building, floor, vertical-space, space, room and area, levels in the hierarchy. Zones which do not have a valid value at a specific point in time, are set to null.

The only attribute that is not taken directly from state update messages, and which requires additional processing, is field-of-view (FOV). The State Model Generator performs additional calculation to generate the set of other users and entities in each individual user's field of vision. The model of human vision is based on anthropometric data from the book *Human Dimension & Interior Space –A Source Book of Design Reference Standards* [85].

According to these parameters, the human horizontal field of vision is comprised of a central binocular region of vision of 124°, and two monocular regions of vision, left and right, each of between 32° and 42°. This provides a maximum horizontal field-of-vision of 208°. This defines the range that can be seen without a person moving their eyes or head, however objects are less clear in the regions of monocular vision. Bots in Half-Life 2 have presets for FOV at 50°, 90°, 120°, 270°, and 360°. Since these options are not particularly close to the anthropometric model, this calculation was performed outside the game engine, based on the state value of a bot's orientation.

An assumption made in this model is in the calculations of physical boundaries for a bot's FOV. The model assumes that a bot can see the full distance of the room in which they are present. This means that walls are the obstacle and boundary to a bot's FOV. A limitation of the model is that it is calculated as a view wedge, rather than a view cone. A more complete model would take into account the human vertical field of vision.

5.5.3 Situation Model Implementation

Drools rules take the form of a *when* ... *then* statement. These are referred to as *left-hand-side (LHS)* and *right-hand-side (RHS)* respectively. The LHS defines a set of conditions that must be true in order for the rule to fire. Situation specifications are defined in the LHS of a Drools rule. The example below illustrates a situation specification which describes a user in the dark. The first part of the example shows the situation specification, as would be written using the Situation Model format defined in Chapter 4. The second part of the example, shown in Listing 5.1, provides the equivalent Drools rule encoding.

Situation Specification A.1: *User in the Dark*

if

$\exists$ (U $\longleftarrow$ u)

$\wedge \nexists$ (E ((type = "light") $\wedge$ (roomLocation = u.roomLocation) $\wedge$ (state = "on")))

then

Output $O(t) \subset G(t)$

Listing 5.1: Example Drools Rule: A user is in the dark.

```
rule "ID.1: Lights Off for User"
  when
    $user : UserStateFact ($userLocation:userLocationRoom)
    not exists ( EntityStateFact (entityType == "light",
                  entityLocationRoom == $userLocation,
                  entityState == "on") )
  then
    //$Output alert using Adaptive Engine Java method calls
end
```

Logging Test Results

The RHS of a Drools rule is used to generate test results when an instance of a situation specification is found. The information recorded by the RHS of a Drools rule is at the discretion of the tester, however it is recommended that, at a minimum, it should include an identifier for the situation specification, as well as spatial and temporal information so that the spatio-temporal time line can be charted during analysis of test results. The RHS of a rule is a collection of Java method calls which produce an *alert*. Each alert logs information about a detected situation. Alerts can contain information from any part of the State Model, which means that the information content of an alert is limited by the State Model, and not by the bounds of the situation.

The Adaptive Engine provides the custom functions which are used to navigate and output Test Results. These are generated in XML format and saved in a file called an Alert Report. Four functions in particular are used for this process. The first is *navigateModel* which is used to change location in the XML nesting. The second is *addNode* which is used to add a new element to the XML report, e.g. when creating a fresh alert. The third is *addAttribute* which is used to add an attribute to an existing element. The fourth is *storeModel* which is used to save new changes to the XML file after a new alert has been output to the model.

At run-time, the RHS encoding of rules for the Situation Model generates one alert each time any rule detects a specified situation. This can result in multiple outputs by the same rule if two instances of a situation are detected in the same rule cycle. For example, in Listing 5.1, the rule is testing for occupants in the dark. According to this encoding, if two users are experiencing this situation simultaneously, the rule will output two alerts. Additionally, multiple alerts will be output for the same situation instance if it persists over two or more rule cycles.

Rules can be configured to run after each state update or on a timed cycle, at the discretion of the tester. A number of factors should be considered when deciding on this setting. The criticality of an inappropriate situation is influential in how important it is that all instances are found. For highly critical situations, rules may need to be configured to run after each state update, particularly, if situations are of short duration. This should be paired with highly frequent generation of state update messages in the Simulation Engine.

Duration is the second factor that should be considered. Situations of long duration tested by rules configured to run very frequently will generate very large Alert Reports. This has the potential to place an increased, and in some cases, an unnecessarily heavy load on the tester during manual processing and analysis of the Alert Report.

Examples of full situation specification encodings, i.e. Situation Models, are provided in Appendix A and Appendix B, which include both situation specifications (LHS) and alert structure definitions (RHS).

5.5.4 Monitoring Summary

This section presented the implementation of the Monitoring Engine, which is included in the Toolset to address requirement **R6**. It also completes the fulfilment of requirement **R3**, which was partially fulfilled by the design of the State Model.

Requirement **R3** was stated as *a testable model of environment state must be defined and maintained*. The State Model discussed in the design chapter fulfils the requirement that a model of environment state must be defined. A set of custom Drools facts was implemented to realise the elements of the State Model, i.e. user, entity, location and door. The Monitoring Engine uses initialisation values, the Spatial Model, and state updates from the Simulation Engine to create and maintain instances of these objects at run-time, which collectively define the State Model at any given point in time.

Requirement **R6** was stated as *a Monitoring Engine must be able to test environment state, which can be distributed in the ubicomp space, to automatically identify instances of inappropriate situations, without prior knowledge of the sequences and scenarios that will arise at run-time.* The Drools engine provides the capability to define tests as rules, and test them against the state of the environment, modelled as facts. The Simulation Engine provides the capability to gather state information from the simulated deployment environment. Finally, the support which Drools provides for testing generalised rule definitions enables situation specifications to identify situations without requiring explicit information about the users, entities, locations or doors to be embedded in rules.

5.6 Analysing Ubicomp System Behaviour

Requirement **R7** stated that *structured feedback must be generated, which can be analysed and traced for the purpose of investigating the causal factors leading to specific situations.* This section illustrates how an Alert Report can be systematically processed and analysed to produce charted views of the test results. Listing 5.2 provides an example of a single alert from an Alert Report. The details and information contained in this alert are easily deciphered, and include the situation ID, location, occupant ID and timestamp. However, during the case study simulations, which will be presented in the next chapter, very large Alert Reports were generated which contained in the order of tens of thousands or alerts. It is not feasible for a tester to interpret meaningful data from XML files of this size without some additional processing.

Listing 5.2: Example Output.

```
<alert id="1" timestamp="1267391637360">
  <rule id="ID.1: Lights Off for User" />
  <location>fs-corridor-1</location>
  <user>occupant1</user>
</alert>
```

To provide more intuitive views of this data, it is processed and summarised in charts. Microsoft Excel was selected for this task because it has built in functions to parse XML files directly to a spreadsheet format. It is also a widely used tool for data processing and data analysis. After an Alert Report has been converted to spreadsheet format, the tester can perform data processing and analysis on the results. Scatter plot representations of the information were chosen in order to preserve the time-line, due to the uneven temporal distribution of alerts. Figures 5.8, 5.9, 5.10, 5.11, 5.12 and 5.13, on the next pages, illustrate example views that can be generated by charting data from an Alert Report and test log files.

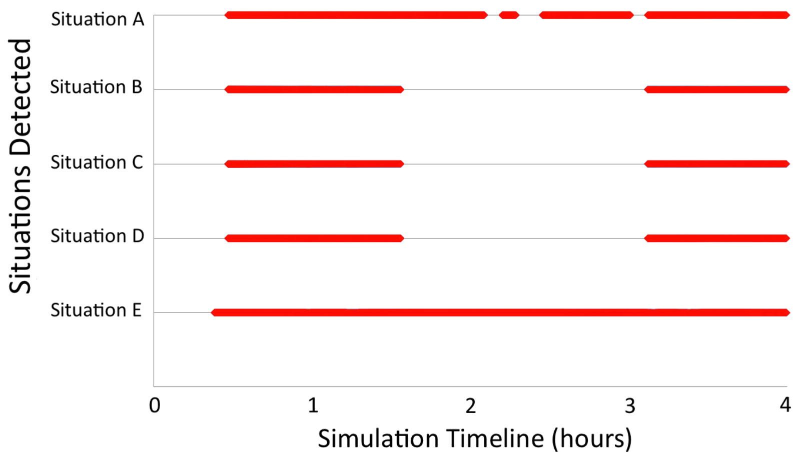


Figure 5.8: A charted view of an Alert Report illustrating detected instances of a set of five situation specifications.

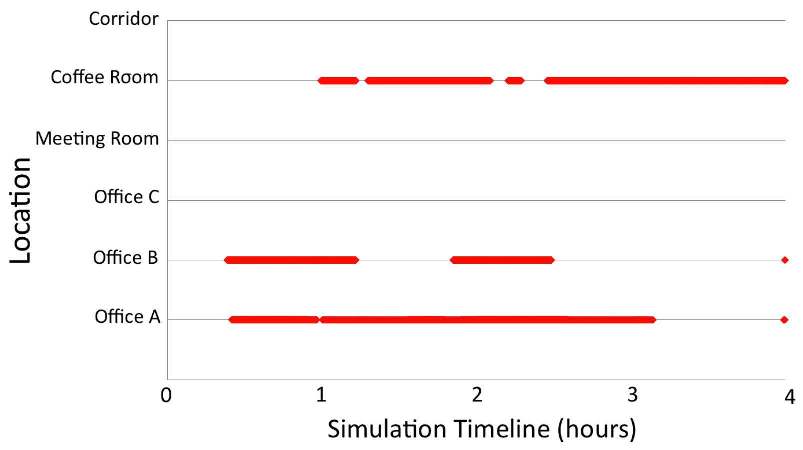


Figure 5.9: A view from an Alert Report that charts *Situation E* from Figure 5.8.

Figure 5.8 shows the results from the first six hours of a simulation that tested five situation specifications. Each situation is charted on a horizontal line over the timeline of the simulation. Although the results appear as continuous bars, each situation is actually plotted as a discrete point. The continuous lines are a product of density and points, and the plot point size. Figure 5.9 charts the times and locations where the instances of situation specifications were detected. In Figure 5.8, it appeared that *Situation E* occurred continuously for over 3.5 hours during this test. However, Figure 5.9 shows that these results are actually comprised of situational instances that occurred in three different locations, the coffee room, office B and office A.

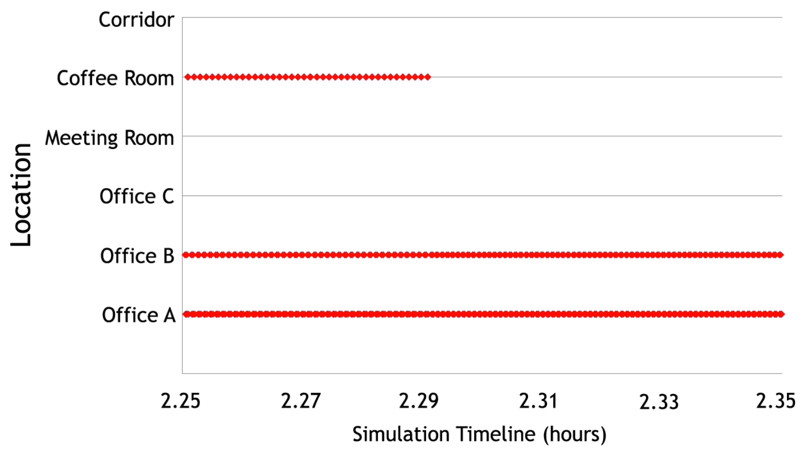


Figure 5.10: This chart illustrates a zoomed view of figure the Alert Report in 5.9.

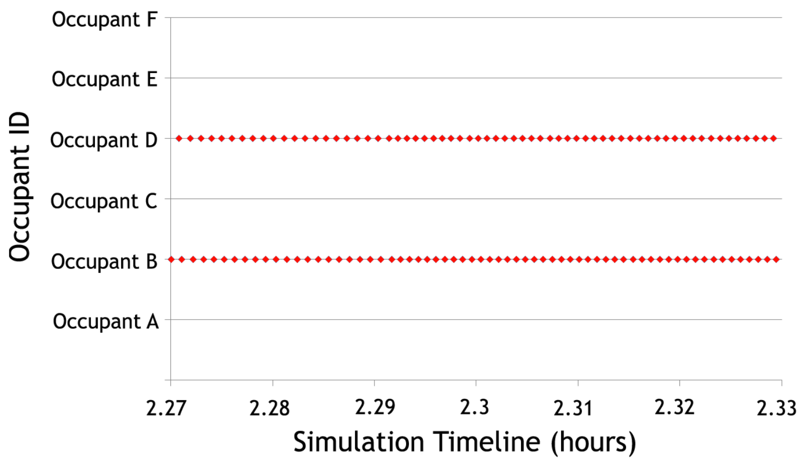


Figure 5.11: A view of the data line *Office B* from Figure 5.10.

Figure 5.10 shows a chart that zooms in on Figure 5.9 to demonstrate that the continuous lines appearing in the charts up to this point, are in fact comprised of discrete points. This chart focuses the view of that data to only show the test timeline between 2.25 hours (2 hours 15 minutes) and 2.35 hours (2 hours 21 minutes), i.e. a six minute window of time. Figure 5.11 provides a view of the data line *Office B* from Figure 5.10. It can be seen in Figure 5.10 that *Office B* has a more dense level of alerts than *Coffee Room* on the same chart. This is because more than one occupant experienced this *Situation E* at the same time in *Office B*. This chart shows that *Occupant B* and *Occupant D* experienced *Situation E* at the same time in the same location, i.e. office B.

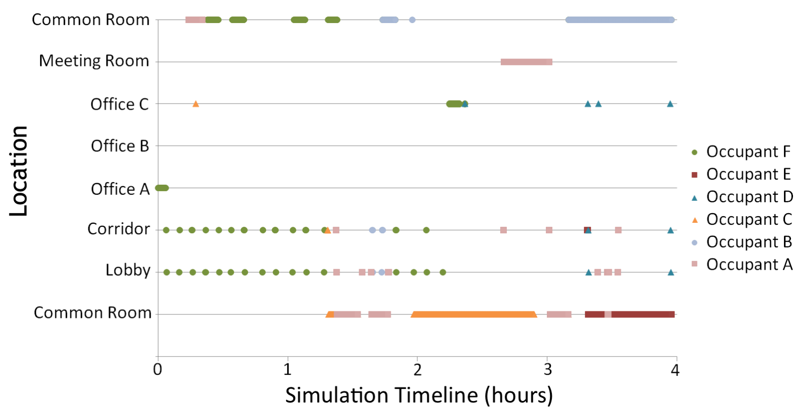


Figure 5.12: Trace of Occupant Locations.

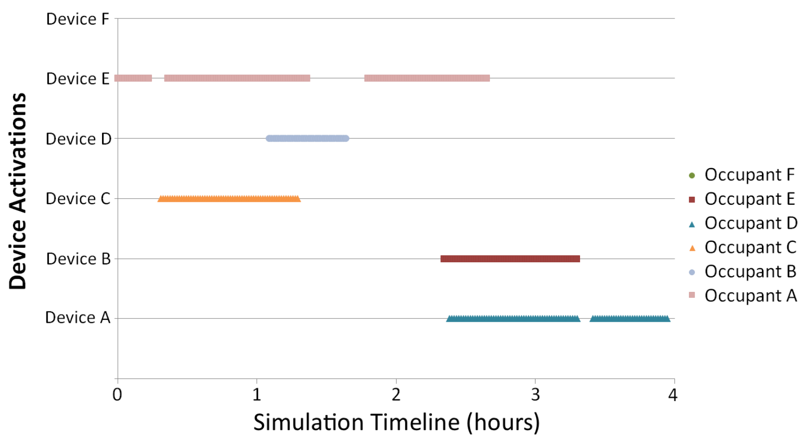


Figure 5.13: Trace of device activations and the occupants who triggered the action.

Figures 5.12 and 5.13 illustrate example views that can be generated from the Update Router's log file. Figure 5.12 charts the trace of occupant locations over the timeline of a simulation. In location-tracking systems, particularly systems in which the user does not require a mobile device, users are physical causal factors for situational outcomes. Figure 5.13 charts the record of device activations by specific users. Device activations provide a digital trail of causal factors that can be traced in the environment.

5.6.1 Test Analysis Summary

This section discussed analysis of the Alert Report with a view to satisfying requirement **R7**, which stated that *structured feedback must be generated, which can be analysed and traced for the purpose of investigating the causal factors leading to specific situations.*

This section discussed how a series of XML encoded alerts, saved in the Alert Report, can be parsed by Microsoft Excel to spreadsheet format. Using the spreadsheet, data processing and analysis can be used to convert the data into scatter plot charts. Scatter plots preserve the timeline of the tests, which can enable the tester to reconcile findings between the different views of data that can be generated. The examples presented here included (i) a chart of the times each situation was detected, (ii) the locations in which a situation was detected, (iii) zoomed views of charts to gain finer granularity views of the timeline, (iv) the bots that were affected by a situation, (v) the locations of all bots during a simulation, and (vi) the device activations that occurred during a simulation. By reconciling these views, the tester can gain insight into the circumstances surrounding inappropriate situations, and investigate both physical and digital factors leading to specific situations, e.g. a user changing location, or the activation of a sensor.

The method of data processing and charting the structured feedback in an Alert Report, provides testers with a process for analysing the results of a simulation to determine the causal factors leading to inappropriate situations.

5.7 InSitu Test Process Implementation

Figure 5.14 illustrates the implementation of the InSitu Test Process defined in Chapter 4. Where Chapter 4 provided an abstract diagram of the test process, Figure 5.14 shows how the process has been realised in the Toolset, and the roles of individual components and resources. Dashed arrow lines indicate data flows, while solid arrow lines indicate program or execution flow. To support a clear mapping between design and implementation of this test process, the coding for the steps in this process have been reused from Figure 4.6, i.e. ITP 1, ITP 2 and ITP 3.

Prior to a test cycle commencing, the designer develops a computer-based interactive prototype of the ubicomp application. This is illustrated in the *Design - Develop* horizontal pool. The *Configuration* pool shows the configuration phase. This involves creating the model instances of the InSitu Model Framework, and the state initialisation XML file. The tester must also create the configuration file for the Update Router during this stage.

The *Testing* pool illustrates the start of a simulation execution which begins when the tester starts the software processes running. The Update Router must be invoked first because it is responsible for managing the testing phase. If RMI is being used, the RMI server should also be running at this point. The Update Router maintains an open port to receive new connections so that more than one ubicomp system can join the test. The ubicomp system is invoked next and it establishes a connection to the Update Router, providing it with the location of the test configuration file. This file provides the Update Router with the necessary information to establish a connection with the Simulation Engine. The Simulation Engine can be started using RMI or manual invocation.

When the Update Router has established a connection with the Simulation Engine, the tester can invoke the Monitoring Engine. The Monitoring Engine connects to the Update Router using the same interface as the ubicomp system. The Monitoring Engine must provide its own test configuration file that includes its data subscriptions. The Monitoring Engine looks up the location of the Model Framework in the test configuration file and loads the models to populate the Drools Facts State Model and the Drools Rules for the testing phase.

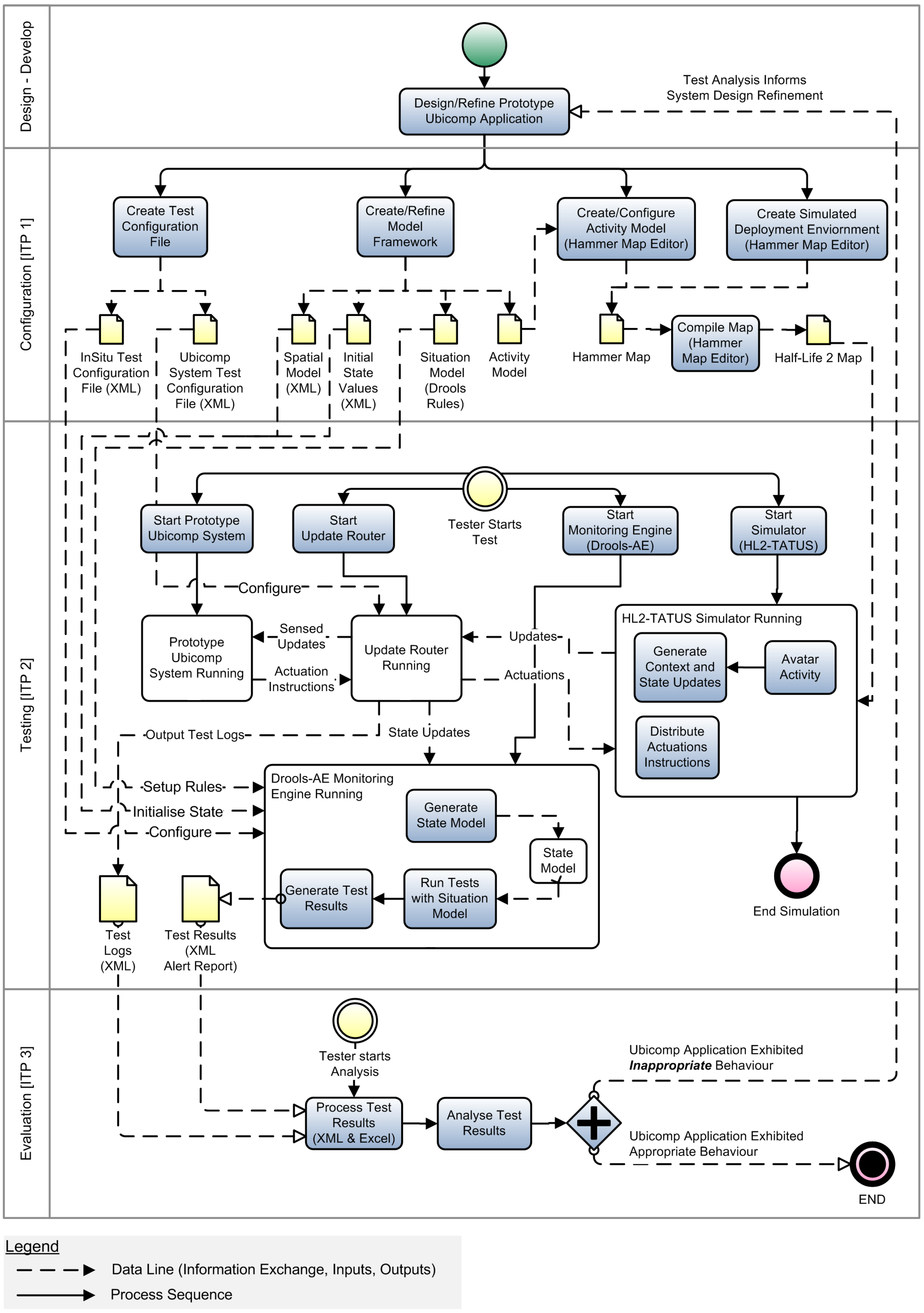

Design - Develop
Configuration [ITP 1]
Testing [ITP 2]
Evaluation [ITP 3]
Design/Refine Prototype Ubicomp Application
Test Analysis Informs System Design Refinement
Create Test Configuration File
Create/Refine Model Framework
Create/Configure Activity Model (Hammer Map Editor)
Create Simulated Deployment Enviornment (Hammer Map Editor)
InSitu Test Configuration File (XML)
Ubicomp System Test Configuration File (XML)
Spatial Model (XML)
Initial State Values (XML)
Situation Model (Drools Rules)
Activity Model
Hammer Map
Compile Map (Hammer Map Editor)
Half-Life 2 Map
Start Prototype Ubicomp System
Start Update Router
Tester Starts Test
Start Monitoring Engine (Drools-AE)
Start Simulator (HL2-TATUS)
Configure
Prototype Ubicomp System Running
Sensed Updates
Actuation Instructions
Update Router Running
Updates
Actuations
HL2-TATUS Simulator Running
Generate Context and State Updates
Avatar Activity
Distribute Actuations Instructions
Output Test Logs
State Updates
Setup Rules
Initialise State
Configure
Drools-AE Monitoring Engine Running
Generate State Model
State Model
Generate Test Results
Run Tests with Situation Model
Test Logs (XML)
Test Results (XML Alert Report)
End Simulation
Tester starts Analysis
Process Test Results (XML & Excel)
Analyse Test Results
Ubicomp Application Exhibited *Inappropriate* Behaviour
Ubicomp Application Exhibited Appropriate Behaviour
END
Legend
Data Line (Information Exchange, Inputs, Outputs)
Process Sequence

Figure 5.14: Implementation of InSitu Test Process

The *Testing* pool also illustrates the operation of a running simulation. During this phase, both context and state updates are being generated, and the ubicomp application is issuing actuation instructions. The Update Router is routing messages between the Simulation Engine, the ubicomp application and Monitoring Engine. The Monitoring Engine is examining the state of the environment, testing it for the specified situations and logging the details of any instances found. At run-time the Update Router maintains a log of the test. This is used to supplement the Alert Report during the evaluation phase.

The *Evaluation* pool illustrates the final stage of a testing cycle in which the feedback in the Alert Report is analysed by the tester. If the tester finds instances of inappropriate behaviour that warrant further investigation or refinement of the design, the Test Process iterates back to the start again. Design refinement might include changes to either the decision logic in the system or to the configuration of the deployment environment. If changes are made to the latter then the tester may also need to refine the Model Framework definition. If no refinement or further investigation is required, the Test Process exits the cycle.

5.8 Toolset Performance

This section presents some performance results recorded for the Toolset implementation. These results are presented to demonstrate that the implemented Toolset has been sufficient for the evaluations discussed in the next chapter. The results were recorded using commodity laptop and desktop hardware as follows:

- The laptop's hardware included an Intel Core 2 Duo CPU T7700 @ 2.40GHz, with 3.5GB of RAM, and a NVIDIA GeForce 8600M GT graphics card. The laptop was running Microsoft Windows XP Professional[10] Version 2002 Service Pack 3.

- The desktop's hardware included an Intel Core 2 CPU 6700 @ 2.66GHz, with

[10]Windows XP Professional http://www.microsoft.com/windowsxp/pro/default.mspx.

2GB of RAM, and a NVIDIA GeForce 7900 GS (256MB) graphics card. The desktop was running Microsoft Windows XP Professional Version 2002 Service Pack 3.

5.8.1 Message Processing

Results for the performance of the Update Router and the Monitoring Engine are reported in terms of routing and processing messages respectively. Recorded timings show that the Update Router takes on average $3ms$ to route each message. This number was calculated using a sample of 300 messages. The Monitoring Engine was found to process state updates in an average time of $1.9ms$, over a sample size of 300 messages. These results were recorded on the laptop computer detailed previously.

5.8.2 Testing Environment State

This section reports on the performance of the Monitoring Engine and the time taken to run a set of situation specification rules, as well as to output alerts when these rules fire.

Initial results gathered recorded timing for a single bot simulation, feature 31 Drools facts to represent the State Model, and one rule representing situation specifications. This set of results was recorded on the laptop computer with the full InSitu Toolset running on it, i.e. Simulation Engine, Update Router and Monitoring Engine, as well as the ubicomp service under test. Over the course of an hour, running the rule once per second, time taken to assess the state of the environment averaged $150ms$, with a maximum of $340ms$, and a minimum of $40ms$. These calculations omitted the initial loading time for the rulebase, because it is a one off event, which took $6.2s$.

A second set of results recorded timing for a larger scale simulation. This set of results was recorded using two computers to distribute the processing load for the InSitu Toolset across more than one machine. During these tests, measurements were recorded on the desktop computer, which was host only to the Update Router, Monitoring Engine and ubicomp service. This configuration made use of the TCP/IP connection to run the Simulation Engine on a separate machine. This test configuration featured 83 facts, including 20 bots, 46 zones, and 17 entities, tested against six situation specifications. Over the course of an hour, running the rules once per second, time taken to assess the state of the environment averaged $67ms$, with a maximum of $275ms$ and a minimum of $61ms$. These calculations omitted the initial loading time for the rulebase, because it is a one off event which took $1.17s$. During this test, 1716 alerts were generated, an average of 28.6 per minute.

A third set of results recorded timings for a test which artificially increased the load on the Monitoring Engine, both in terms of examining environment state and outputting alerts to the Alert Report. The configuration for this test featured 50 bots, 80 facts in total, and tested six situation specifications. Over the course of three minutes, running the rules once per second, 413 alerts were generated and output to the Alert Report, an average of 137.6 alerts per minute, significantly higher than the previous performance tests. At this load, the Monitoring Engine averaged a rule cycle of $0.14s$, and did not exceed $0.4s$. This set of results were recorded on the laptop computer, with only the Monitoring Engine running. This was an artificially contrived test, specifically to increase the load on the Monitoring Engine, and so a dataset of messages was used to create this artificial test set up.

It was expected that the times recorded for the first performance test should be lower than the times recorded for the second set of performance results, however this was not the case. It is likely that the extra processing load on the laptop, because it was running the full Toolset, impacted the performance times. In particular, the game engine would have placed a heavy drain on system resources and processing power.

5.8.3 Performance Summary

The case studies, presented in the next chapter, test the situation specification rules once per second. The upper limit, in terms of the scale of testing required for the case studies, is 20 users, 46 zones and 17 entities, totally 83 facts, tested against six situation specifications. The results presented here show that this Toolset implementation is satisfactorily efficient as a tool to conduct these case study evaluations.

5.9 Chapter Conclusions and Summary

The Toolset implementation presented in this chapter was developed to provide the mechanism for conducting the evaluation of the situation-based testing approach. The Toolset realises an implementation of both the InSitu Model Framework and the InSitu Test Process, which together provided the definition of the proposed situation-based testing approach. The implementation integrates and extends a set of existing technologies, which includes the TATUS Simulator and Proxy, the Drools Engine, the Adaptive Engine, and XML information encoding, in order to create the Toolset.

This chapter described how the Toolset components each address one of the technical requirements set out for the Toolset in the last chapter. The technical requirements were defined to expand on objective **O4**, which was stated as *'To realise the framework and test process in a simulation-based test-bed, as a mechanism for evaluating the situation-based testing approach'*. As a whole, the implementation of the Toolset addresses research objective **O4**. Performance testing conducted for the Toolset showed that it is sufficiently efficient to act as an instrument for the evaluations presented in the next chapter.

Chapter 6

Evaluation

6.1 Introduction

At the outset of this thesis a situation-based testing approach was proposed as a method for testing the summative effect of system and user behaviour in a simulated ubicomp deployment. This is important for ubicomp systems because the effects of exhibited system behaviour are not always bounded in the digital environment. The InSitu Model Framework, Test Process and Toolset provide a design and implementation that realise the situation-based testing approach. The evaluations presented in this chapter aim to assess the suitability of the situation-based testing approach and the expressiveness of the InSitu design and implementation.

A real-life case study investigates whether InSitu can be used to correctly identify examples of real world inappropriate situations. A second case study features a hypothetical, sensor-driven, access-control system, designed for a real-world office space to investigate the expressiveness of the Model Framework. A user trial presents an objective assessment of the situation-based testing approach, both in terms of the expressiveness of the Model Framework and the suitability of the testing approach. Finally, the comparison framework, which was defined to summarise the state of the art review, is revisited as an instrument to compare InSitu to the reviewed tools.

6.2 Case Study 1: Motion Sensor Driven Lighting

The central objective for this case study is to investigate whether InSitu can be used to correctly identify examples of real-world inappropriate situations. This places emphasis on assessing (i) whether the Spatial Model can represent a real-world building, (ii) whether the Situation Model can represent inappropriate situations observed in a real-life environment, (iii) whether generalised situation specification can be applied at a global level to identify instances of situations in the simulated environment.

6.2.1 Case Study Description

This case study examines a commercial lighting system deployed in a building in Trinity College Dublin. This case study was chosen because it provides a real-world example of behaviour exhibited by a sensor-driven system causing inappropriate situations to arise in the physical deployment environment. A further benefit of examining a real world case study is that it was possible to survey the occupants of the building to inform the case study and in particular the Activity Model. The survey was conducted using Survey Monkey[1] to gather responses, and as such respondents were not supervised while they completed the questions. Of the 50 permanent occupants on the first floor 18 surveys were collected 14 of which were complete (28% of occupants).

Two inappropriate situations were observed in the building. The first affects occupants, who suffer during times of low occupancy because the lights can fail to turn on in communal zones of the building, if a specific set of circumstances arise. The second affects the facility manager, who faces a problem in terms of energy management, because due to the deployment configuration of the lighting system, the lights can remain on in the building when no users are present.

[1]Survey Monkey http://www.surveymonkey.com/ (accessed 12th December 2010).

Lighting System Description

The lighting system is deployed indoors in the communal areas of a multi-storey office building. The system is designed to automatically switch lights on for building occupants when an occupant is present and to switch lights off when no occupants are present. Essentially the system performs a simple action of toggling the lights between on and off states, based on user presence, detected by motion sensors.

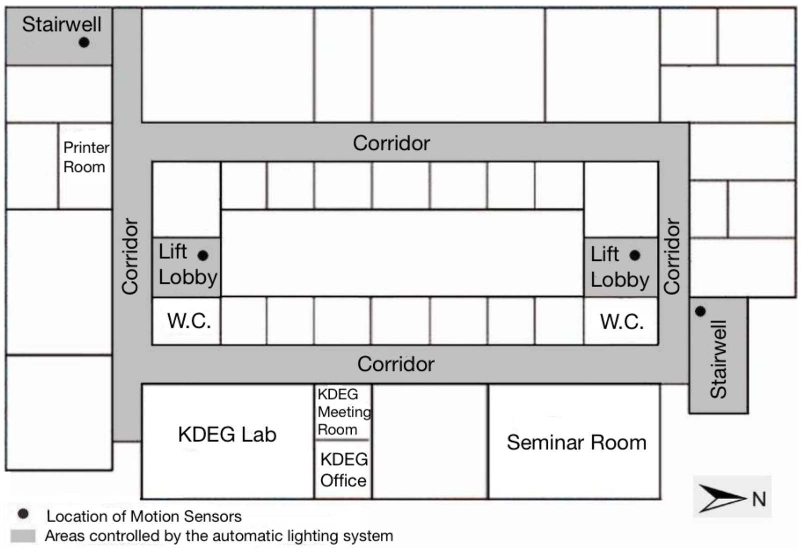


Figure 6.1: Floor-plan for the 1st level of the building central to the lighting case study. This building has a floor area of 8440m^2.

The floor plan for the 1st level of the building, featured in this case study, is illustrated in Figure 6.1, which has a floor area of 8440m^2. The entry points to the floor lie at opposite ends of the oblong building, i.e. at the stairwells and lifts. The remaining space surrounding the atrium and corridor is made up of offices, lecture rooms and shared lab spaces for postgraduate students.

Motion sensors are installed at the entry points for each floor, in the lift lobbies and stairwells. An in-built timer in the system controls switching the lights off. Lights in the corridor zone on each floor are all connected to a single circuit. Similarly, the lights in each stairwell are all connected on a single circuit. In private zones of the building, such as offices and labs, wall mounted manual light switches are installed so occupants can control lighting in those areas.

During times of low activity in the building, situations arise whereby the lights in corridors have been turned off while occupants are still present in their offices or in lab spaces. Under these conditions, when an occupant emerges from one of these rooms, they must find their way to the nearest motion sensor in a stairwell or lift lobby in order to switch the lights on, assuming of course that they understand the relationship between sensors and lights. For an occupant who wants to visit another nearby office, it can mean that the occupant must go out of their way to activate the motion sensor, or make the trip in the dark. Without ambient daylight, or even street lighting from the windows, especially in the internal corridors, the building will be in darkness.

The second inappropriate situation occurs after an occupant exits a floor in the building. The motion sensor in the lift lobby or stairwell will reset the timer as the occupant travels through the exit zone. By switching the lights on, and resetting the timer at this point, there is little to be gained for the building in terms of energy conservation. Additionally, if the occupant has just been experiencing the first inappropriate situation, no benefit has been gained at all because the occupant will have made no use of the lighting.

6.2.2 Modelling the Lighting System

This section discusses how this lighting system is modelled using the Spatial Model, Situation Model and Activity Model from the InSitu Model Framework. The case study focuses on the first two floors of the building. This provides sufficient zone coverage to demonstrate the manifestation of the previously described inappropriate situations. The simulated building model, for use with the Half-Life 2 game engine, was reused from another project so it was only necessary to position sensors and configure the Activity Model using the Hammer World Editor.

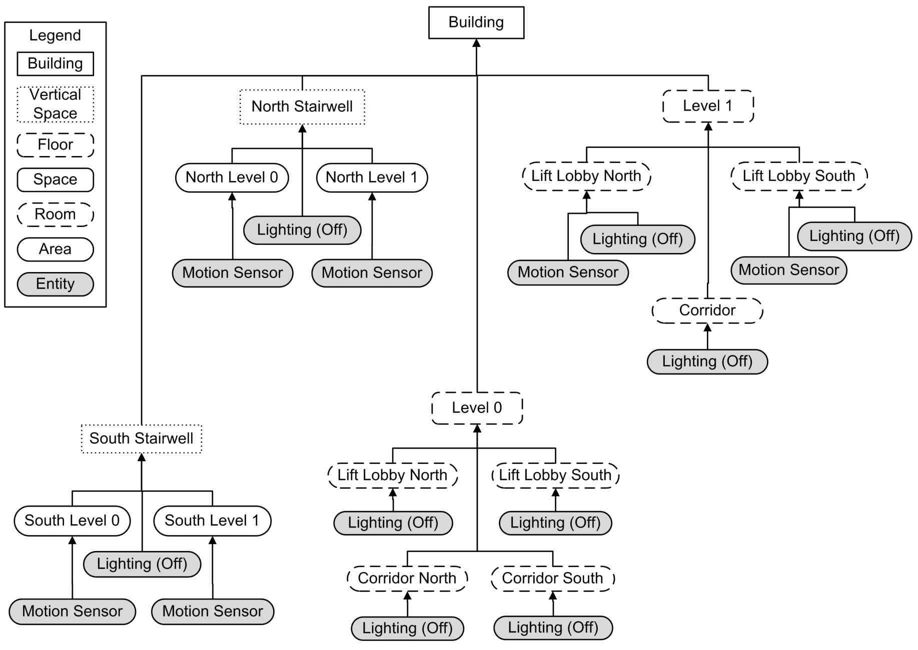


Figure 6.2: Hierarchy of zones and entities in the Lighting System Case Study.

Spatial Model

The hierarchy of zones for the first two floors is illustrated in Figure 6.2, including the location of motion sensors and lighting in the building. The lighting operates on a single circuit in each corridor and stairwell, and so it is represented as wholly contained by those zones. However, occupant location can be more precisely pin-pointed in state updates, by the addition of areas within those zones. This is illustrated for the stairwells, which have areas defined at the door (entry point) for each floor that the stairwell provides access to. The XML encoding for the Spatial Model is lengthy, so for brevity it is included in Appendix A.

Situation Model

This section defines the situation specifications for the two cases of inappropriate situations previously described, i.e. (i) a user is in the dark and (ii) a light is on unnecessarily. For the purpose of brevity, only the logical definition is presented. The Drools encoding for this model is included in Appendix A. The logic for the first situation specification (user in the dark) uses information about an occupant's location to find the lights that are in proximity to the occupant. This situation defines close proximity as a user and light in the same room. The situation's logic also uses information about entity type and state to identify lights that are off. The situation is detected **when** there exists a user, u, and the lights in proximity to u are off.

Situation Specification A.1: *User in the Dark*
if
U $\longleftarrow$ u
$\wedge \nexists$ (E ((type = "light") $\wedge$ (roomLocation = u.roomLocation) $\wedge$ (state = "on")))
then
Output $O(t) \subset G(t)$

The logic for the second situation specification (lights on unnecessarily) uses information about the state of lights and their location, as well as the location information for all users in the space. The situation defines proximity as a user and light in the same room. The situation can be read as occurring **when** there exists a light e, which has a state *on*, and there does not exist any user in the same room.

Situation Specification A.2: *Lights on Unnecessarily*
if
E ((type = "light") $\wedge$ (state = "on")) $\longleftarrow$ e
$\wedge \nexists$ (U (roomLocation = e.roomLocation))
then
Output $O(t) \subset G(t)$

Activity Model

The Activity Model for this case study focused on common office activities, including working-in-office, lunch-break, coffee-break, collect-printouts and attend-meeting.

Occupants of the real-life building were surveyed about their experience of the duration of these activities. Responses were used to inform the Activity Model rather than define it because the collected values were estimates on the part of the respondents. Responses were collected using Survey Monkey[2] and are presented in charted format in this section.

A limitation of using Survey Monkey was that these surveys were unsupervised and so respondents could not ask for clarification on the questions. It appears this led to a number of misleading responses where respondents misunderstood the question being asked. These responses were omitted from the case study to minimise contamination of the data set. However for the purpose of completeness of reporting, these values are included in the charts in this section. Long-term knowledge and experience of the building and office space was used to inform the decision points indicated on the charts in Figures 6.3 and 6.4.

The data-set of occupant responses captured to inform the bounds on duration for the **working-in-office** activity is charted in Figure 6.3. Horizontal red dashed lines indicate the upper (4 hours) and lower (1 minute) bounds on duration used for this case study. These bounds retain the majority of responses and omit the extreme maximum data points that are considered to be misrepresentative of the working-in-office activity.

The data-set of occupant responses captured to inform the bounds on duration for the **lunch-break** activity is charted in Figure 6.4. For this activity it was decided to use values closer to the average working day because the intention was to capture an activity that emulates a lunch-break. In a research environment it is not unusual for occupants to work late, so this activity can also be appropriate for an evening food and refreshment break. Horizontal red dashed lines indicate the upper (2 hours) and lower (5 minutes) bounds on duration for the lunch-break activity. These bounds retain the majority of responses and place emphasis on the bounds of an average break time.

[2]Survey Monkey http://www.surveymonkey.com (accessed 21st April 2011).

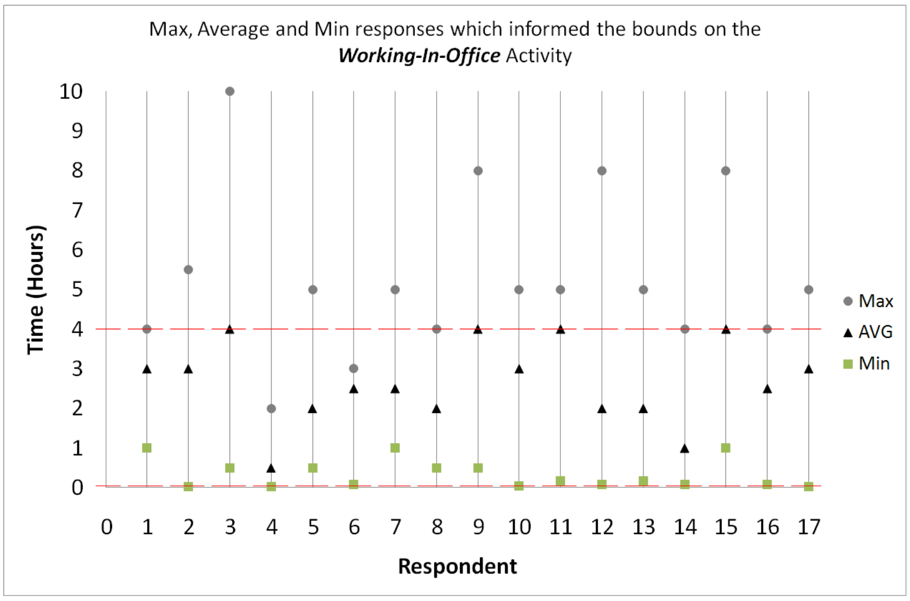


Figure 6.3: The data-set that informed the duration of the **working-in-office** activity. Red dashed lines indicated the upper (4 hours) and lower (1 minute) bounds on duration. Points on the chart illustrate reported minimum, average, maximum durations.

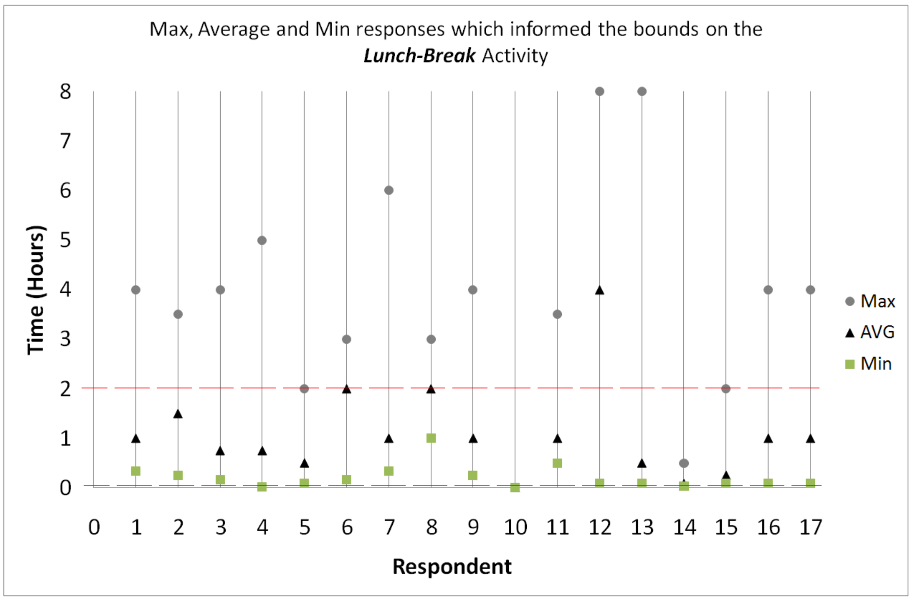
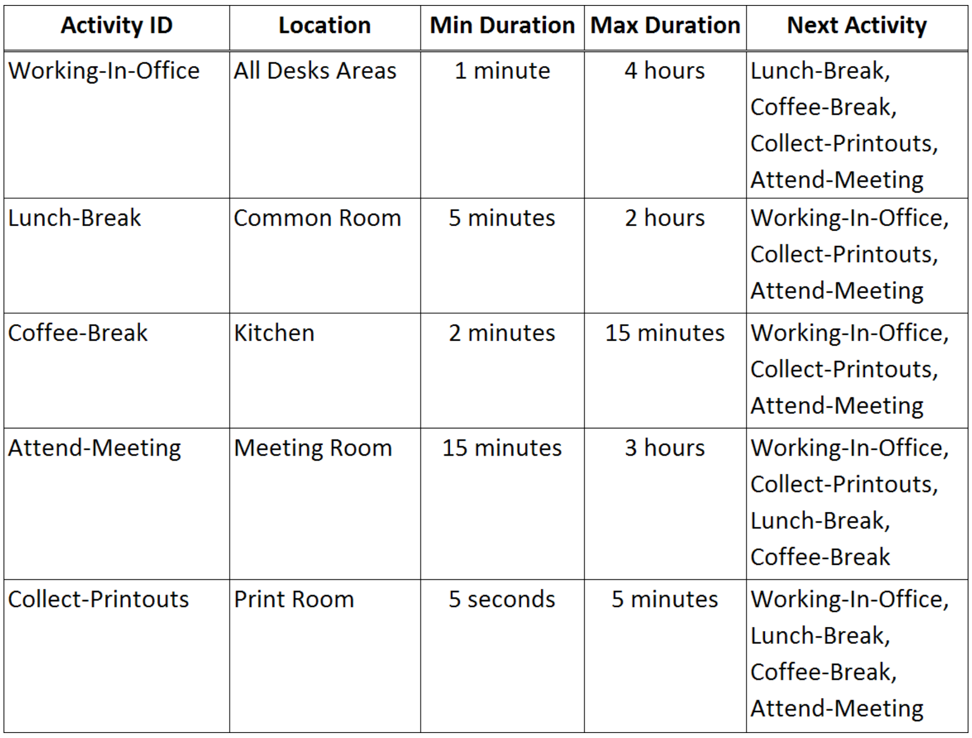


Figure 6.4: The data-set that informed the duration of the **lunch-break** activity. Red dashed lines indicated the upper (2 hours) and lower (5 minutes) bounds on duration. Points on the chart illustrate reported minimum, average and maximum durations.



Activity ID	Location	Min Duration	Max Duration	Next Activity
Working-In-Office	All Desks Areas	1 minute	4 hours	Lunch-Break, Coffee-Break, Collect-Printouts, Attend-Meeting
Lunch-Break	Common Room	5 minutes	2 hours	Working-In-Office, Collect-Printouts, Attend-Meeting
Coffee-Break	Kitchen	2 minutes	15 minutes	Working-In-Office, Collect-Printouts, Attend-Meeting
Attend-Meeting	Meeting Room	15 minutes	3 hours	Working-In-Office, Collect-Printouts, Lunch-Break, Coffee-Break
Collect-Printouts	Print Room	5 seconds	5 minutes	Working-In-Office, Lunch-Break, Coffee-Break, Attend-Meeting

Figure 6.5: Activity Model for the Lighting Case Study.

The remaining activities were not directly informed by the survey:

- The duration of the meeting activity was defined based on experience and by sampling the research group online diary of meeting room bookings. The attend-meeting activity was bounded between 15 minutes and 3 hours.
- The collect-printouts activity was estimated at between 5 seconds and 5 minutes, to allow for paper jams, long print jobs, or serendipitous conversation at the printer.
- The coffee-break activity was estimated to be shorter than the lunch-break activity, and so was set to between 2 minutes and 15 minutes. Although not directly informed by the occupant survey, these values are representative of the shorter durations reported in Figure 6.4

A second part of the occupant survey asked participants to map out paths which they regularly took through the building from start to destination. These start and end points were used to inform the location of activities. These locations complete the Activity Model defined in Figure 6.5. The working-in-office activity can occur in any office. This is an example of an individual activity because each occupant has their own desk. The lunch-break activity was allocated to the common room, and the coffee-break activity to the kitchen. The attend-meeting activity is associated with the meeting room. Although the printer room did not feature in the survey, it is a natural choice to put the collect-printouts activity at the printer.

6.2.3 Simulation Results for Lighting Case Study

The test configuration, presented in this section, is contrived to focus on low occupancy conditions in the building by using a single bot. This approach was taken because both of the examples of inappropriate situations observed in the deployment are closely linked to times of low occupancy. Since this is already known, the single bot configuration enables the case study to focus on the objective of investigating whether InSitu can identify examples of real world inappropriate situations. A secondary benefit of the low occupancy configuration is that a lower number of bots generates a smaller data set at run-time. This is beneficial for evaluating the InSitu approach because it facilitates presentation of a full set of test results in this section, which were produced during a 12 hour test. This is used to demonstrate that simulations can run for extended periods of longer uninterrupted duration than could reasonably be achieved with human testers driving avatars. The results presented in this section provide overlays of the Alert Report on the test log files. Each set of independent charted results are included in Appendix A, Section A.2.

Situation A.1: Occupant in the Dark

The charted results in Figures 6.6 and 6.7 show that this situation was detected shortly before the 3 hour mark, between the 4-5 hour marks, and again shortly after the 7 hour mark. Investigating the first instance of situation A.1 more closely, it can be seen that the Toolset detected that the bot was in the dark in the first floor corridor at approximately 2 hours and 45 minutes into the simulation. Cross-referencing this with the bot location trace, marked by the blue diamonds in Figure 6.6, it can be seen that the bot was indeed present in the first floor corridor at this time. It can also be seen that the bot changed location a number of times in the period of time surrounding this event. In Figure 6.6 it is clear that the bot had been out of range of the motion sensors for at least 2 hours preceding the occurrence of this inappropriate situation. However it is difficult to determine the exact locations and timings surrounding this event due to the density of the data in the chart.

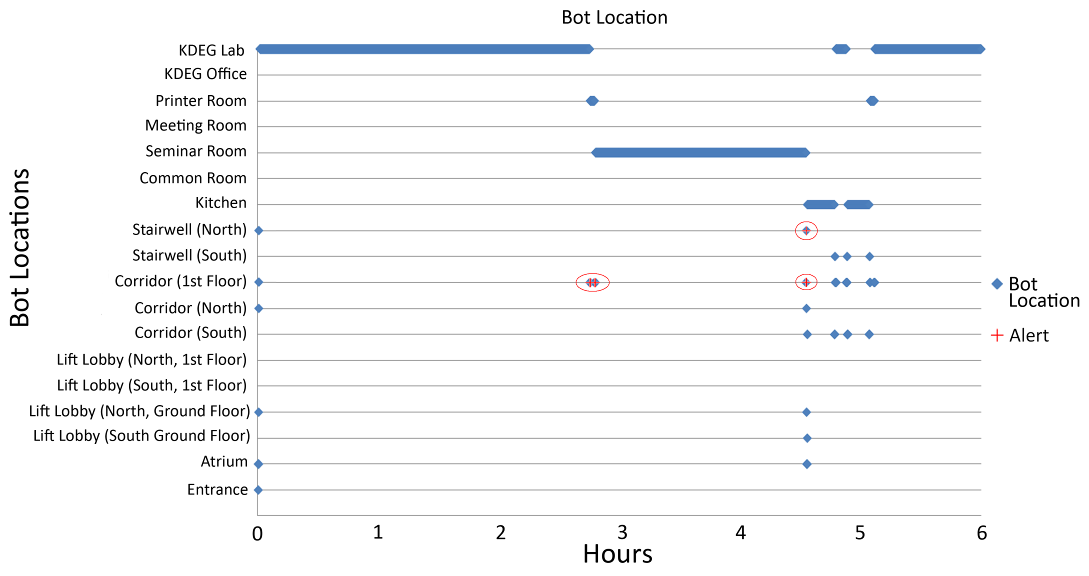


Figure 6.6: Situation A.1: bot in the dark (Hours 0-6).

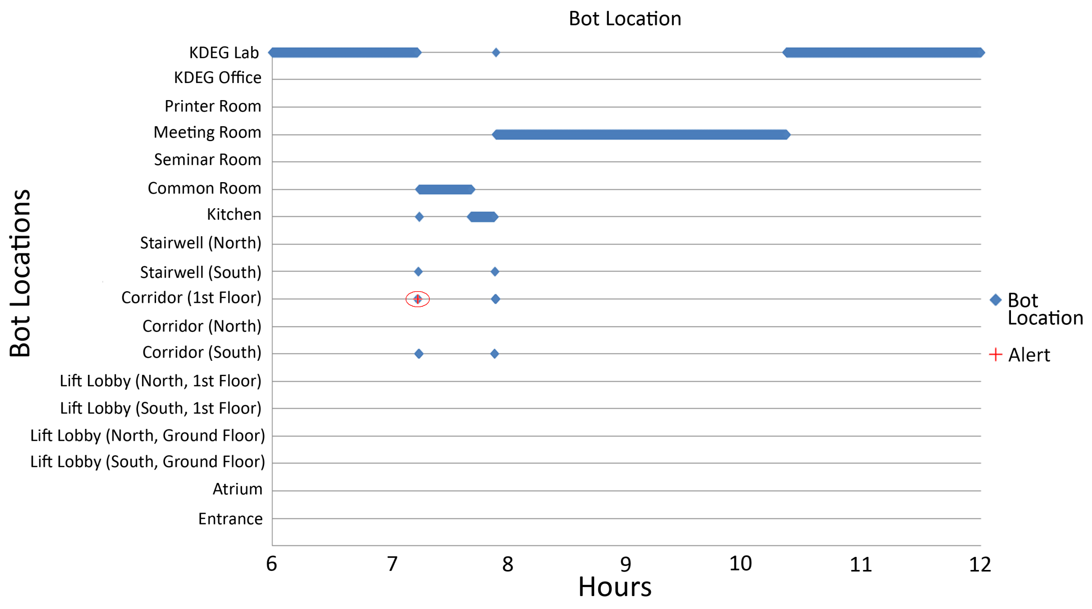


Figure 6.7: Situation A.1: bot in the dark (Hours 6-12).

Figure 6.8 shows greater detail of this time period by zooming in on the timeline (x-axis) to show the period between 2 hours 30 minutes and 3 hours. Based on this it is possible to more easily trace the sequence of events leading to the bot emerging from the KDEG Lab into the dark corridor. Figure 6.8 shows that actually two separate instances of situation A.1 occurred. The first when the bot exited the KDEG Lab to go to the printer, the second when the bot exited the printer room. Figure 6.9 documents the spatio-temporal trace of the bot's activity at the time that situation A.1 was detected.

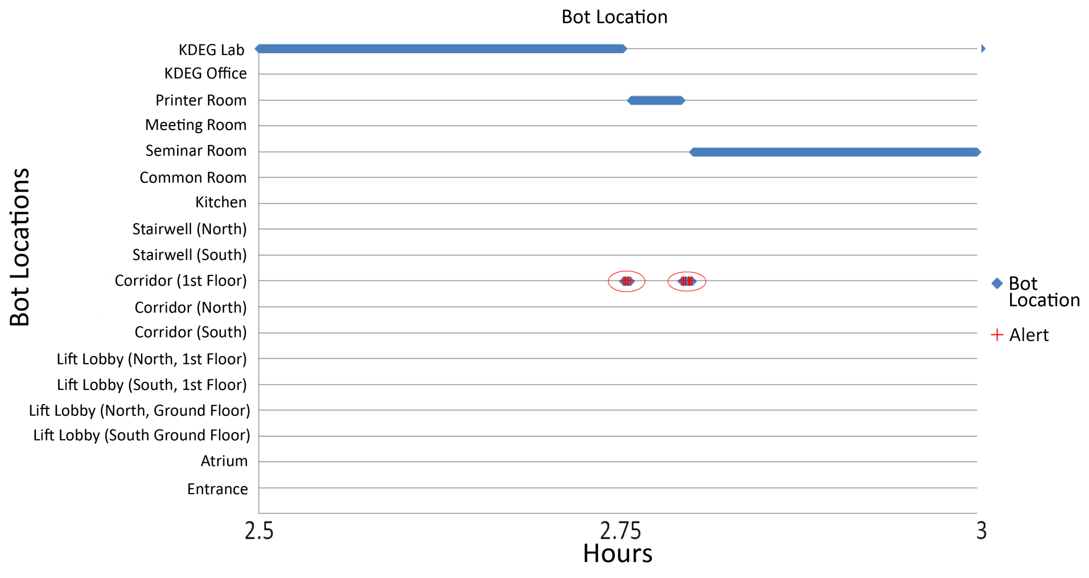


Figure 6.8: Situation A.1: Detected instances between 2.5 hours and 3 hours

Location	Duration	Detected Situations
KDEG Lab	164.08 minutes	None
Corridor 1st Floor	17 seconds	A.1: Occupant in Dark
Printer Room	2.09 minutes	None
Corridor 1st Floor	27 seconds	A.1: Occupant in Dark
Seminar Room	104.71 minutes	None

Figure 6.9: Trace of bot activity around to the two detected situational instances illustrated in Figure 6.8.

Situation A.2: Lights on Unnecessarily

For energy saving schemes the absence of users is equally important as their presence. Ideally the lighting system should be conserving energy by switching off when users are not present. Situation specification A.2 is concerned with identifying times when energy is potentially being wasted. Figures 6.10 and 6.11 show the times and locations that lights were on and no occupants were present. It can been seen that for this test configuration, this inappropriate situation exists extensively in the building. All of the locations which are controlled by the automatic lighting system are affected by this problem i.e. the stairwells, corridors and lift lobbies.

Used in this capacity, the situation-based testing approach can alert facility managers to circumstances in which energy conservation could be improved. Even though the Toolset implementation does not return a kW analysis of power wasted, it clearly highlights that an extensive amount of energy can be wasted during times of low occupancy in the building.

Optimisations for energy saving purposes can be made in the system through the addition of relatively few additional motion sensors. By increasing the number of sensors from 4 to 24, it was found that every doorway, leading to a communal area, on the first floor fell into the shadow of a sensor. Motion sensors are available at relatively low cost, certainly comparatively low relative to the annual electricity cost[3] that needs to be covered by a university. The additional motion sensors have the benefit of allowing the system to use a shorter timer cycle. The installation of motion sensors at more regular intervals means that ongoing activity in communal space will reset the timer to prevent the lights switching off.

[3]The energy cost for the building featured in this case study, both gas and electricity, in the academic year 2007/08 was €483,100. Source: http://www.fp7ireland.com/cms/Documents/PPP Ireland June 11 TrinityHaus_1261.pdf (accessed 3rd November 2010).

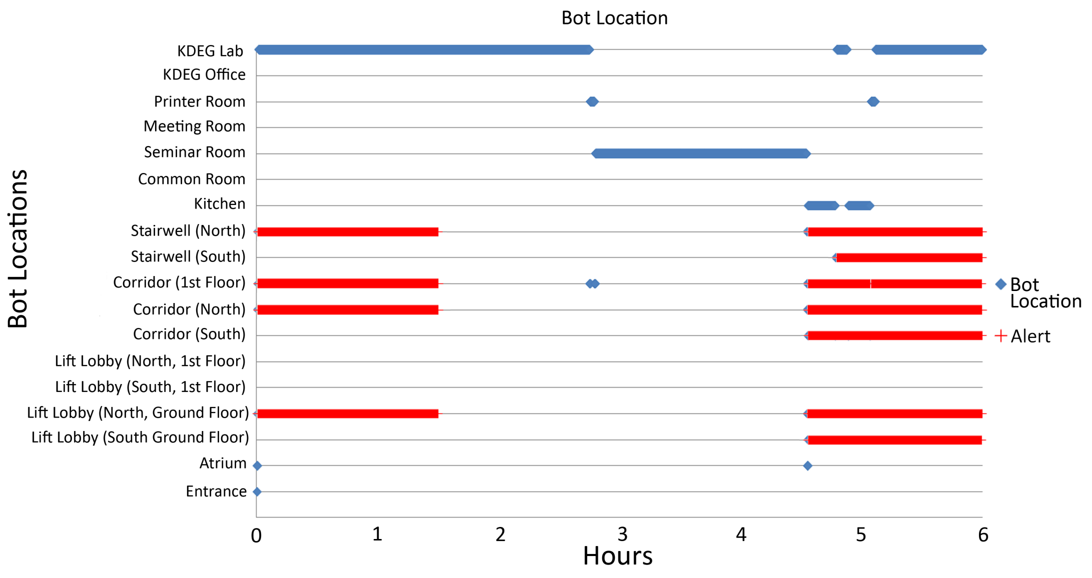


Figure 6.10: Situation A.2: Lights on unnecessarily (Hours 0-6).

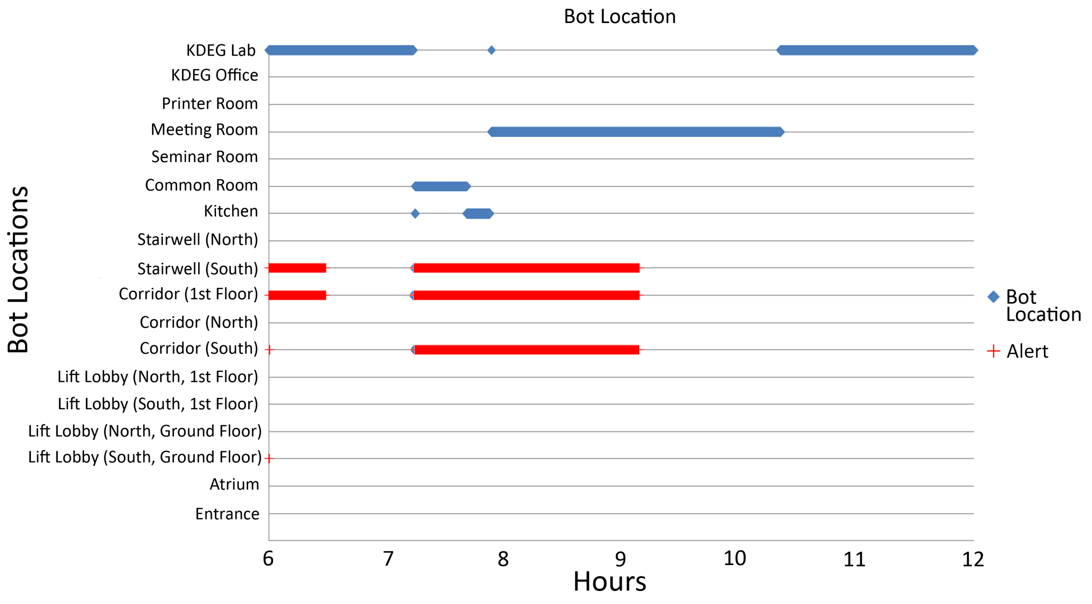


Figure 6.11: Situation A.2: Lights on unnecessarily (Hours 6-12).

6.2.4 Case Study 1: Findings

The lighting system case study provides a real example of an indoor sensor driven system that features an invisible interface and a spatio-temporal dependency between users and the physical environment. Although this case study is a simple example of a problematic system, it clearly illustrates how, even in a straightforward system, it is possible for problems to creep into the design. This case study was chosen for the initial evaluation specifically to demonstrate that the implementation of the defined Model Framework and Test Process can correctly identify examples of real-world inappropriate situations which manifest in the physical environment and are caused by the behaviour of a sensor-driven system.

Two inappropriate situations were observed in the case study system, one occupant-centric (user in the dark), the other environment-centric (lights on unnecessarily). In the first of these situations, each individual occupant is only concerned with the lights in their own immediate area. This requires only localised testing of the environment relative to the occupant's position. The second of these situations focuses on the role of a facility manager whose concern is monitoring the performance of the whole building. This requires a situation specification that tests all areas of the building controlled by the ubicomp system, regardless of occupants' locations. Figures 6.6, 6.7, 6.10 and 6.11 presented charted results which demonstrate that instances of both types of situation were detected and identified during testing.

Additionally, the benefit of the generality of situation specifications was demonstrated by showing that they can be used to monitor the a global view of environment state. The charts in Figures 6.10 and 6.11 illustrate that the testing process identified inappropriate situations in multiple locations concurrently.

Feedback is provided through the combined analysis of the Alert Report and the InSitu log files. Together these files provide times that inappropriate situations are detected during a simulation along with a trace of occupant activities and paths during the simulation. This information assisted particularly in deciphering the sequence of events surrounding the situation detected in Figure 6.6. Although this chart only provided enough information to confirm that an instance of situation A.1 had been detected, the finer granularity views in Figure 6.8 provided more detail. These charts illustrated that the lack of activity in the building combined with poorly positioned motion sensors led to a situation where the bot travelled through the corridor in darkness.

A secondary finding from these simulations is that the Toolset can perform simulations of longer durations than can reasonably be conducted when testing with user-controlled avatars. First person control of a simulation suffers from regular breaks in continuity to refresh the test-user. The generality of situation specifications, using the Model Framework, enable simulations to run unsupervised. The Activity Model provided sufficient bounded randomness to generate dynamic bot behaviour. The automatic generation of the State Model provides the testable view of the state of the deployment.

This case study presented twelve hours of continuous situation monitoring for a simulated sensor-driven system deployment. The results are taken from a simulation that was executed overnight and that was unsupervised. This demonstrates that the approach and implementation support time scalable testing, at least exceeding the duration that can reasonably be achieved when human interaction or human monitoring is required.

6.2.5 Case Study 1: Summary

In summary, this case study demonstrated that the Toolset implementation of the situation-based testing approach, can correctly detect real world inappropriate situations for a spatio-temporally aware, sensor-driven system. The case study demonstrated that the Toolset correctly monitored a global view of environment state at run-time to identify both occupant-centric and environment-centric problematic situations.

It was shown that the generality of the Situation Model enables the tester to define situation specifications that can be used for testing throughout a simulated deployment. The attributes of the State Model were sufficient to complete the situation specifications for this case study. The Activity Model generated instances of both types of inappropriate situations at run-time, using the random model, bounded in the spatio-temporal domain. Finally, the containment relationship of the Spatial Model was used to define the hierarchy of the environment. The proximity relationship, which the Spatial Model supports, was used in the situation specifications to determine the proximity of users to lights.

In the next section, a second case study is discussed which builds on these findings. An access control case study is examined, which increases the complexity of testing by introducing multi-bot testing in an interconnected office-space that features operational zones. The access control case study also goes further to demonstrate the generality of the framework by reusing the situation specifications from the lighting case study, and applying the tests to a different simulated physical deployment.

6.3 Case Study 2: Sensor-Driven Access Control System

This section presents an access control case study that builds on the findings from the previous lighting system case study. The previous case study demonstrated that the InSitu Toolset correctly identified real-world examples of two inappropriate situations, using a simulation of the deployment. The lighting case study provided an example of a single bot simulated configuration, which tested the immediate physical location of both bots and lights. This second case study demonstrates that the InSitu Toolset can assess a multi-bot simulation configuration in a deployment space comprised of both physical and operational environment zones.

The objective set out for this case study is to demonstrate that the InSitu testing approach is sufficiently generalised for the purpose of (i) testing across multiple users, locations and entities and (ii) reuse of situation specifications across different deployment environments and simulation configurations. This case study also aims to demonstrate that InSitu can test both physical and operational spatial zones.

6.3.1 Deployment Environment Description

During the period Nov 2008 to May 2009 a number of security vulnerabilities were observed in a real office-space, in Trinity College Dublin. At the time, the offices were controlled by a system of manual locks and keys. The office space is comprised of four interconnected rooms, including three offices and a coffee room. Adjoint, but not connected to the space is a meeting room. These five rooms form a rectangle which is surrounded by a public corridor. Locations outside the office space are of interest primarily because they impact on user activity patterns. These locations include the common room, the corridor and any place outside the building. The layout of this space is illustrated in Figure 6.12.

A survey of occupants and frequent visitors to the space, was conducted through

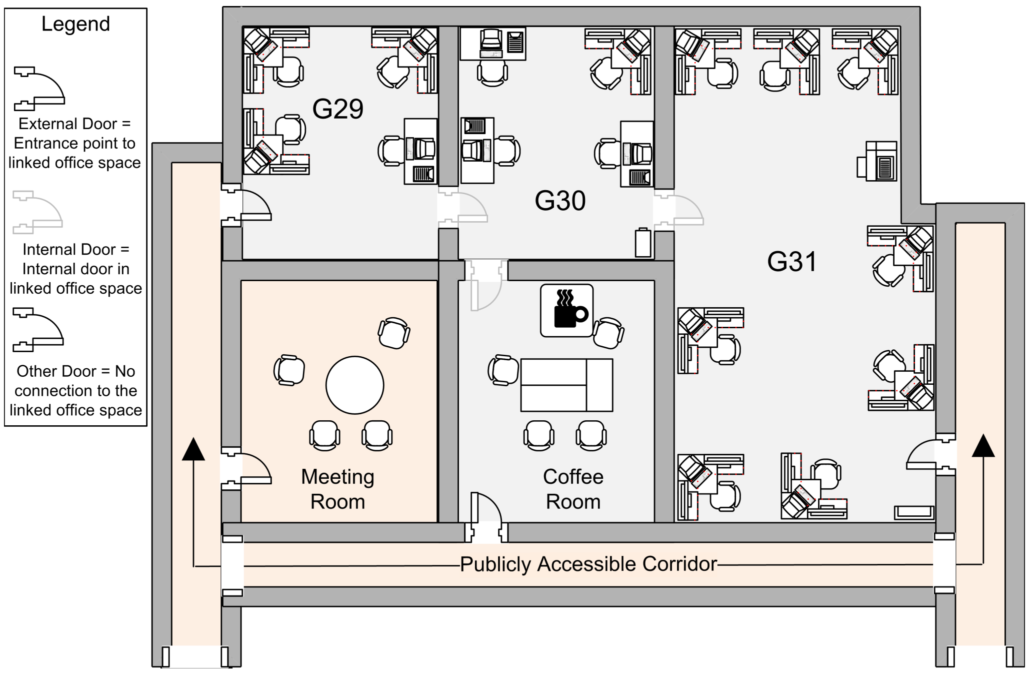

Legend
External Door = Entrance point to linked office space
Internal Door = Internal door in linked office space
Other Door = No connection to the linked office space
G29
G30
G31
Meeting Room
Coffee Room
Publicly Accessible Corridor

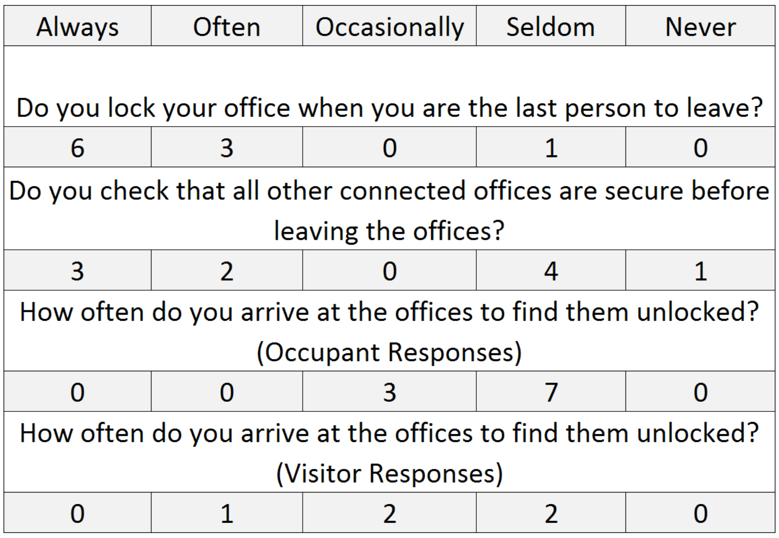
Figure 6.12: Layout of the Office Space for the Access Control System

Survey Monkey[4], to determine that the observed security vulnerabilities are an ongoing issue. Figure 6.13 summarises the key responses from the survey that confirm the following vulnerabilities:

Publicly accessible corridors: The corridor surrounding the offices essentially has public access during building opening hours, which are dependent on the time of year. This allows students and visitors to move freely about the building. Although a security desk is in operation at the entrance to the building, there is no sign-in process in place and visitor details for the building are not recorded. Outside of building opening hours, swipe access is available however this has not been sufficient to prevent intruders gaining access to the building, and which has directly affected the office space in this case study.

Complex combination of internal doors and exits: Internal doors within the space are generally left open to enable and encourage collaboration among occupants. This introduces the risk that at times offices are not secure because they are only partially occupied and partially secured. The results

[4]Survey Monkey http://www.surveymonkey.com/ (accessed 21st April 2011).



Always	Often	Occasionally	Seldom	Never
Do you lock your office when you are the last person to leave?				
6	3	0	1	0
Do you check that all other connected offices are secure before leaving the offices?				
3	2	0	4	1
How often do you arrive at the offices to find them unlocked? (Occupant Responses)				
0	0	3	7	0
How often do you arrive at the offices to find them unlocked? (Visitor Responses)				
0	1	2	2	0

Figure 6.13: Responses from occupants and visitors when surveyed about security risks in the office space for this case study.

in Figure 6.13 show that occupants leave the space unlocked at times, and both occupants and visitors reported finding the offices empty and unlocked, which indicates that this is a real vulnerability of the space.

Line of Sight Obstruction: Although all offices in the space are connected, the walls which separate the offices also mean that users cannot see the full space and so may not be aware of who is or is not present in other offices.

This case study proposes an automatic access control system for the office space, which is tested and assessed using the InSitu Toolset. The case study features two types of users, occupants and visitors. Occupants are defined as key-holders of the office space i.e. those with access privileges to unlock the doors to the space. This group includes postgraduate students and post-doctoral researchers who have desks located in the space. These occupants are all members of the same research group. Visitors are defined as non-keyholders and do not have access privileges to unlock the doors to the space.

6.3.2 Modelling the Access Control Case Study

This section discusses how the office-space and test specification are modelled using the Spatial Model, Activity Model and Situation Model from the InSitu Model Framework. To accommodate the reuse of the Activity Model from the first case study, the physical deployment space also makes use of a common room in the same building as the offices, but not co-located with them. The simulated building model was developed specifically for this case study, including positioning of sensors and configuration of the Activity Model. For brevity, the model encodings are not included in this chapter but they can be found in Appendix B.

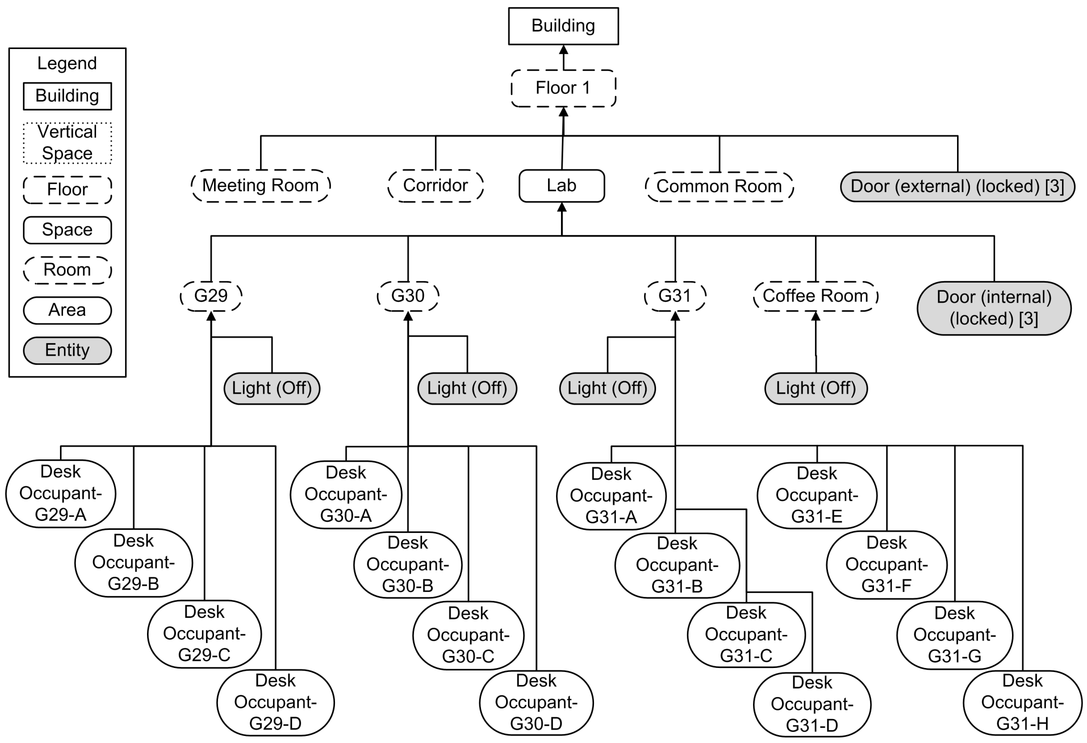


Figure 6.14: Hierarchy of Zones and Entities.

Spatial Model

The hierarchy of zones, for the case study, is illustrated in Figure 6.14, including the containment of doors and lighting in the offices. Entity nodes are displayed in a condensed view due to the quantity of each that needs to be depicted. The access control case study also features a space zone, called 'Lab', which models the operational use of the connected offices as an open connected lab space for postgraduate students.

Three door types are defined for the case study, including *external-doors*, *internal-doors* and *doors*. Doors that are not part of the connected office space are simply described by the *door* type e.g. the meeting room door. External doors are defined as any door that connects the corridor to the lab space e.g. the door between the Corridor and G29. Internal doors are part of the lab space, but are not exit points from the space e.g. the door between the Coffee Room and G30. In the XML model of the space, the initial-state of external-doors are denoted set to locked, and internal-doors are set to unlocked.

Situation Model

This section describes a set of situation definitions which are relevant for testing the access control systems. For brevity, only the logical definitions are included in this section. The Drools encodings of these situations are included in Appendix B.

Situation B.1: *Visitor present without an occupant.*
if
U(type = "visitor") $\longleftarrow$ v
$\wedge$ L((zoneID = v.roomLocation) $\wedge$ ((type = "office") $\vee$ (type = "coffee-room")))
$\wedge$ $\nexists$ (U(((type = "staff") $\vee$ (type = "student")) $\wedge$ (roomLocation = v.roomLocation))
then
Output $O(t) \subset G(t)$

Situation specification B.1 checks for the presence of a visitor in the office-space without the supervision of an occupant. It first checks for users of type 'visitor'. The logic then tests if the visitor is in an office or the coffee room, i.e. inside the connected offices. Finally, if a visitor is present in any part of the connected office-space, the logic checks if a staff member or student is present with the visitor. If no occupant is present, the rule fires and outputs some defined subset of the global state space as an alert to the Alert Report.

<u>**Situation B.2:**</u> *Visitor unseen by occupants in the same room.*

if

U (type = "visitor") $\longleftarrow$ v

$\wedge$ L ((zoneID = v.roomLocation) $\wedge$ (type = "office" $\vee$ type = "coffee-room"))

$\wedge$ $\exists$ (U ((type = "staff" $\vee$ type = "student") $\wedge$ (roomLocation = v.roomLocation)))

$\wedge$ $\forall$ (U ((type = "staff" $\vee$ type = "student") $\wedge$ (roomLocation = v.roomLocation)):

U (v $\in$ FOV))

then

Output $O(t) \subset G(t)$

Situation specification B.2 checks for the presence of a visitor in the offices, when occupants are also present but the visitor is outside of all occupants' line of sight. The logic first looks for the presence of a visitor in an office or in the coffee room. The rule then checks for occupants in the same room as the visitor. Finally, the rule checks the field-of-view of all occupants in the same room as the visitor to determine if any have the visitor in their line of sight. If no occupant has the visitor in their line of sight, the rule fires and outputs some defined subset of the global state space as an alert to the Alert Report.

<u>**Situation B.3:**</u> *Office space unlocked and no occupant present.*

if

L ((zoneType = "space") $\wedge$ (type="office-space")) $\longleftarrow$ l

$\wedge$ $\nexists$ (U ((type $\neq$ "visitor") $\wedge$ (spaceLocation = l.zoneID)))

$\wedge$ $\exists$ (E ((type = "external-door") $\wedge$ (state $\neq$ "locked")))

then

Output $O(t) \subset G(t)$

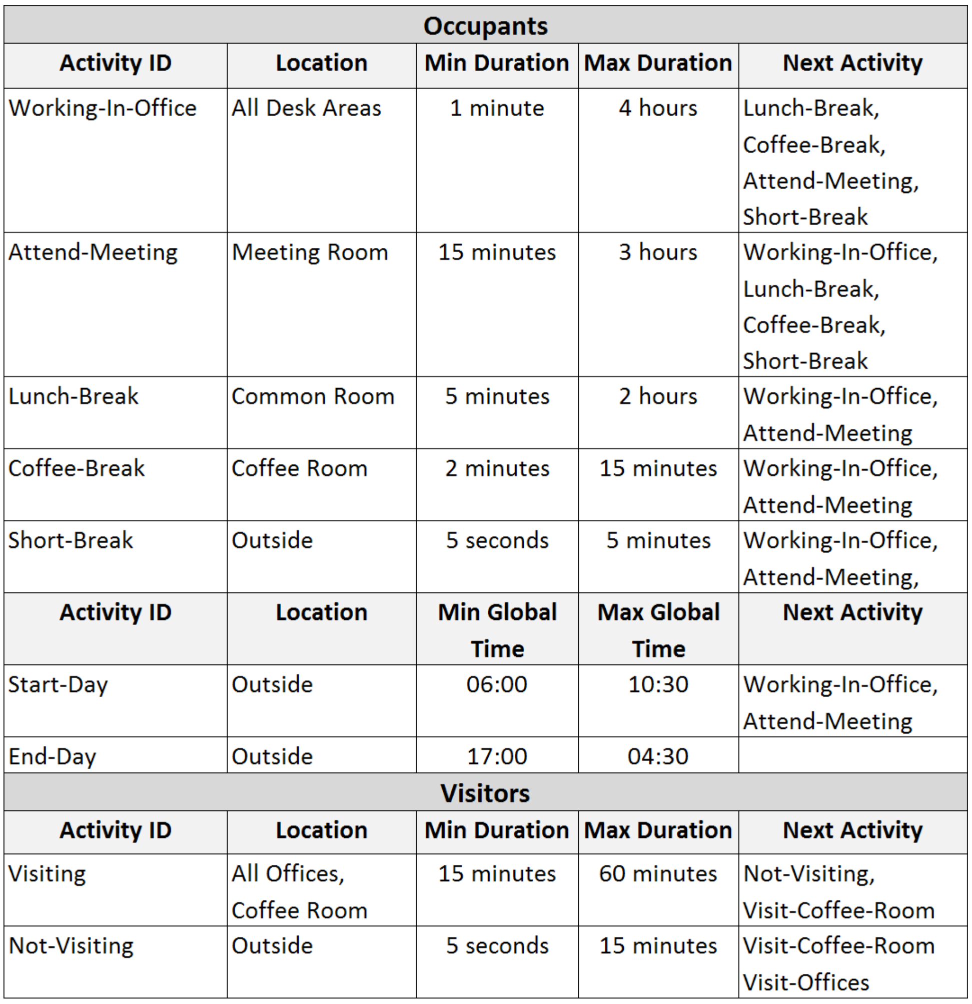
Situation specification B.3 checks for the absence of all occupants across the whole office-space, while the space is unlocked and therefore unsecured. This situation differs from the previous two in that it must consider the connected offices collectively, rather than each individually. To test this, the logic first checks for locations of zone type office-space. The logic then checks if any door, that is an access point to the space, is unlocked. Finally, the logic checks that no user exists in the space who is not a visitor, i.e for the purpose of this case study the user is a staff member or student. If any door is unlocked and no occupants are present, the rule fires and outputs an alert.

Situation B.4: *User waiting to enter space.*
if
U (type = "visitor") $\wedge$ (areaLocation *matches* "wait.*")
then
Output $O(t) \subset G(t)$

Situation specification B.4 makes use of the finest granularity spatial division offered by the Spatial Model, i.e. area. This rules checks for times that a visitor is waiting outside an office attempting to enter. The intention with this rule is to test the level to which a locked office is prohibitive to non-malicious visitors entering the office space. This situation checks for the presence of a visitor in an *area* demarcated as a waiting area outside the offices. When the situation exists, the rule fires and outputs an alert to the Alert Report.

Activity Model

The Activity Model for the access control case study, as shown in Figure 6.15, builds on the Activity Model from the first case study, by reusing some of the activity definitions including working-in-office, lunch-break, coffee-break and attend-meeting. It was considered valid to reuse this model because although the case study focuses on a different office-space, due to relocation, many of the occupants of the space overlapped with the first case study. Although the extent of overlap is difficult to ascertain due to anonymity of the survey responses, the occupants of both spaces are members of the same research group and at a minimum the workplace culture is consistent across both groups.



Occupants				
Activity ID	**Location**	**Min Duration**	**Max Duration**	**Next Activity**
Working-In-Office	All Desk Areas	1 minute	4 hours	Lunch-Break, Coffee-Break, Attend-Meeting, Short-Break
Attend-Meeting	Meeting Room	15 minutes	3 hours	Working-In-Office, Lunch-Break, Coffee-Break, Short-Break
Lunch-Break	Common Room	5 minutes	2 hours	Working-In-Office, Attend-Meeting
Coffee-Break	Coffee Room	2 minutes	15 minutes	Working-In-Office, Attend-Meeting
Short-Break	Outside	5 seconds	5 minutes	Working-In-Office, Attend-Meeting,
Activity ID	**Location**	**Min Global Time**	**Max Global Time**	**Next Activity**
Start-Day	Outside	06:00	10:30	Working-In-Office, Attend-Meeting
End-Day	Outside	17:00	04:30	
Visitors				
Activity ID	**Location**	**Min Duration**	**Max Duration**	**Next Activity**
Visiting	All Offices, Coffee Room	15 minutes	60 minutes	Not-Visiting, Visit-Coffee-Room
Not-Visiting	Outside	5 seconds	15 minutes	Visit-Coffee-Room Visit-Offices

Figure 6.15: Activity definitions for Access Control Case Study.

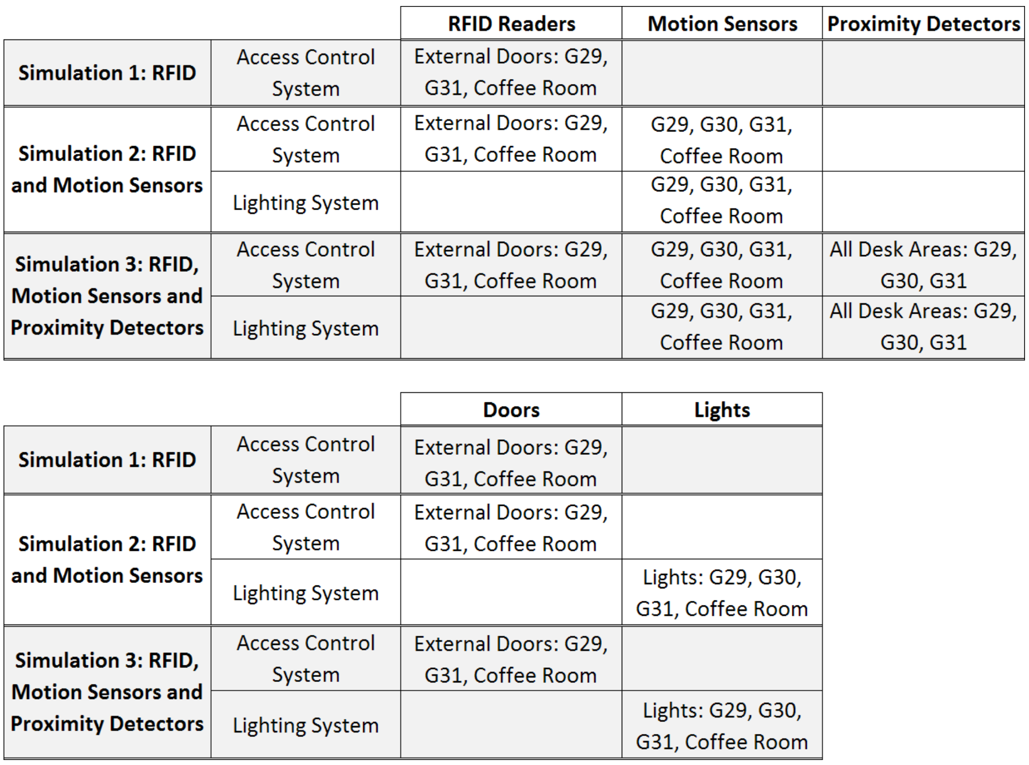
Additions to the Activity Model include activities for start-day, end-day and short-break. In a similar manner to case study 1, occupant survey responses were used to inform the bounds on the start-day and end-day activities. For brevity the charted results are included in Appendix B. The only difference for these activities is that they report duration in terms of time of day, rather than relative duration of an activity. The start-day activity is bounded between 06:00 and 10:30, while the end-day activity is bounded between 17:00 and 04:30. The latter of these bounds, 04:30, is assumed to be an extreme situation, however four respondents returned times distributed between midnight and 04:30. This time was retained because it was not an isolated response.

It was considered intrusive to survey occupants about WC breaks, a privacy issue which Tabak and de Vries [110] also acknowledged and respected in their study. For this reason a general short-break activity was defined. This activity is used to represent short breaks such as WC breaks, cigarette breaks or taking a call on a mobile phone. The minimum and maximum duration of the short break was defined as between 5 seconds and 5 minutes.

The addition of a second user type, *visitor*, requires additional activities, which include visiting and not-visiting. Instances of the visiting activity were placed in G29, G30, G31 and the Coffee Room. The not-visiting activity was positioned outside the office-space, close to the entrance to the building to simulate a visitor bot entering and leaving the building. The duration of the non-visiting activity was contrived with the aim of generating high frequency access attempts to the office-space because the objective of the case study is to demonstrate the capabilities of InSitu specifically in terms of the situation-based testing approach. Similarly, the duration of the visiting activity was of longer duration with the aim of generating situations in which the visitor is not unsupervised in the office-space.

6.3.3 Simulation Configurations for Access Control Case Study

The simulations conducted for this case study each feature a different combination of access control decision logic and environment configuration. In each configuration access control focuses on the external doors of the office space that can exist in any of three states: *open*, *closed* or *locked*. To demonstrate parallel ubicomp system testing the lighting system is run alongside the access control system in two of the simulations. This also demonstrates the reusability and generality of situation specifications through the reuse of the situation specifications from the lighting case study. All of the simulations use the same Activity Model configuration and featured 16 occupant bots and one visitor bot.



		RFID Readers	Motion Sensors	Proximity Detectors
Simulation 1: RFID	Access Control System	External Doors: G29, G31, Coffee Room		
Simulation 2: RFID and Motion Sensors	Access Control System	External Doors: G29, G31, Coffee Room	G29, G30, G31, Coffee Room	
	Lighting System		G29, G30, G31, Coffee Room	
Simulation 3: RFID, Motion Sensors and Proximity Detectors	Access Control System	External Doors: G29, G31, Coffee Room	G29, G30, G31, Coffee Room	All Desk Areas: G29, G30, G31
	Lighting System		G29, G30, G31, Coffee Room	All Desk Areas: G29, G30, G31

		Doors	Lights
Simulation 1: RFID	Access Control System	External Doors: G29, G31, Coffee Room	
Simulation 2: RFID and Motion Sensors	Access Control System	External Doors: G29, G31, Coffee Room	
	Lighting System		Lights: G29, G30, G31, Coffee Room
Simulation 3: RFID, Motion Sensors and Proximity Detectors	Access Control System	External Doors: G29, G31, Coffee Room	
	Lighting System		Lights: G29, G30, G31, Coffee Room

Figure 6.16: Simulation configurations for the Access Control Case Study.

The following describes the prototype systems tested, which are also summarised in Figure 6.16:

RFID Access Control System This simulation tests an RFID-based access control system in which occupants of the space scan a personal RFID card at the readers located outside the external doors of the office space. Doors automatically lock again when a door closes.

RFID and Motion Sensor Access Control System This simulation tests an RFID and motion sensor based access control system. RFID access control operates as already described. Motion sensors deployed within the office space keep the external doors in an unlocked state while activity is detected in the environment. A timer controls locking the doors three minutes after the most recent motion sensor activation. The aim is to make the offices more accessible to visitors.

In this simulation the lighting system is also tested using inputs only from the motion sensors. The timer for the lighting system expires after 10 minutes. The lighting case study situation specifications from case study 1 are reused without making any changes.

RFID, Motion Sensor and Proximity Detection Access Control System

This simulation tests an RFID, motion-sensor and proximity-detection based access control system. RFID and motion sensor access control operates as already described. Proximity detectors are installed at each of the desks to sense when occupants are present. Proximity detection is introduced to further improve accessibility for visitors and also to improve the automatic lighting, by keeping doors unlocked and lights on while occupants are present in an office. This third simulation configuration is relevant for improving both the access control system and the lighting system, through the introduction of proximity sensors.

6.3.4 Simulation Results for Access Control Case Study

The main focus of this second case study is on examining the modelling capabilities and expressiveness of the InSitu testing approach, for this reason detailed charts from these simulations are included in Appendix I. Figures 6.17, 6.18 and 6.19, provide summaries of the situations detected during each of three simulations.

Situation B.4 (*User waiting to enter space*) was detected in each simulation. This situation arises any time the visitor is outside the offices waiting for the door to open. In the simulation which uses RFID control only, Figure 6.17, the duration of these situations is prolonged. This is because often the visitor does not gain access because a swipe card is necessary. In contrast, the simulation which uses RFID, motion detection and proximity detection, Figure 6.19, exhibits this situation in much shorter duration. This is because the doors were unlocked in the office space for more time during this simulation, increasing accessibility to the office space for the visitor bot.

Situation B.3 (*Office space unlocked and no occupant present*) is detected in all three simulations. However, the situation that is detected is not a problematic one and so can be disregarded. The situation can be detected by the Monitoring Engine in the time it takes for the door to open and the bot to enter the space. Figures 6.17, 6.18 and 6.19 illustrate the short duration of this situation detection, which can be seen early in each of the simulations.

Situation B.2 (*Visitor unseen by occupants*) and situation B.1 (*Visitor present without an occupant*) are exclusive situations for location. In any given location, only situation B.2 or situation B.1 can occur at any one time. This is evident in each of the charts, Figures 6.17, 6.18 and 6.19, because there is only one visitor bot in the simulations.

Situation B.2 (*Visitor unseen by occupants*) could have been defined as a subspace of situation B.1 (*Visitor present without an occupant*), because the visitor will always be unseen by occupants when no occupants are present, therefore when situation B.1 occurred, situation B.2 (*Visitor unseen by occupants*) also occurred. However, a more specific specification for situation B.2 was defined, which searched for the situation *Visitor unseen and at least one occupant present.* This provides more informative results to the tester.

Situations A.1 (*User in the dark*) and A.2 (*Lights on unnecessarily*) were included in the second and third simulations. The results highlight the impact of the proximity sensors in reducing situation A.1 (*User in the dark*) in the final simulation, Figure 6.19. Proximity sensors do not eliminate the problem entirely because they were not deployed in the coffee room, and they do not respond to the visitor bot. The increase in occurrences of situation A.2 (*Lights on unnecessarily*) is noticeable although not significant and could be reduced by shortening the timer cycle on the lights, particularly because the proximity sensors alleviate the problem of users in the dark extensively.

Although not included in this chapter, for reasons of brevity, Appendix I provides examples of information overlays in which the information from an alert report is combined with test logs in charted views to improve integration and reconciliation of the Alert Report and simulation log data.

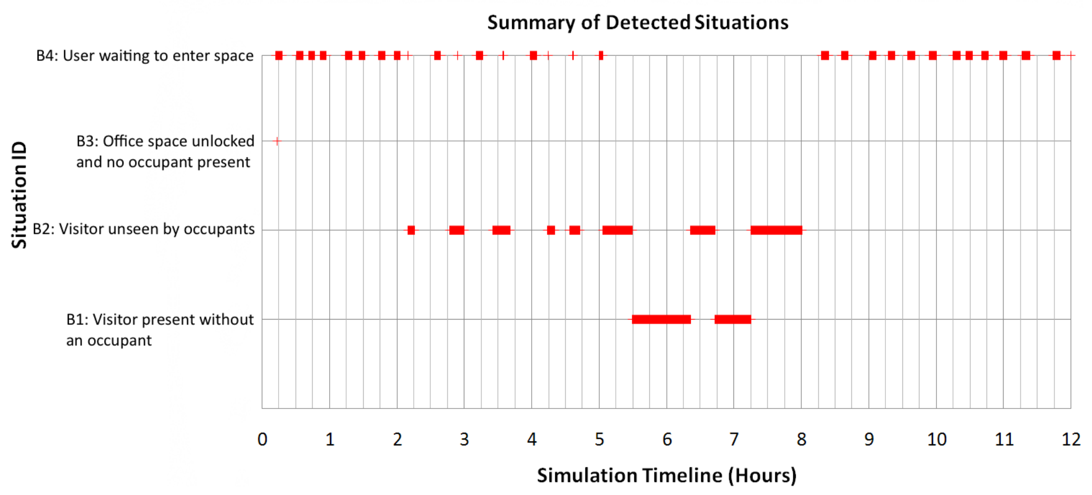


Figure 6.17: Simulation 1 (RFID): Summary of detected situation instances.

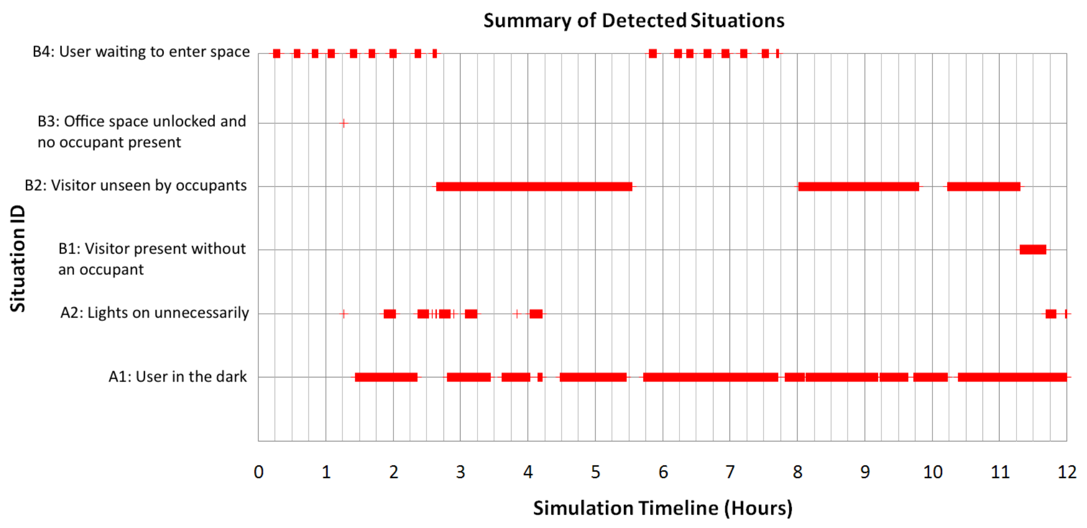


Figure 6.18: Simulation 2 (RFID and Motion Sensors): Summary of detected situation instances.

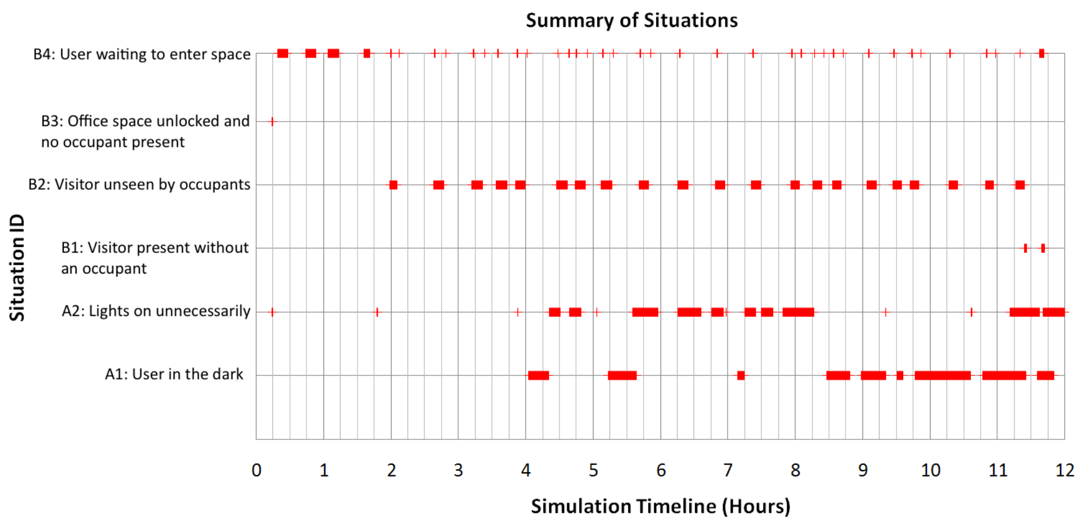


Figure 6.19: Simulation 3 (RFID, Motion Sensors and Proximity Detection): Summary of detected situation instances.

6.3.5 Case Study 2: Findings

This access control case study builds on the findings from the lighting case study. Where the first case study focused on demonstrating that InSitu can correctly identify examples of real-world inappropriate situations, case study 2 focuses on providing a broader demonstration of the extent and expressiveness of the InSitu Model Framework, including the Spatial Model and Situation Model.

The major finding from this case study is that generality of situation specifications provides a major benefit because the tester does not need to foresee every situational instance that might arise. The tester can define the general or abstract situation and the Monitoring Engine can apply it globally in the test space, to individuals, groups or aggregations of elements of the deployment space. This provides a novel approach to testing ubicomp systems compared to previous examples, which use overly specific situation specifications or which do not support run-time monitoring. This is reinforced through the reuse of the Situation Model from the lighting case study, which featured during two of the simulations in this case study. The separation of concerns between the Spatial Model, State Model and Situation Model, combined with the generality of the situation specifications enabled reuse of the lighting system situations.

The multi-user nature of this case study, and the introduction of the *visitor* user type, required that situation specifications test both individual users as well as collective groups of users. Examples of this are evident in situations B.1 and B.2, which test an individual visitor's location against the location of each member in the group of occupants. Similarly, the operational use of the office-space required that the aggregate state of the space be examined, not limited to individual rooms. This is particularly evident in situation B.3, which analyses the state of the connected office space. The Spatial Model is the enabling factor for these situations.

A secondary finding from this case study is the importance of the relationship between situation specifications, in particular the subsumption, overlap and exclusivity relationships. The situation of an occupant being unobserved is related to both situations B.1 (*Visitor present without an occupant*) and B.2 (*Visitor unseen by occupants*). There is an overlap between the situation of an occupant being present and a visitor being unobserved, subject to the orientation of the occupant. However, an occupant will always be unobserved when no occupants are present, under the assumption that the doors to the space are closed, as is the predominant state during these simulations.

During initial work on this case study, situation B.2 (*Visitor unseen by occupants*) was defined to test only that a visitor was unobserved. This manifested the overlap with situation B.1 (*Visitor present without an occupant*). This was subsequently refined to result in an exclusive relationship between situations B.1 (*Visitor present without an occupant*) and B.2 (*Visitor unseen by occupants*), by specifying the latter to test for times that a visitor is unobserved and at least one occupant is present.

This case study also contributes towards assessing traceability in the results generated in the Alert Report. The charts for this case study, presented in Appendix B and Appendix I, demonstrate the Toolset's capability to provide traceable results surrounding the *who*, *where* and *when* of a situation. The Toolset also records *what* situation was experienced (and detected), enabling the tester to trace the spatio-temporal events that were causal to the situation arising. This can assist the tester to uncover *why* a particular situation arose.

Case Study 2: Summary

In summary, the case studies have presented an evaluation of the correctness and expressiveness of the Model Framework. The next section of this chapter presents an assessment of the ability of technical users to work with the Model Framework and the situation-based testing approach for ubicomp systems.

6.4 User Trial

The purpose of the user trial is two-fold. Firstly, to objectively assess the expressiveness and suitability of the InSitu Model Framework and InSitu Test Process for testing situations and evaluating the behaviour exhibited by ubicomp systems. Due to time constraints per trial session, the focus in the trial was placed on the expressiveness of the Situation Model and State Model. These models were prioritised because together they form the framework for defining the test situation specifications that are core to the situation-based testing approach. A second aim of the user trial is to determine the level to which computer scientists can interpret and infer meaning from a charted Alert Report.

6.4.1 User Trial Description and Methodology

User Trial Outline

The user trial featured a collection of tasks which involved definition of situation specifications, configuration of a simulation, and interpretation of a charted Alert Report. During the trial participants were observed and timed as they completed these tasks. The simulation configuration produced during each trial session was run for 12 hours and a resulting Alert Report was generated and charted. Due to this delay while the simulation was completed, the charted simulation results were emailed to participants. Email distribution of charted results was chosen because it was considered important to capture feedback while the trial session was as fresh as possible in participants' memory, rather than to reconvene another session.

The lighting system from the first case study was chosen as the system under test for this user trial. Using the lighting system provided two main benefits to this user trial. Firstly, the lighting system was known to exhibit problematic behaviour which provided a specific comparison point across trial sessions. Secondly, the simplicity of the lighting system meant that one full iteration of a test simulation cycle could be completed within the time-frame for each trial session. Participants were not asked to build the lighting system i.e. to write code or configure sensor positions, because an assumption of working with InSitu is that the designer has an implemented ubicomp prototype ready for testing. Additionally, the focus of the user trial was specifically to capture an objective assessment of the situation-based testing approach, which is primarily concerned with the testing and evaluation phase of a prototyping cycle, rather than the design phase.

Trial Session Procedure

Each trial session was comprised of six tasks. A trial booklet which included instructions for each task as well as instructions for the set of InSitu tools, including Eclipse, Drools and Hammer, was provided to each participant. The instruction booklet is included in Appendix D. To assist with timing each task, the booklet was given to participants in individual sections, each covering one task at a time. Prior to each task, a presentation or tutorial was delivered to demonstrate the relevant features of InSitu. Presentation slides and printouts were used to help maintain consistency in the tutorials across trial sessions.

Task 1 asked participants to use pseudocode to generate situation specifications. The role of pseudocode was to ensure that situation specifications could be captured for each team, regardless of participants' ability to code Drools rules. It also resulted in situation specifications, which were created independent of the InSitu Model Framework, and which could be used as reference points to assess the expressiveness of the State Model and Situation Model. Pseudocode provided participants with the freedom to specify situations without being bounded by the vocabulary, and logical and spatial relations of the Model Framework. As a result it was possible to capture the intent of participants, in terms of the situation specifications that they aimed to write, in order to test the lighting system.

Task 2 asked participants to convert the pseudocode to a Drools encoded Situation Model. There is a strong link between this task and the pseudocode providing a natural flow between these two tasks. This task was required in order to produce testable situation specifications for use in the simulation for each trial session.

Task 3 involved completing the Situation Model by creating the output schema for the Alert Report using prescribed Java method calls to build the XML node structure for individual alerts. In this task users were asked to select the appropriate Environment State Model attributes that would be helpful when trying to trace causal factors leading to inappropriate situations. This step used the Java XML model manipulation functions from the Adaptive Engine.

Task 4 involved creating an Activity Model for the simulations. Participants used a Microsoft Excel spreadsheet to define activities in terms of activity name, location, duration and next-steps. Due to the constraints of the experimental setting, it was considered unreasonable that participants would be able to make informed decisions about the duration of activities. For this reason, participants were provided with the list of survey durations collected for the lighting case study.

Task 5 asked participants to transfer the Activity Model from the spreadsheet to Hammer. This is a two step process firstly involving creation of each individual activity and subsequently creation of the interconnecting relationships between activities.

Task 6 asked participants to review the charted results from the Alert Report produced by their simulation configuration and situation specifications.

Questionnaires

In order to capture participants opinions and feedback three questionnaires were completed by each participant during each trial session. The first was a pre-experiment questionnaire to capture participant profiles. The second questionnaire was issued after the simulation configuration process was completed, i.e. after Task 5. This questionnaire was completed under supervision and asked the participant about their experiences defining situation specifications and configuring a simulation. The final questionnaire was issued and returned by email after the simulation had been run. This questionnaire focused on participants' ability to interpret charted Alert Report results.

Trial Dry Run

Two dry-run trial sessions were conducted before commencing the trials to check the trial process for problems that would interfere with the findings from this experiment. This uncovered three specific issues.

Firstly, during the dry-run trials it was found that Drools was a major obstacle for participants. Due to a shortage of volunteers with knowledge of both Drools and ubiquitous computing, it was decided that all trial sessions needed to be conducted in teams of two. The aggregate knowledge of each team could then provide at least basic knowledge of Drools and relevant aspects of ubiquitous computing, i.e. context-aware computing, context modelling, intelligent buildings or embedded sensor technology. Participants were randomly assigned to teams based on an indication of their knowledge contribution available to a team.

The remaining technologies used in the dry run trial sessions include Hammer, Microsoft Excel and Java method calls. The role of these technologies, for situation-based testing, was discussed in the Chapter 5, in relation to the Toolset implementation. These technologies were found to be general enough that additional expertise was not required by participants in the dry-run trial sessions.

- Hammer Editor involves using drag-and-drop, copy and paste, data entry to dialog boxes, as well as standard navigation controls for PCs.
- The Microsoft Excel spreadsheet interaction is limited to data entry.
- Java programming is limited to method calls. Java is a commonly taught language to undergraduate computer science students.

The second issue uncovered during the dry run trial sessions was that some participants initially defined absolute situation specifications. For example, one team started out by writing tests for every room in the building instead of testing using relative location. To overcome this issue, an example situation specification was added to the trial tutorial. The example illustrated how to define the situation specification of proximity between any occupant and any door.

The third issue uncovered during the dry run trial sessions was that participants instinctively tried to correct inappropriate situations rather than log them for further analysis. In this case, participants either forgot or misunderstood that InSitu is not designed to run post-deployment, i.e. it is a pre-deployment design tool. To address this, an instruction explicitly reminding participants to log rather than fix situations was added to the tutorial.

6.4.2 Profile of Participants

Ten people volunteered to take part in this experiment which provided enough participants for five teams in five trial sessions. This section details the responses and findings collected in the pre-experiment questionnaire.

Participant Prerequisite Requirements

The general requirement for all participants was that they had no prior experience of working with InSitu and no prior in depth knowledge of the InSitu approach. Participants were required to have a formal education in computer science, i.e. by holding a degree in computer science or a more relevant field. The choice of the lighting case study as the test system meant that participants were working with a system familiar in everyday terms so that it required only basic understanding of context aware systems. This provided greater scope of eligibility for volunteers. Participants were not expected to have experience of 3D virtual reality or first-person shooter games since these are not core skills for a context-aware system designer.

In summary, participant prerequisite requirements were:

(i) Participants should be of a technical background i.e. they should have a degree or other qualification in the field of computing.

(ii) Participants should have knowledge of Drools and/or a relevant aspect of ubiquitous computing, e.g. context-aware computing, context modelling, intelligent buildings or embedded sensor technology.

(iii) Participants should not have prior experience of working with the InSitu tool and should not have prior in depth knowledge of the InSitu approach.

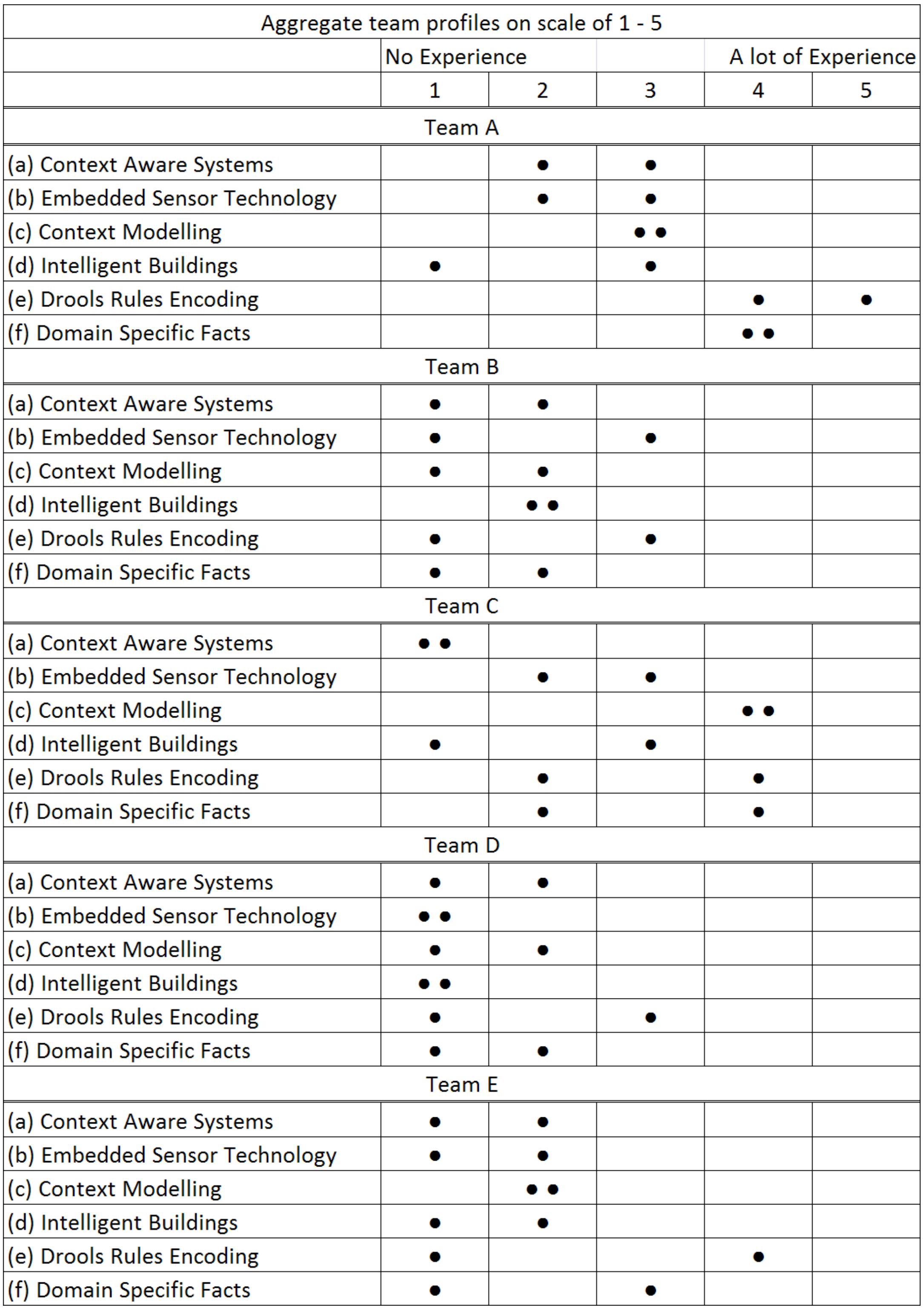
Technical Experience

In the pre-experiment questionnaire, all participants indicated that they had a technical background with a formal education in computer science or closely related field. The breakdown of relevant degrees across the group of participants included nine in computer science, one in computer engineering, one in ubiquitous computing, two in multimedia and one in mathematics. Only one participant did not report holding a degree in general computer science but instead held a more relevant degree in ubiquitous computing. This group of participants appropriately reflects the technical user base that would be expected to work with InSitu.

Aggregate Team Experience

This section demonstrates that each team held at least basic experience in some aspects of both ubicomp and Drools. Teams are reported in no particular order to preserve anonymity, i.e. they are not presented in the order which the trial sessions were conducted. Team profiles are illustrated in Figure 6.20.

Team A was very experienced with Drools, with both participants reporting at least a level 4, *significant*, experience for Drools LHS encoding and Domain Specific Facts. Both participants also had experience of ubicomp systems with both reporting level 3, *good*, experience for context modelling and only one of the participants reporting *no experience* in one category, Intelligent Buildings.



Aggregate team profiles on scale of 1 - 5					
	No Experience			A lot of Experience	
	1	2	3	4	5
Team A					
(a) Context Aware Systems		●	●		
(b) Embedded Sensor Technology		●	●		
(c) Context Modelling			● ●		
(d) Intelligent Buildings	●		●		
(e) Drools Rules Encoding				●	●
(f) Domain Specific Facts				● ●	
Team B					
(a) Context Aware Systems	●	●			
(b) Embedded Sensor Technology	●		●		
(c) Context Modelling	●	●			
(d) Intelligent Buildings		● ●			
(e) Drools Rules Encoding	●		●		
(f) Domain Specific Facts	●	●			
Team C					
(a) Context Aware Systems	● ●				
(b) Embedded Sensor Technology		●	●		
(c) Context Modelling				● ●	
(d) Intelligent Buildings	●		●		
(e) Drools Rules Encoding		●		●	
(f) Domain Specific Facts		●		●	
Team D					
(a) Context Aware Systems	●	●			
(b) Embedded Sensor Technology	● ●				
(c) Context Modelling	●	●			
(d) Intelligent Buildings	● ●				
(e) Drools Rules Encoding	●		●		
(f) Domain Specific Facts	●	●			
Team E					
(a) Context Aware Systems	●	●			
(b) Embedded Sensor Technology	●	●			
(c) Context Modelling		● ●			
(d) Intelligent Buildings	●	●			
(e) Drools Rules Encoding	●			●	
(f) Domain Specific Facts	●		●		

Figure 6.20: Summary of aggregated team profiles.

Team B had level 2, *basic*, experience with Domain Specific Facts and level 3, *moderate*, experience with Drools LHS encoding. Similar to Team A, both participants had *good* experience of ubicomp systems with only one of the participants reporting *no experience* for the category Intelligent Buildings.

Team C had *good* experience in the Drools categories with both participants reporting at least *basic* experience and one participant reporting level 4, *significant*, experience. This team was also reasonably strong in the ubicomp categories since both team members had at least some experience in two of the ubicomp categories.

Team D had the least experience of all the teams however it was still sufficient to take part in the experiment. Based on the aggregated results the team had *basic* experience in two of the ubicomp categories and at least *basic* experience in the Drools categories.

Team E was also a low experience team. One participant provided all of the Drools experience but both participants contributed some experience of the ubicomp categories. The aggregate experience of the team members provided this team with at least some experience in all six categories.

The team profiles show that although all teams had the prerequisite experience, team A was the strongest team and team D was the weakest team that took part in the trial sessions. There was sufficient ubicomp experience and Drools experience among the group so that each team's aggregate experience included at least basic knowledge of both fields.

6.4.3 Participant Performance Configuring InSitu

During each trial, teams were timed as they completed each of the first five tasks in the experiment. Figure 6.21 provides a chart of times recorded for each of these tasks. Tasks are marked on the $x - axis$ and the time recorded for each team is marked by the points plotted in the chart area. Participants did not have the

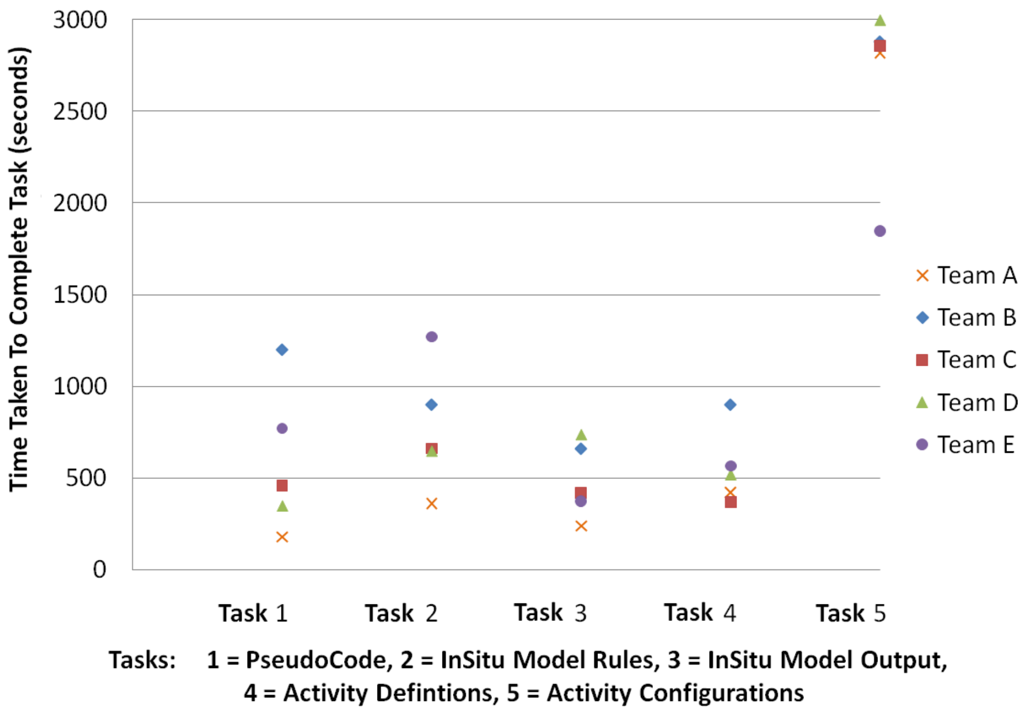


Figure 6.21: Participant performance configuring an InSitu simulation measured through timed tasks.

opportunity to check that their model worked before the end of the trial session due to time constraints. It was found that four of the five models had syntax errors. Corrections were made only to fix syntax errors but the logic was not changed for any of the models. Corrections included: fixing typos; replacing single equals sign = with double equals signs ==; correcting State Model attribute calls to use the defined getter methods.

Trial Session Observations

The most significant challenge that arose for every team was an instinct to correct inappropriate situations rather than just log the situation for analysis. Instinctively, all teams initially wrote pseudocode to test ***if*** *the lights were off when a user is present* ***then*** *switch lights on*. This was despite the explicit instruction not to do this, that was introduced as a result of the dry run trials. Some of the teams realised this mistake during Task 1, other teams only realised the issue when they started Task 3, to create the output schema for the Alert Report. However, all teams

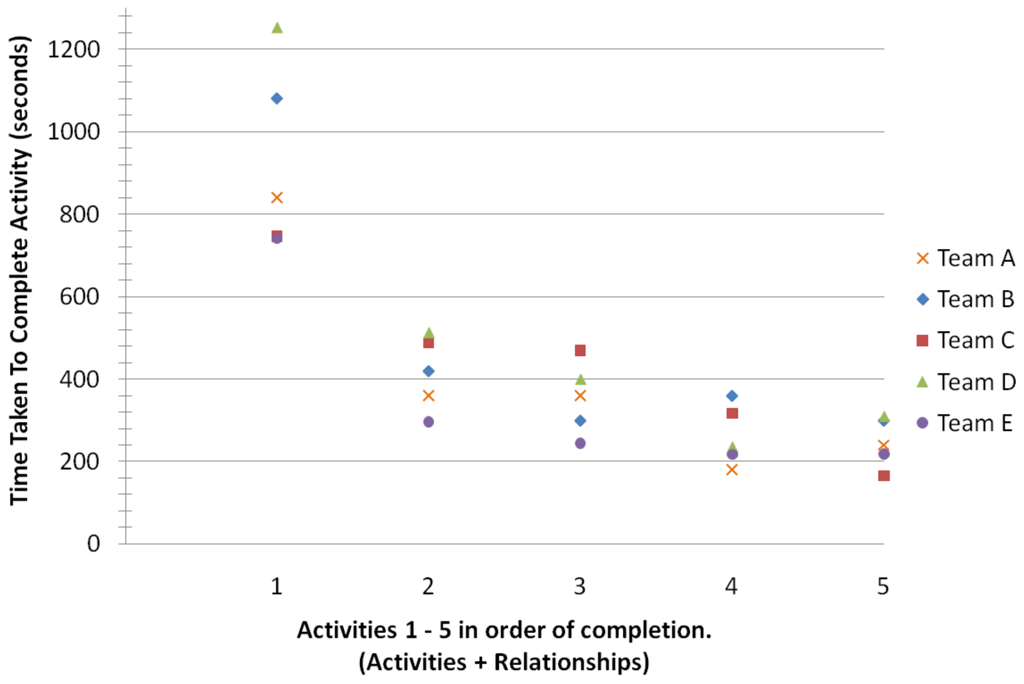


Figure 6.22: Participant Performance configuring the Activity Model in Hammer.

overcame this problem before progressing beyond Task 3. It is unclear how it might be possible to assist users to avoid this problem, however since Task 3 took less than eight minutes to complete, the learning curve is surmountable in a relatively short amount of time and is not particularly problematic.

In Task 2, one team found that they were limited by the expressiveness of InSitu. The team in question tried to test how long a room had been empty before raising an alert but had difficulty writing this rule using the attributes of the Environment State Model. This is because temporal modelling was not included for situation specifications because it requires management of context histories, which are a significant research challenge, both in terms of the quantity of information involved and the additional complexity of temporal analysis. However, suggestions about how this can be addressed are discussed as part of the future work for this thesis.

An interesting observation during Task 4, defining the Activity Model, was the reaction of teams to the surveyed results for the duration of activities. During the trial sessions, two of the five teams that took part in this experiment strongly disagreed with the duration of activities as defined by the survey results and were reluctant to use the values. In one case, a team wanted to cut the activity of working in an office from a maximum of 4 hours down to 1 hour. Using this in an Activity Model would not have uncovered the problematic behaviour in the lighting system leading to the inappropriate situation of the *user in the dark.* This shows that some participants were at risk of imposing their own assumptions about an environment onto the users who will occupy the space on a day-to-day basis. Although participants were not required to be designers for this trial, this result is relevant with respect to the current state of ubicomp in research and industry. Within the field of ubicomp, professional designers of context and sensor-driven systems are not yet widespread among the community. This role is often still filled by specialist architects, engineers, computer scientists and researchers. These professionals are not always informed or interested in human centric design and may place less emphasis on the needs of occupants, as opposed to the technical or physical challenges in the design.

In Task 5, although the process for configuring activities is lengthy, involving 58 steps (detailed in Appendix D), it was shown by timing teams that participants quickly overcame the learning curve associated with completing this process. It can easily be seen in Figure 6.22 that through repetition the overall performance of all teams improved within five activity configuration cycles. The average time to complete an individual activity configuration dropped from 15 minutes 32 seconds to 4 minutes 7 seconds. However, despite this improvement, the proportion of time which Task 5 consumed during each trial, compared to other tasks, highlights the need for an improved user interface or, as suggested by participants, automation of the process of configuring the Activity Model into Hammer, the map editor.

6.4.4 Participant Perception of Configuration Process

This section reports on the results gathered from the second questionnaire which posed questions about participants' experience of InSitu's underlying concepts and models, and the configuration process. Two scales were used to capture participant responses for this questionnaire. The first scale ranged from 1 - 5, where 1 = Yes and 5 = No. The second scale ranged from 1 - 5, where 1 = Easy and 5 = Difficult. Responses to this questionnaire are charted in Appendix C.

In response to *How easy was it to generate pseudocode to describe tests for your simulation?*, 7 out of 10 participants responded positively about InSitu indicating that they found this task easy. Although this is a simple question, it is an important result for the user trial. In writing pseudocode the participants were asked to work with InSitu without the constraints of the technical implementation and so this question assesses the participants' ability to relate to the underlying approach of testing situation specifications rather than defining use cases, procedural tests or scripting scenarios. Although some of the participants provided responses that were indifferent, none of the participants responded with an indication that the process was difficult. Corroborating this result, in response to the statement *I was able to complete the task of writing pseudocode*, all users responded with Strongly Agree.

In response to *How easy was it to convert the pseudocode to Drools Rules?*, 7 out of 10 participants responded positively that the process was easy. No participants indicated that they felt the process was difficult. This is a relevant result because it indicates that Drools was an appropriate choice of technology. This is corroborated by responses to the statement *I was able to complete the conversion of pseudocode to Drools Rules.* All users, except one, agreed strongly with this statement, the remaining participant responding neutrally.

Also important to this point is the expressiveness of Drools declarative logic and the defined Fact vocabulary for the Environment State Model. In response to the question *Did you find Drools LHS sufficiently expressive for your needs?*, 8 out of 10 participants returned a strong positive response. Although neither of the two remaining responses disagreed with the statement, it was expected that this question would not be completely positive because of the previous observation that one of the teams could not express their pseudocode completely due to the limitation on temporal modelling.

The next questions addressed the process of creating the schema for the Alert Report in the RHS of a Drools. In response to the question *How easy was it to encode the* *code to generate an alert (RHS)?*, 8 of the 10 participants responded with Easy. This is in-line with the expectations that were mentioned when setting out the prerequisite requirements for participants, that writing the Alert Schema for the Situation Model would not be a difficult task for computer scientists. Participants were also asked to respond to the question *Did you find Drools RHS sufficiently expressive?*. Although all participants responded positively, this was a premature result because it is difficult to judge the usefulness of an Alert Schema until after the simulation has run. This will be revisited in the next questionnaire.

In response to the question *Did you find the process for defining Activities using the Excel spreadsheet was systematic (i.e. there was a clear method to complete this task)?*, all participants agreed with this statement, with seven of those responses strongly positive. In response the statement *Did you find the process for transferring Activity definitions from the Excel spreadsheet to Hammer was systematic (i.e. there was a clear method to complete this task)?*, 8 of the 10 participants responded positively. Although, the configuration process is easy, it is also tedious and repetitive. Two participants highlighted this in their comments: *"Many details (in menus) to fiddle with"* and *"Having to change between different entities was tedious"*. These comments affirm the need for an improved interface to ease the burden on the tester during this part of the configuration process.

Perceived Limitations of the Configuration Process

In terms of improving the configuration process the main points identified include issues surrounding the automation of the configuration of activities. Responses included comments such as *"The excel [sic] gives a systematic way of defining possible sequences, if it could be embedded somehow in Hammer it would be nicer"* and *"Very time consuming activity, it seems like a lot of it could be automated. It takes a long time to enter very little information. Issue with using a general tool for specific job."*.

Also mentioned was improving the work flow during the configuration of a simulation, in particular in terms of tracking progress through the configuration tasks. Comments from participants included: *"Maybe some way to observe overview work done"* and *"If there is a lot of repetition it may be beneficial to tick off / overview tasks / work carried out and what is left to do."*.

These findings indicate that although the configuration process is systematic and users were able to follow the instruction booklet, InSitu could benefit from an improved interface. This was expected because a dedicated user interface has not yet been created for InSitu. It is an assuring and positive outcome that participants did not report limitations in the expressiveness of the underlying models or with the flexibility of the InSitu.

6.4.5 Participant Interpretation and Perception of Charted Simulation Results

This section reports on the responses gathered from the final questionnaire which focused on participants' ability to interpret charted Alert Report results. Participants were provided with the charted results from the Alert Report generated during their simulation. An example of the charted results provided to each team is included in Appendix J.

In response to *Do your results indicate that problematic behaviour exists in the prototype system?*, all ten participants responded *Yes.* This indicates that all of the participants were able to determine that inappropriate situations were detected during their simulation. To elaborate on and support their responses, participants were also asked to respond to the statement *Please specify the evidence from your simulation results to support your opinion.* In asking participants to elaborate the purpose was to determine that their interpretation of the results was correct. Six participants provided comments which confirmed that they were interpreting the results correctly. Three participants provided responses which were inconclusive or blank. One participant responded with a partially incorrect comment: *Situations with unnecessary lights on were logged. But situations of user in dark is not logged. I am not sure if this situation never happened during simulation or the was not*

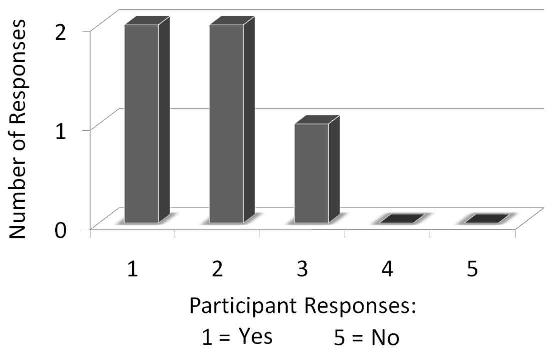


Figure 6.23: Responses from Ubicomp Team Members to the question: *Did the simulation results yield any unexpected findings for you?*

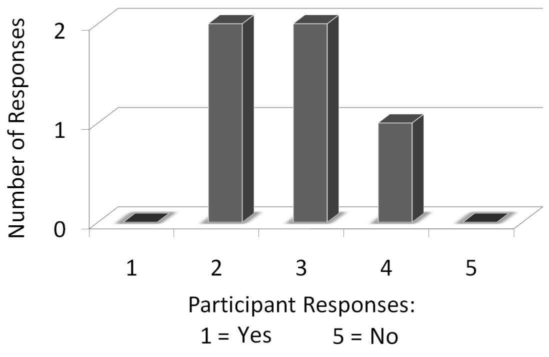


Figure 6.24: Responses from Drools Team Members to the question: *Did the simulation results yield any unexpected findings for you?*

working. This could be attributed to the fact that the user in the dark situation is visually less obvious in the charted results. Comments are included in Section C.4 of Appendix C.

To gain insight into the usefulness of the charted results, participants were ask to respond on a scale of 1 - 5, where 1 = Yes and 5 = No, to the statement, *Did the simulation results yield any unexpected findings for you?*. Figures 6.23 and 6.24 chart the responses to this illustrating that in general the ubicomp participants tended to respond more positively than Drools participants. Overall, more participants agreed with this statement than not which reinforces the observation that even simple sensor-driven systems can cause unexpected effects.

In response to the question *Do you think you could have uncovered problematic behaviour in the automatic lighting system more quickly without using this tool?*, two participants answered that they thought they could have uncovered the problem more quickly without the tool, which was a disappointing result in the survey. However, on closer inspection, all of the ubicomp participants were strongly positive in favour of InSitu's usefulness. The responses from Drools participants were less conclusive, as illustrated in Figures 6.25 and 6.26. This may indicate that Drools participants generally had less appreciation of the challenges faced by ubicomp designers and developers.

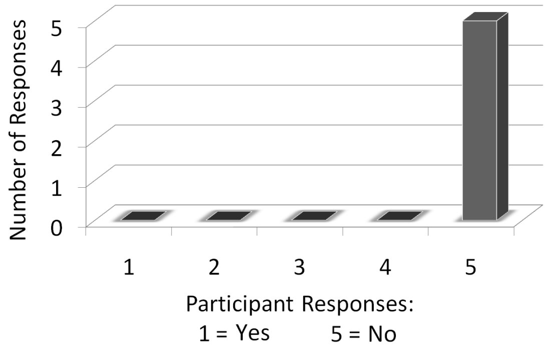


Figure 6.25: Responses from Ubicomp Team Members to the question: *Do you think you could have uncovered problematic behaviour in the automatic lighting system more quickly without using this tool?*

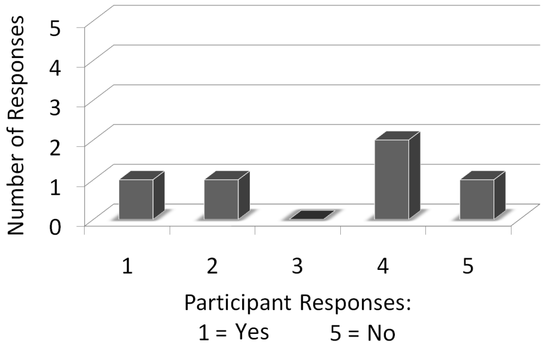


Figure 6.26: Responses from Drools Team Members to the question: *Do you think you could have uncovered problematic behaviour in the automatic lighting system more quickly without using this tool?*

For additional confirmation that participants were able to understand the results, participants were asked to make suggestions about how they would attempt to improve the lighting system. Half of the participants made suggestions that would correctly improve at least one aspect of the design of the lighting system, for example by adding additional motion sensors. The incorrect responses were deemed so because they fell back into the trap of using InSitu to correct inappropriate situations, rather than refining the ubicomp system design. These comments are included in Section C.4 of Appendix C.

6.4.6 Analysis of User Trial

The user trial determined that participants could relate to the approach and concepts used by InSitu to configure simulations and define situation specifications. This finding was primarily investigated through Tasks 1-3. The pseudocode captured during the trials showed that participants were able to define situation specifications that were non-specific in terms of particular locations, users or entities. The transition from pseudocode to the Situation Model and State Model demonstrated two points. Firstly, that these models were sufficiently expressive for participants of this trial to encode their respective pseudocode situation specifications. Pseudocode was used specifically as an unrestricted comparison point both in terms of environment attributes, as well as logical and spatial relationships. All teams were able to encode their pseudocode using the elements, attributes and relationships of the Situation Model and State Model.

In relation to participants' interpretation of the charted Alert Report, all participants indicated that they interpreted the charts to mean that inappropriate situations were detected. This provides an indication that participants were able to read and understand the charted output from the Alert Report. Reinforcing this result, six participants were able to correctly describe the recorded instances of inappropriate situations in their own words. Five participants were able to make correct suggestions about how to improve the design of the lighting system, indicating that these participants were able to correctly trace the causal factors of the problems in the system design. This shows that half the participants were able to both (1) correctly determine that inappropriate situations were detected, and (2) correctly make suggestions for improving the design of the lighting system. In general, the context-aware participants had higher appreciation for benefits of InSitu, specifically for identifying inappropriate situations and uncovering unexpected system effects.

The main finding in relation to participants' performance during the configuration part of this trial is that despite the lack of an integrated toolset or dedicated user interface for InSitu, the process of configuring a simulation has been defined in a sufficiently systematic way that participants were able to complete all tasks in a reasonable time-frame. Configuration of the Activity Model showed that with repetition, participants improved in efficiency to complete the activity configuration task. There appears to be no conclusive correlation between the participants' prior experience and the performance of teams completing the user trial tasks.

Although participants perceived the process of configuring a simulation as generally systematic and easy to complete, the results from the previous section on the performance of participants, indicate that there is clearly room to improve the efficiency of the configuration process. In general, in their comments participants identified the realism, and environment visualisation as beneficial to their ability to relate to the environment. Participants also perceived that the flexibility, power and simplicity, in terms of defining inappropriate situation specifications, is a strength of InSitu. The weaknesses noted by participants largely focused on usability which was anticipated prior to the user trial. The suggestions collected from participants can provide a useful basis for prioritising improvements to InSitu particularly to overcome the tedious and time-consuming nature of the configuration process.

6.5 Comparison Framework Analysis

This section revisits the comparison framework that was defined in Chapter 3 as an instrument to draw comparisons between surveyed tools. The discussion on the comparison framework analysis in Chapter 3, noted that most of the current active open issues and emerging needs, relevant to this thesis, lie in the categories of *Testing* and *Evaluation*. For this reason, these will be discussed first, with a focus on differentiating InSitu from existing approaches. Following this, the remaining categories, *Tool Design* and *Deployment Environment*, are discussed, with an emphasis on demonstrating how InSitu conforms in these categories. Figure 6.27 presents the chart of the comparison framework, extended to include InSitu.

One of the emerging needs of the *Testing* category is the facility to monitor distributed ubicomp deployments. The issue of the distributed, invisible interface in multi-user, physical-digital environments raise this challenge. DiaSim [59] uses visual monitoring of a 2D global overview of the environment as an initial contribution to address this issue. Visual monitoring offers the benefit that the designer can observe an overview of distributed locations. However, visual monitoring suffers from poor time scalability since there are limits to the length of time a designer can monitor a simulation and it has the potential to place a significant cognitive load on the tester.

UbiREAL [80] is the only tool among those reviewed that provides automatic monitoring of a ubicomp deployment. Both DiaSim [59] and AmISim [46] have similar ambitions however their publications to date have not shown evidence of this yet. UbiREAL monitors effects in the environment based on a system specification. The monitoring tool uses logical specifications to test for situations and then either checks if a corresponding sequence of events was executed, or checks that a corresponding outcome was achieved.

			1999	1999	2002	2002	2002	2004	2005	2005	2006	2007	2009	2009	2010	2011
			Aware Home	Context Toolkit	Labscape	QuakeSim	UbiWise	Topiary	Paper Prototypes	Ubiquitous Home	UbiReal	VisualRDK	SituVis	DiaSuite & DiaSim	AmISim	InSitu
Live (L) | Simulation (S) | Prototyping (P)			L	P	L	S	S	P	P	L	S	P	P	S	S	S
Tool Design	Reusability		●	●		●	●	●		●	●	●	●	●	●	●
	Extensibility		○	●		●	●	●		○	●	●	●	●	●	●
	Context Generation	User	●	●	●	●	●	●		●	●	●		●	●	●
		Entity	●	●	●	○	●	○		●	●	●		●	●	●
		Location	●	●	●	●	●	●		●	●	●		●	●	●
	Toolkit			●			●	●				●		●		
	Test-bed		●			●	●			●	●			●	●	●
Deployment Environment	Scalability	Physical Space		○		●	●	●			●	○	○	○	○	●
		Entities	○	○	○	○	●	○	●	○	●	○	○	●	●	●
		Users	○	○		○	○	○	○	○	○	○	○	○	●	●
	Heterogeneity	Context Sources	●	●	●	○	●	○		●	●	●		●	●	●
		Environments		○		●	●	●			●	●	○	●	●	●
	Configurability		○	○	○	●	●	●		○	●	●		●	●	●
	Fidelity		●	○	●	○	○		○	●	○	○		○	○	○
	Low Investment			●		●	●	●	●		●	●	●	●	●	●
Testing	Repeatability		○			○	○			○	●			○		○
	Scalability	Time	●		○				○	●	○		○	○	●	●
	Spatio-Temporal Relationship		●	○	●	●	●	○	○	●	○	○		○	○	●
	Visual Monitoring		●		●	●	●	●		●	●		○	●	●	●
	Automatic Monitoring										●	●				●
	Spontaneous/Unanticipated Situations		●		●	○	○	○	○	●			○		○	○
Evaluation	Technical Effectiveness										○	○				
	Situational Appropriateness		●		●					●						●
	Traceable Causal Factors													○		●
	Structured Feedback															●

Legend	
●	Yes
○	Partial
	No

Figure 6.27: Comparison Framework Chart with InSitu.

Although InSitu adopts a similar situation-based approach, to that used in UbiREAL, it differs on two key points. The first is generality of situation specifications, and the second is reporting of outcomes. It is the combination of automatic monitoring, generality of situation specifications, and the opportunity to uncover unanticipated situational instances that differentiates InSitu in the Testing category.

The *Evaluation* category identified open issues surrounding assessment of situational appropriateness, traceability of causal factors and provision of structured feedback, particularly as part of an iterative design cycle. Live test environments provide the best opportunity to gain deep understanding about the effectiveness and appropriateness of ubicomp deployments. However, the limiting factors of the cost, time and resources that need to be invested mean that practically speaking, these test-beds are not accessible to all developers and are not always suitable for use throughout the design cycle. For this reason, alternative approaches are required.

DiaSim and UbiREAL have made notable contributions to the category of Evaluation. UbiREAL provides run-time analysis of a technical specification, whereas DiaSim enables designers to build a prototype from a parametrised specification, which in turn can be visually analysed. DiaSim supports playback of events, however this does not facilitate traceable data analysis for an experiment.

Neither tool addresses the issues of tracing the causal factors surrounding specific events, particularly to isolate spatio-temporal relationships and the combination of physical-digital causal factors. InSitu provides a unique contribution in this respect through the combined ability to identify situations in the global environment state, and support the designer to trace the spatio-temporal causal events leading to the situation.

InSitu has been categorised as a test-bed for the purpose of the comparison framework analysis, primarily because it supports testing rather than prototyping or development of ubicomp systems. The *Tool Design* and *Deployment Environment* sections of the comparison framework analysis illustrated the recurring strengths of simulation-based test-beds for ubicomp. These are features that InSitu should conform to, including reusability, extensibility, context generation, scalability, heterogeneity, configurability and lower investment. The choice of the Half-Life 2 engine at the centre of the InSitu Toolset was core to addressing many of these features. As Trenholme and Smith [113] discuss, modern game engines can be beneficial tools in research environments because they are tested rigorously for performance, reliability and usability.

Half-Life 2 in particular provides sophisticated bot behaviour and animations, which were relevant for implementing the Activity Model. The integration of TATUS context generation functionality, to extend Half-Life 2, provided the sensed context sources that are required in order to test ubicomp systems. Half-Life 2 is also open-source, and its SDK provided the extensible Hammer World Editor, which with the addition of context source types, was suitable for the full simulated deployment environment configuration process, i.e. construction of the physical space, implementation of the Activity Model, and positioning of context sources.

Reusability is addressed by a number of aspects of InSitu's design. The separation of the Model Framework and Test Process allows for the Situation Model, Activity Model and Spatial Model to be reused, or tweaked, for separate test simulations. The Simulation Engine also supports reusability, which was demonstrated through the reuse of a building model from another project in the lighting case study. This same map (building model) also appears in the work by Quinn [91] and Feeney [40].

AmISim aims to support extremely large scale simulations and mentions success of simulating 200,000 users in a 200 floor building. However, the application of simulations of this scale to real-world environments and scenarios, is a research problem in itself, which the authors have yet to address. DiaSim [19] reports on modelling the ENSEIRB school, a three storey building with a floor area of 13,500m^2. UbiREAL does not report on scale and so a comparison cannot be made for it.

InSitu is comparable with DiaSim in terms of physical scale. A demonstrator model of a real three storey building in the Simulation Engine, covers a floor area of 8440m^2 and features 104 rooms. Additionally, InSitu models continuous space, where as DiaSim's model must be discrete across floors because it is a 2D simulator.

The extensibility of the simulator, combined with the integration of TATUS functionality affords heterogeneity of context sources. Although this thesis focused on spatio-temporal information, Trenholme and Smith [108] have reported some success using Half-Life 2's particle system to simulate fire, heat and smoke for emergency evacuation situations.

Finally, the Half-Life 2 engine is low cost, €10.00 at the time of writing this thesis. The major investment is the time required to construct the simulated physical deployment space. Trenholme and Smith report that they constructed a building, using Hammer in three weeks, however they did not report the size of the building. The experience of the KDEG research group has been closer to six weeks for a 2-3 storey building that includes photo-realistic textures.

6.5.1 Summary of Comparison Framework Analysis

In summary, this comparison framework analysis demonstrates that InSitu at least matches other testing and evaluation approaches in the categories of Tool Design and Deployment Environment. The discussion also illustrates the emerging needs addressed by InSitu and in this way differentiates InSitu from other state of the art tools. InSitu is differentiated in particular by supporting situational monitoring in a simulated ubicomp environment and supporting traceability in the physical-digital space.

6.6 Chapter Summary

The evaluations in this chapter set out to (i) investigate whether InSitu can correctly identify examples of real-world inappropriate situations; (ii) demonstrate that the Model Framework can specify tests for multi-user problems in complex built environments, (iii) capture an objective perspective about the suitability of InSitu for testing exhibited ubicomp system behaviour, and (iv) compare InSitu to the state of the art. To achieve this, the evaluation chapter presented two case studies, a user trial, and a state of the art comparison framework analysis.

Section 6.2 discussed a case study of a real-life problematic lighting system. This simple case study performed a single user experiment and assessed situations that arose in the deployment environment based on the user's location information and the state of devices in the environment, considering all individuals in the environment uniformly. This case study demonstrated that InSitu can correctly identify examples of real-world inappropriate situations that exist in a live sensor-driven system. The case study also showed that using a set of boundary conditions, informed by a survey of building occupants, the Activity Model was able to dynamically generate the real-world inappropriate situations without using pre-scripted scenarios.

Section 6.3 builds on the findings from the lighting case study to present a more complex design challenge in an access control case study. To investigate an increase in modelling complexity, the second case study introduced multi-user, multi-location situations and made more extensive use of the spatial modelling capabilities of the Spatial Model. This case study demonstrated reasoning about both physical and operational zones, as well as aggregated zones and varying levels of granularity in the containment spatial relationship. The case study discusses the relevance of the subsumption, exclusivity and overlap relationships to situation specifications and the results that appear in the Alert Report, based on the experiences modelling the access control case study.

Section 6.4 presented results from a user trial which found that participants were able to configure the Activity Model and design the Situation Definition Model for a simulation, which was based on the first case study, the lighting system. Since InSitu ultimately aims to provide a testing approach that can improve the evaluation process for ubiquitous computing systems, it is necessary that the InSitu approach does not exceed the cognitive capabilities of technical users. It was considered important to confirm that potential users of InSitu could relate to the approach of testing situations based on an environment state model. Of the ten participants in this trial, all were able to complete the configuration process and half were able to provide viable design refinements based on a set of charted results.

Section 6.5 presented a comparison of InSitu with related work. The comparison shows that InSitu is unique among the reviewed tools in terms of the generality and formality of the designed and implemented situation-based testing approach. Strengthening the benefit of this, is the capability to provide automatically generated, structured feedback, relating to inappropriate situations detected in a deployment configuration.

Chapter 7

Conclusion

This chapter presents an assessment of the extent to which the thesis objectives were achieved. Following this, the chapter discusses the contribution made by this work as well as possible future work.

7.1 Objectives and Achievements

This thesis hypothesised that a situation-based testing approach could be used to test the summative effect of user activity and ubicomp system behaviour in a simulated ubiquitous computing deployment, and to support assessment of the appropriateness of system behaviour. To pursue this, a research question was set out for this thesis as:

Can situations be identified in a simulated ubicomp deployment, independent of the causal factors, so that structured, traceable feedback can be provided that supports assessment of the appropriateness of ubicomp system behaviour?

To address this research question, the aim of this thesis was to define the situation-based testing approach so that it could be used to support an iterative or evolutionary prototyping cycle. To achieve this, a set of objectives were set out as follows:

O1 To research and survey the state of the art in prototyping, testing and evaluation tools for indoor, location-tracking, ubicomp systems.

O2 To provide a formal definition of the situation-based testing approach by devising a test specification framework.

O3 To define a systematic test process, which includes identification of inappropriate situations and which supports a feedback loop to assist informed design refinement as part of an iterative testing approach.

O4 To realise the framework and test process in a simulation-based test-bed, as a mechanism for evaluating the situation-based testing approach.

O5 To evaluate the suitability and limitations of the proposed situation-based testing approach.

A background review of ubiquitous computing and a state of the art survey of testing approaches were conducted to address research objective **O1**, which was stated as: *to research and survey the state of the art in prototyping, testing and evaluation tools for indoor, location-tracking, ubicomp systems.*

A survey of context and building modelling approaches was conducted to inform the design of the test specification framework, i.e. the InSitu Model Framework. This included both the physical layout of an environment, operational usage and occupant activity, and yielded the Spatial Model and Activity Model. The modelling survey also investigated both context and situation specification models to inform the design of the State Model and Situation Model.

The survey of testing and evaluation tools placed emphasis on those tools that can assess action-based ubicomp systems. The state of the art survey identified that although there has been much emphasis placed on prototyping for ubicomp systems, e.g. Context Toolkit [94], Topiary [69], Paper Prototypes [22], VisualRDK [117], less emphasis has been placed on the testing and evaluation phase of iterative design cycles. This view is supported by Tang et al. [111] who discuss many of the open issues relating to this. Additionally, a number of active and recent examples of research in this area were discussed including UbiREAL [80], VisualRDK [117], DiaSim [59] and AmISim [46], all of which feature simulation-based test-beds and provide evidence of the growing research interest in testing approaches for ubicomp systems.

While existing tools and approaches are making contributions towards testing and evaluation of ubicomp systems, a number of open issues remain. There is a need for tools that can monitor and assess the situational outcomes, caused by exhibited system behaviour, in a physical deployment. Tools that can deliver this in a cost-effective manner can be advantageous in terms of providing a method of testing that is accessible to designers.

To address research objective **O2**, *to provide a formal definition of the situation-based testing approach by devising a test specification framework*, the InSitu Model Framework was defined.

In order to realise generalised situation specifications, the situation-based testing approach required a testable model of the environment. Due to the extent of environment context it is difficult for any single model representation to be universally complete, however, as noted by Clear et al. [27], it can be domain specific. The focus of this thesis is the spatio-temporal relationship that exists in the environment and that is central to the unique dependency between users, their environment and the appropriateness of exhibited ubicomp system behaviour at a given point in space and time.

The Spatial Model builds on previous hybrid hierarchical models. To reflect the dynamism of ubicomp environments, this model introduced operational zones to assist with modelling building usage. The Situation Model observes the logical operators defined by Henricksen and Indulska [50] and observes Clear et al.'s [27] definition of a situation specification. The State Model places an emphasis on spatio-temporal reasoning to support testing location tracking systems for indoor environments. The Activity Model follows the trend of activity based models, featuring the location and duration of activities, as well as relationships between activities.

The combination of these models in a situation-based testing approach is novel in terms of the generality of the approach. This is advantageous because generalised situation specifications support identification of unanticipated situational instances, particularly with the support of dynamically generated user activity by the Activity Model. The limitation, in terms of uncovering unanticipated situations, is that the designer must have the foresight to specify the general situation. However, the iterative nature of the testing approach lends itself well to situation specifications that evolve and emerge between testing cycles. It was the experience of the author during the access control case study evaluation that feedback from test results informed the definition of additional or refined situation specifications.

To address research objective **O3**, *to define a systematic test process, which includes identification of inappropriate situations and which supports a feedback loop to assist informed design refinement as part of an iterative testing approach*, the InSitu Test Process was defined.

The InSitu Test Process defines the situation-based testing approach within the bounds of an iterative prototyping cycle, with an emphasis placed on testing and evaluation for the purpose of generating structured feedback to the designer. The Test Process defines the key features that must be implemented to realise the situation-based testing approach proposed in this thesis. These include the central role of simulation, which due to the affordance of capturing a state view of a simulated ubicomp deployment, provides an advantage not available in live test environments. In this way, it supports the tester to analyse the effect of ubicomp system behaviour, as opposed to technical execution of the ubicomp system. Also core to the approach are separation of state and sensed context and generation of alerts to automatically notify the tester of instances of detected situation specifications. The definition of the InSitu Test Process in the form of a UML activity diagram illustrated the systematic process that is central to the testing approach.

To address research objective **O4**, *to realise the framework and test process in a simulation-based test-bed, as a mechanism for evaluating the situation-based testing* *approach*, the InSitu Toolset was implemented.

The InSitu Toolset combines and extends the technology of the TATUS Simulator and Proxy, the Adaptive Engine and the Drools Engine. At run-time the Toolset generates an Alert Report which provides a structured set of alerts that record the situational instances detected during testing. Scatter plots are generated using the combined sources of the Alert Report and test log files, to provide meaningful representation of test results as feedback to a tester or designer. These charts provide configurable views of test data at varying granularities on a simulation timeline.

The frequency of alert generation is closely linked with the frequency at which state updates are produced by the simulator. The Monitoring Engine must only perform an assessment of the state of the environment at least as fast as the Simulation Engine generates state updates. Both the frequency of the Monitoring Engine cycles and the frequency of state update generation are configurable, but in practical terms will be limited by the processing power of the hardware. Although this thesis produced no conclusive result to inform the configuration of this frequency, it is suggested that a balance between criticality and duration should be taken into account during test configuration.

The duration of a situation can be indicative of the severity of the problem. For example, in some simulations, the Monitoring Engine detected normal delays that occur between the time a motion sensor is activated and the time the lights actually switch on, i.e. approximately 1 second. It is not expected that this should be considered an inappropriate situation that warrants redesign. Situations of long duration do not require highly frequent rule cycles and, in fact, overly frequent rule cycles will cause an unnecessary increase in the post simulation data processing task. Instead, test log files could be used to supplement the Alert Report findings in order to uncover the start and end points of a situational instance, even if the Monitoring Engine does not capture the full duration. Critical situations, on the other hand, should be prioritised for prevention prior to a live deployment and so highly frequent state updates and Monitoring Engine test cycles may be desirable for testing these situations.

The combined findings from the case studies, user trial and comparison framework analysis are presented to address research objective **O5**, *To evaluate the suitability and limitations of the proposed situation-based testing approach.*

The overwhelming benefit of the situation-based testing approach is the capability to test the summative effect of user and system activity in a simulated physical deployment environment. In the lighting case study it was shown that the InSitu approach correctly identified examples of real-world inappropriate situations that arose due to a specific combination of system behaviour and user activity. A secondary finding of the lighting case study was an indication of the relevance of spatio-temporal relationships to the appropriateness of sensor-driven system behaviour, and the impact of both the absence and presence of end-users on system behaviour. This supports the relevance of the InSitu approach as a testing approach for ubiquitous computing systems.

Although situation-based testing has appeared in the literature in the past, e.g. in the work on UbiREAL [80], it previously lacked generality in test specifications. The significant benefit of InSitu is that generalised situation specifications provide greater range to detect instances of inappropriate situations. The result is that InSitu can uncover unanticipated situational instances. In the access control case study it was shown that the InSitu approach could identify situational instances for multiple users, across multiple locations, in both physical and operational zones at various levels in the hierarchy of a built environment. Additionally, since situations can be defined independent of a particular sequence, any ordering of specified situations can be detected and so they are not confined to a procedurally scripted scenario.

A second major benefit of this situation-based testing approach is the feedback supported by InSitu that offers the potential to inform both software and hardware configurations of ubicomp deployments. Processing of the Alert Report and simulation logs provides detailed, configurable views of detected situations and the causal physical and digital events leading to individual situations. Interpretation of these data charts by a designer provides insight into the appropriateness of exhibited system behaviour and to determine how design refinement could improve the end-user experience. The user trial provided an objective assessment, by a group of computer scientists, which confirmed that participants could relate to the approach and concepts used by InSitu, configure simulations, define situation specifications, and interpret results to mean that situational instances were detected. Eight participants were able to correctly describe the recorded instances of inappropriate situations, indicating that these participants were able to correctly interpret charted results. Five participants were able to make correct suggestions about how to improve the design of the lighting system, indicating that these participants were able to correctly trace the causal factors of the problems in the system design.

Although situation-based testing has the potential to address a significant challenge for ubicomp designers through the support it offers for testing the effects and outcomes of ubicomp system behaviour, some limitations exist. The first centres on the capabilities of InSitu to uncover unanticipated outcomes. Although the InSitu approach offers the potential to uncover unanticipated situational instances, it does not directly alleviate the issue of uncovering unexpected situation specifications. Testing methods which use human observers, particularly in live test deployments, have an advantage in this respect because human reasoning capabilities can identify truly unanticipated situations. However, as already mentioned in this section, the iterative testing process used for InSitu lends itself well to situation specifications that evolve with testing.

The second limitation centres on completeness of both testing and reporting. The situation specifications presented in the case studies in this thesis, test only a subset of all situations that arise during testing. Additionally, completeness of reporting is also a factor. Regardless of how well the continuity of the spatio-temporal dimension is maintained during testing, both testing and reporting are ultimately conducted at discrete points, which introduces the risk that inappropriate situations will go undetected. The risk can be managed by considering the issues of criticality and duration of situations when configuring a simulation. In practical terms, however, a limiting factor could be the processing power and memory of the hardware available for testing.

7.2 Contribution to SOA

The major contribution of this work is a novel situation-based testing approach for examining the effect of ubicomp system behaviour in a simulated deployment environment. The use of generalised situation specifications enables the situation-based testing approach to examine the summative effects and outcomes of exhibited system behaviour and user activity in a simulated ubicomp deployment environment. This differs from previous testing approaches that examine only system behaviour, without investigating the implications for the state of the deployment environment as a whole, and as a result cannot assess the appropriateness of the effects of system behaviour. This is important as ubicomp systems can be technically and functionally correct in their behaviour but still cause unintended effects in the deployment environment, if they do not operate in harmony with the target deployment environment. Determining that a system implementation is correct with respect to its design will not always verify that the design is appropriate for the target physical deployment environment. Additionally, the generality of the InSitu approach provides the opportunity to uncover unanticipated situational instances because tests can be applied globally to the deployment environment.

The minor contribution of this work is the traceability of causal factors that lead to specific situations in a simulated physical deployment environment. Traceability of situational causal factors in ubicomp environments introduces new challenges for the designer of ubicomp systems due to the loose-coupling and the spatio-temporal distribution between system inputs and system effect. It is not sufficient to trace the causal relationships in the digital environment alone. The structured feedback generated by InSitu provides a means to examine the physical and digital events leading to specific situations. Traceability augments the benefits of the situation-based testing approach, because identification of inappropriate situations alone is not sufficient to inform the redesign process.

7.2.1 Peer Reviewed Publications

This work has been peer-reviewed and has been accepted to a number of international conferences and to the International Society for Simulation and Modelling *SIMULATION* Journal. A list of relevant author publications appear at the front of this thesis, the most significant of which are noted here.

The most comprehensive publication to date of the InSitu approach and tool, appears in the Society for Computer Simulation International SIMULATION Journal in a special issue on *Tools, Techniques and Methodologies.* This paper presents the InSitu Model Framework and InSitu Toolset, along with the performance test results for the InSitu Toolset, and the findings from the lighting case study.

O'Neill, E., Conlan, O. and Lewis, D., *'Modeling and Simulation to Assist Context Aware System Design'*, Simulation: Transactions of the Society of Modeling and Simulation International, vol. 87(1-2), pp. 149 - 170, 2011.

This SCS Simulation paper is an extension of a previous peer-reviewed conference paper presented at the 2nd International Conference on Simulation Tools and Techniques, SimuTools '09. This conference paper documents early requirements for InSitu alongside an early implementation of the InSitu Model Framework and InSitu Toolset.

O'Neill, E., Lewis, D., and Conlan, O., *A Simulation-based Approach to Highly Iterative Prototyping of Ubiquitous Computing Systems*, 2nd International Conference on Simulation Tools and Techniques, Rome, Italy, 2-6th March 2009.

The following peer reviewed conference paper was presented at the XIII International Workshop on Design, Specification and Verification of Interactive Systems

(DSV-IS 2006). This paper discusses the Simulation Engine and Update Router, and describes experiences configuring the simulation environment, as well as integrating two independently developed applications with the Update Router; (i) a location-aware instant messaging service, and (ii) a community based policy management system.

O'Neill, E., Lewis, D., McGlinn, K., and Dobson, S., *Rapid User-Centred Evaluation for Context-Aware Systems*, XIII International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 2006), Dublin, Ireland, July 2006, LNCS 4323, Springer, pp. 220 - 233, 2006.

7.3 Future Work

There are a number of practical and research issues in which there is potential for extensions and advances to this work. A subset of these are discussed in this section, along with some proposed approaches for addressing these future work issues.

7.3.1 Debugging Ubicomp Systems

A recurring theme in recent research publications has been the need for debugging tools for ubiquitous computing systems [111,59,46,80]. Traditional debugging tools have generally been limited to identifying and tracing bugs in a program's source code. This however is not sufficient for ubiquitous computing because the causal factors and system effects that need to be debugged are distributed throughout the physical and digital deployment space. Tang et al. [111] suggest that there is promise in *"A simulation debugging mode, which allows the developers to debug a prototype early even if the actual physical device is absent in the debugging environment"*.

InSitu has been designed to identify and resolve non-technical bugs, i.e. unwanted or inappropriate behaviour, in a ubiquitous computing system. The significance of this is that it allows the designer to examine the physical effects of a prototype system, in the context of a target deployment environment. The limitation is that InSitu does not enable the designer to examine the internal process of the system. However, a combined approach that includes both InSitu and traditional debugging techniques has the potential to be mutually complimentary for the testing process for ubiquitous computing systems. While InSitu offers the possibility to trace causes and artefacts of system behaviour within the context domain for a specific situation, a more traditional debugging approach can focus on following the control flow and execution of a ubicomp system.

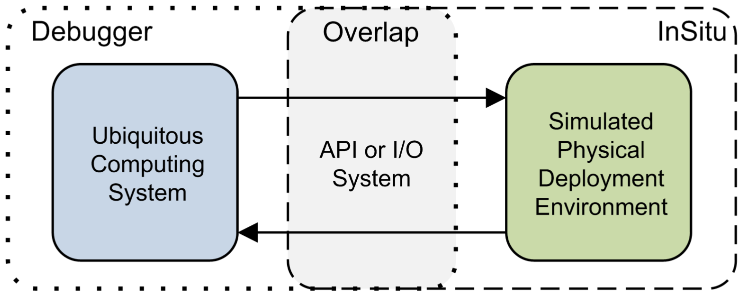

Debugger
Overlap
InSitu
Ubiquitous Computing System
API or I/O System
Simulated Physical Deployment Environment

Figure 7.1: Tangential roles for InSitu and a debugger.

Figure 7.1 illustrates the relationship between InSitu, a traditional debugging approach, the prototype ubicomp system and the simulated physical deployment environment. The area of overlap indicates the meeting point of the monitoring capabilities for each tool, which is the key enabler to this potential future work. This provides a point of reference that has the potential to allow the results from these two different approaches to be reconciled. A combination of these approaches has the potential to enable a designer to trace causal factors throughout the physical and virtual design space. Of particular relevance for achieving this is research, such as VisualRDK [117], which places emphasis on enabling debugging for the *distributed* digital deployment environment.

A further challenge in pursuing this future work is the additional complexity introduced when an intermediary Context Management System (CMS) must also be considered during the traceability process. This may require additional techniques to support traceability, specifically where processing of sensor data and its mapping to context inputs is concerned.

7.3.2 Complete Analysis

An open issue reported in the literature, for ubicomp testing and which is relevant to InSitu, is the challenge of completeness of testing for ubicomp systems. Nishikawa et al. [80] discuss this issue, reporting that it is infeasible for the UbiREAL situation-based testing approach to examine all possible sensor state values. The UbiREAL approach uses a predefined subset of values that aims to find a balance between thoroughness of testing and investment required for testing, but which is ultimately the responsibility of the tester to define.

It is suggested here, however, that steps could be taken to assist the designer by providing informative data views that show the extent of testing in the design and context spaces. Of interest in this respect is the work by Padovitz et al. [84] who described a conceptual framework for a context model that is centred on *context domains*, *situation subspaces* and *context states*. A context state can be considered the equivalent of a *situation*, as referred to in this thesis. A situation subspace is defined by a set of bounded attributes relevant to a particular situation. A single attribute example provided by Padovitz et al. is the range for heart rate (BpM) that is applicable to the situation subspaces for running or walking. Running is listed as ranging between $150 - 200BpM$, while walking is listed as ranging between $100 - 160BpM$. This illustrates a boundary within which each situation occurs, and the overlap or intersection that exists between the two subspaces. This is similar to the role of boundaries used in the Activity Model, in which boundaries are used to assist the designer to configure tests that generate relevant and reasonable occupant behaviour.

Using information visualisation techniques, the aim would be to highlight areas of the design and context spaces that are tested so the designer is informed about the level of completeness of testing. The intention would be to provide the designer with the opportunity to configure InSitu simulations to assess previously untested parts of the design and context spaces. Initial investigation has been conducted into adoption of a tool or an approach to situation visualisation, such as that used by Situvis [28], in order to provide the designer with a clearer view of the coverage of a simulation. Although the underlying data-sets implemented for InSitu and Situvis are not directly compatible, the situation specification model used by both tools is fundamentally the same and so there is potential to adopt a similar approach. A major challenge to overcome however, as noted by Clear et al., is visualisation of the temporal relationships across instances of situations. There are also other considerations that need to be taken into account such as scalability of visualisations, and granularity and density of the data that must be represented.

A more advanced approach might include support to monitor the progress of a test in order to direct testing at run-time. It is envisaged that this would increase the role of the Adaptive Engine during InSitu's monitoring process. Adaptation of a simulation on the fly could allow for more targeted testing of the prototype ubicomp application. It is anticipated that existing functionality in the tool could be used to address this. In much the same way that the prototype ubicomp system changes the state of the simulated environment by issuing actuation instructions to the game engine, there is potential for InSitu to direct bots and activities in the environment, based on an informed view of the extent of previously tested space.

7.3.3 Context Histories and Run-time Temporal Analysis

The situation-based testing approach presented in this thesis, in the form of InSitu, provides functionality to perform run-time analysis of spatial relationships and post-testing analysis of spatio-temporal relationships. This section discusses the potential for run-time spatio-temporal analysis of situations. Henricksen et al. [51] mention the relevance of the temporal aspects of context while Yi et al. [54] identified a set of seven key temporal relationships for spatio-temporal modelling. These temporal relationships focus on the start and end points of a time interval, and the duration between these points. For example, two time spans can *overlap*, be *equal*, can *start* or *end before* or *after* one another, or they can *meet*, i.e. one time span ends at the same time that the other starts.

Drools Fusion, a recent extension to the Drools Engine, is proposed as an initial tool to address this for InSitu. Drools Fusion supports complex event processing which provides temporal reasoning for all seven of Yi et al.'s identified temporal relationships. Drools Fusion has additional benefits such as reasoning about absent events and support for sliding windows, e.g. a moving window of time.

The significant benefit of this extension to InSitu is that it would enable the designer to perform stateful analysis of situations, as opposed to the stateless monitoring which has been used in the case studies presented in this thesis. This is a necessary step in order to expand the reasoning capabilities of InSitu to include context histories for situational analysis.

7.3.4 Extensions to the Model Framework

Activity Model

The Activity Model adopted for InSitu follows the trend of activity-based models, placing particular emphasis on the location and duration of each activity. Spatio-temporal bounds define limits on reasonable occupant behaviour within which randomness can be applied. Alternative approaches that can be adopted for this model can include statistical weighting on the sequencing or scheduling of activities. Recent work by Tabak and de Vries [110] investigated the application of probabilistic and S-curve methods to an activity-based model. This work shows promise, particularly in differentiating between activities which increase in urgency with time (S-curve methods), as opposed to activities which are not influenced by the passing of time. However, the findings are not yet fully conclusive. Future work on the InSitu Activity Model could include evaluation of the application of these various approaches and their impact on the situation-based testing approach, specifically to understand the biases or prejudicial tendencies that each model may introduce during testing.

An additional extension to the Activity Model could place more emphasis on bot profiles. The introduction of profiled behaviour could enable individual bots to more clearly assume roles, e.g. in a university setting two roles might be a professor and a student. These two user types in general have different working patterns. Professors tend to have many appointments during their day, whereas students can often have days with no meetings during which they are working on an individual piece of work with little need for collaboration.

State Model

The State Model was demonstrated to be sufficient for post-simulation analysis of the spatio-temporal relationships between users, entities and location for indoor, location-tracking systems. Extensions to this model could allow for more extensive testing of a wider set of applications. Current directives[1]for energy conservation and energy efficient buildings, along with the suitability of sensor driven technology as an enabling factor for smart buildings, are motivators to address this issue in InSitu. Building usage and occupancy have a significant impact on the performance of facility management systems. In much the same way that architects and engineers model and simulate structures before entering the construction phase, the computer systems which control these buildings can benefit from the same process. For this reason, InSitu is a relevant tool for designing these buildings. Extensions to the State Model would be required to address more of the physical aspects of the deployment environment, e.g. temperature and humidity.

The work by Nishikawa et al. [80] for UbiREAL demonstrated a situation-based testing approach that includes physical effects in the environment, e.g. temperature and humidity. Smith and Trenholme et al. [108] have demonstrated the use of Half-Life 2 for simulating the physical effects of fire for the purpose of investigating emergency evacuations in buildings. More specific to this thesis and the extensibility of TATUS [82], Chubb [26] conducted some initial work to extend the TATUS simulator to provide an augmented reality experience. This combined live sensor readings from a real building, with the virtual experience of the building's simulated counterpart. Additionally, McGlinn [75] has performed work to provide more control

[1]On 18th May 2010, EU Directive 2010/31/EU reinforced Directive 2002/91/EC as the main legislative instrument to achieve energy performance in buildings. Directive 2002/91/EC was key in the introduction of minimum standards on the energy performance of new and renovated buildings.

over the configuration of sensor models used in TATUS.

7.3.5 Technical Implementation and Further Evaluation

This section describes extensions which could be made to InSitu and TATUS to increase the test-bed's applicability and usability. The addition of a context management system and a network simulator could enable InSitu to be used later in the design cycle by increasing the fidelity of the virtual test environment. It is also necessary that InSitu be made easier and more efficient to use if it is to be applied to projects of any significant size.

Context Management System

Although the Update Router and XML API has been used as an interface to the TATUS test-bed by a number of researchers [91, 40, 26], a more standardised interface is desirable. The approach adopted by AmISim, using an open source context management system (CMS) called the Open Context Platform (OCP), has three main advantages. Firstly, it allows applications to be switched between real and simulated deployment environments without making any changes. Secondly, it provides access to more sophisticated context processing. Thirdly, in the case of OCP, it is an open source project which is beneficial for InSitu because it could be extended to support traceability.

Network Simulation

Although network simulation was not core to the research challenge identified for this thesis, networking and connectivity is a crucial enabling factor for ubiquitous computing environments. Previous projects by Huebscher and McCann [55] and Morla and Davies [78] demonstrated the successful integration of existing network simulators into ubicomp simulation-based test-beds. Although not directly evaluated for this thesis, previous work with the TATUS simulator looked at integration of the simulator with a heterogeneous wireless network simulator from

Cork Institute of Technology (CIT) [83]. More recently, through the SimCon toolset, the ubiquitous computing simulator has been loosely integrated with CIT's Wi-Design Tool [75]. TATUS and InSitu could benefit from further evaluation to assess the validity of the integration with these tools.

Integrated Toolset

Although the InSitu Test Process was accepted and successfully used to create correct simulations, the process is tedious and prone to configuration error, particularly at the level of complexity required for the access control case study. In general it would be expected that much of the process involved in setting up a test simulation could be automated or could be improved with an integrated user interface. In the case of the Activity Model, a directed graph could provide designers with an intuitive visual aid to show the links between activities. This could be interactive, e.g. drag and drop, and should automatically generate the activity model configuration for a simulation. In relation to the rest of the Model Framework, the tight links between the Spatial, State and Situation Models make them an obvious choice for an integrated user interface. Development of an integrated toolset could increase efficiency of the configuration process, while also minimising the risk for error due to manual data entry.

7.4 Final Remarks

The state of the art review showed that no single tool has emerged to address the many challenges associated with testing ubicomp systems. The review also showed that emerging challenges specific to ubiquitous computing systems raise the need for new testing approaches. Of particular relevance and interest to this thesis is the challenge of testing the actual effect of ubicomp system behaviour in the deployment environment, when appropriateness of system behaviour is subject to the effects of other activity occurring in parallel.

Testing approaches that examine only system behaviour, without investigating the implications in the context of the actual state of the physical environment, cannot assess the appropriateness of the effects of system behaviour. In realising InSitu, it contributes an important and novel testing approach to the field, because ubicomp systems can be technically and functionally correct in their behaviour but still cause unintended outcomes in the deployment environment.

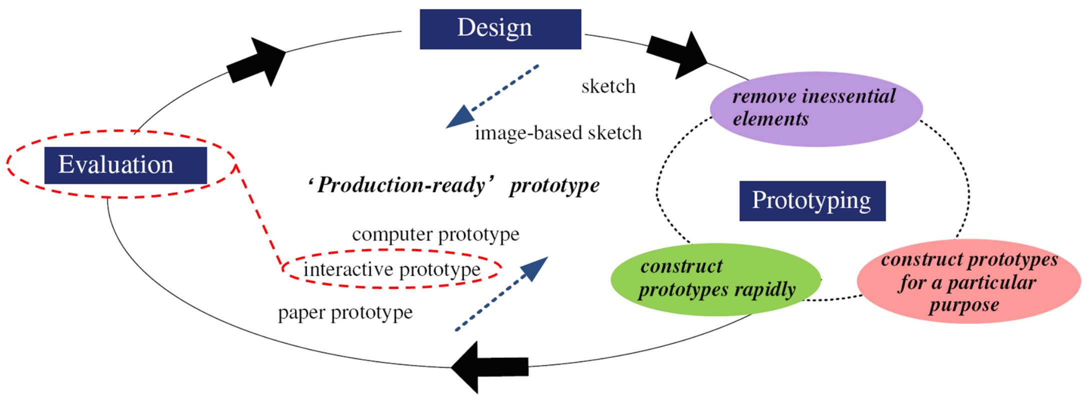

Design
sketch
image-based sketch
remove inessential elements
Evaluation
'Production-ready' prototype
Prototyping
computer prototype
interactive prototype
paper prototype
construct prototypes rapidly
construct prototypes for a particular purpose

Figure 7.2: Contribution of InSitu for evaluating computer prototypes, in the context of Tang et al.'s [111] model of a user centric, evolutionary prototyping cycle for ubiquitous computing (Indicated by the red dashed lines).

The application of InSitu for testing ubicomp systems is indicated in Figure 7.2, which reproduces Tang et al.'s [111] succinct representation of the phases of a user-centric iterative prototyping cycle for ubiquitous computing. InSitu makes a contribution to the field by providing support for testing computer-prototypes in the evaluation phase of this cycle.

InSitu has not been presented in this thesis as a panacea to address the many challenges of testing ubiquitous computing systems. Rather, it is presented as a contribution to the field alongside the many other emerging tools that are making valuable advances in the field. It is envisaged that InSitu can make a significant contribution to a user-centric, evolutionary prototyping cycle, and that it can compliment existing tools and approaches.

Bibliography

[1] Gregory D. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal*, 38(4):508–530, 1999.

[2] Gregory D. Abowd. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering (ICSE '99)*, pages 75–84. ACM, New York, NY, USA, 1999.

[3] Gregory D. Abowd, Christopher G. Atkeson, Aaron F. Bobick, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, and Thad E. Starner. Living laboratories: the future computing environments group at the georgia institute of technology. In *CHI '00 extended abstracts on Human factors in computing systems (CHI '00)*, pages 215–216. ACM, New York, NY, USA, 2000.

[4] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing*, volume 1707, pages 304–307. Springer Berlin / Heidelberg, September 1999.

[5] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):29–58, 2000. Special issue on human-computer interaction in the new millennium, Part 1.

[6] Meenakshi Balasubramanian, Namit Chaturvedi, Atish Datta Chowdhury, and Arul Ganesh. 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 (SUTC'06)*, volume 1, pages 306–311. IEEE Computer Society, 2006.

[7] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[8] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: an application model for pervasive computing. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom '00)*, pages 266–274. ACM, New York, NY, USA, 2000.

[9] Guruduth Banavar and Abraham Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. *Communications of the ACM*, 45(12):92–96, 2002.

[10] S. Bandini, A. Mosca, and M. Palmonari. A hybrid logic for commonsense spatial reasoning. In *Advances in Artificial Intelligence (AI*IA 2005)*, volume 3673, pages 25–37, 2005.

[11] Stefania Bandini, Alessandro Mosca, and Matteo Palmonari. Commonsense spatial reasoning for contextaware pervasive systems. In *Location- and Context-Awareness*, volume 3479 of *Lecture Notes in Computer Science*, pages 180–188. Springer Berlin / Heidelberg, 2005.

[12] David Bannach, Oliver Amft, and Paul Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7(2):22–31, 2008.

[13] John J. Barton and Vikram Vijayaraghavan. UBIWISE, a ubiquitous wireless infrastructure simulation environment. Technical Report HPL-2002-303, Mobile and Media Systems Laboratory, HP Laboratories, Palo Alto, 2002.

[14] John J. Barton and Vikram Vijayaraghavan. UBIWISE, a simulator for ubiquitous computing systems design. Technical Report HPL-2003-93, Mobile and Media Systems Laboratory, Hewlett Packard Laboratories Palo Alto, 2003.

[15] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[16] Christian Becker and Daniela Nicklas. Where do spatial context-models end and where do ontologies start? A proposal of a combined approach. In *First International Workshop on Advanced Context Modelling, Reasoning and Management (Ubicomp 2004)*, 2004.

[17] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, April 2010.

[18] P. J. Brown and G. J. F. Jones. Context-aware retrieval: Exploring a new environment for information retrieval and information filtering. *Personal and Ubiquitous Computing*, 5(4):253–263, 2001.

[19] Julien Bruneau, Wilfried Jouve, and Charles Consel. DiaSim: A parameterized simulator for pervasive computing applications. In *Mobile and Ubiquitous Systems: Networking & Services (MobiQuitous '09)*, pages 1–2. HAL - CCSD, 2009.

[20] Markus Bylund and Fredrik Espinoza. Using Quake III Arena to simulate sensors and actuators when evaluating and testing mobile services. In *CHI '01 extended abstracts on Human factors in computing systems*, CHI '01, pages 241–242, New York, NY, USA, 2001. ACM.

[21] Markus Bylund and Fredrik Espinoza. Testing and demonstrating context-aware services with Quake III Arena. *Communications of the ACM*, 45(1):46–48, January 2002.

[22] S. Carter and J. Mankoff. Prototypes in the wild: Lessons from three ubicomp systems. *IEEE Pervasive Computing*, 4(4):51–57, 2005.

[23] Valentina Casola, Luca D'Onofrio, Giusy Di Lorenzo, and Nicola Mazzocca. A service-based architecture for the interoperability of heterogeneous sensor data: A case study on early warning. In *Geographic Information and Cartography for Risk and Crisis Management*, volume 2 of *Lecture Notes in Geoinformation and Cartography*, pages 249–263. Springer Berlin Heidelberg, 2010.

[24] Damien Cassou, Julien Bruneau, and Charles Consel. A tool suite to prototype pervasive computing applications (demo). In *Proceedings of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM '10)*, pages 1–3. IEEE Computer Society Press, 2010.

[25] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(3):197–207, 2003.

[26] Eric Chubb. Context information as augmented reality. M.Sc. dissertation, Trinity College Dublin, 2008.

[27] Adrian K. Clear, Thomas Holland, Simon Dobson, Aaron Quigley, Ross Shannon, and Paddy Nixon. Situvis: A sensor data analysis and abstraction tool for pervasive computing systems. *Pervasive Mobile Computing*, 6:575–589, October 2010.

[28] Adrian K. Clear, Ross Shannon, Thomas Holland, Aaron Quigley, Simon Dobson, and Paddy Nixon. Situvis: A visual tool for modeling a user's behaviour patterns in a pervasive environments. In *Pervasive Computing*, volume 5538 of *Lecture Notes in Computer Science*, pages 327–341. Springer Berlin / Heidelberg, 2009.

[29] Owen Conlan, Vincent Wade, Catherine Bruen, and Mark Gargan. 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 (AH '02)*, volume 2347, pages 100–111, London, UK, 2002. Springer-Verlag.

[30] Sunny Consolvo, Larry Arnstein, and B. Robert Franza. User study techniques in the design and evaluation of a ubicomp environment. In *Ubiquitous Computing (UbiComp 2002)*, volume 2498 of *Lecture Notes in Computer Science*, pages 73–90. Springer-Verlag, 2002.

[31] Diane J. Cook and Sajal K. Das. How smart are our environments? An updated look at the state of the art. *Pervasive and Mobile Computing*, 3(2):53–73, 2007.

[32] W. H. Dana. The X-15 lessons learned. Technical Report AIAA930309, NASA Dryden Research Facility, 1993.

[33] N. Davies. Proof-of-concept demonstrators and other evils of application-led research: A position statement. In *Proceedings of UbiApp Workshop*, Munich, Germany, 2005.

[34] N. Davies, J. A. Landay, S. Hudson, and A. Schmidt. Guest editors' introduction: Rapid prototyping for ubiquitous computing. *IEEE Pervasive Computing*, 4(4):15–17, 2005.

[35] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[36] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *CHI' 2000 Workshop on the What, Who, Where, When and How of Context Awareness*, 2000.

[37] E. Dias and E. Beinat. *Measuring the Impact of Location-Awareness in the Acceptance of Mobile Systems*, chapter 9, pages 137–154. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, 2008.

[38] Fredrik Espinoza, Per Persson, Anna Sandin, Hanna Nyström, Elenor Cacciatore, and Markus Bylund. GeoNotes: Social and navigational aspects of location-based information systems. In *Ubicomp 2001: Ubiquitous Computing*, volume 2201/2001, pages 2–17. Springer-Verlag, 2001.

[39] M. Fahrmair, W. Sitou, and B. Spanfelner. Unwanted behavior and its impact on adaptive systems in ubiquitous computing. In *14th Workshop on Adaptivity and User Modelling in Interactive Systems (ABIS '06)*, Hildesheim, Germany, 2006.

[40] Kevin Feeney. *Policy Based Management for Dynamic Organisations.* Ph.D. thesis, Trinity College Dublin, 2007.

[41] Niclas Finne, Joakim Eriksson, Adam Dunkels, and Thiemo Voigt. Experiences from two sensor network deployments: self-monitoring and self-configuration keys to success. In *Proceedings of the 6th international conference on Wired/wireless internet communications (WWIC'08)*, pages 189–200. Springer-Verlag, Berlin, Heidelberg, 2008.

[42] Margaret Fleck, Marcos Frid, Tim Kindberg, Eamonn OBrien-Strain, Rakhi Rajani, and Mirjana Spasojevic. Rememberer: A tool for capturing museum visits. In *UbiComp 2002: Ubiquitous Computing*, volume 2498 of *Lecture Notes in Computer Science*, pages 379–385. Springer Berlin / Heidelberg, 2002.

[43] Charles Forgy. *On the efficient implementation of production systems.* Ph.D. thesis, Carnegie-Mellon University, 1979.

[44] Michael Friedewald, Olivier Da Costa, Yves Punie, Petteri Alahuhta, and Sirkka Heinonen. Perspectives of ambient intelligence in the home environment. *Telematics and Informatics*, 22(3):221–238, 2005.

[45] Teresa García-Valverde, Alberto García-Sola, Francisco Lopez-Marmol, and Juan Botia. Engineering ambient intelligence services by means of MABS. In *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71

of *Advances in Soft Computing*, pages 37–44. Springer Berlin / Heidelberg, 2010.

[46] Teresa García-Valverde, Emilio Serrano, and Juan Botia. Social simulation for ami systems engineering. In *Hybrid Artificial Intelligence Systems*, volume 6076 of *Lecture Notes in Computer Science*, pages 80–87. Springer Berlin / Heidelberg, 2010.

[47] Dedre Gentner and Albert L. Stevens. *Mental Models.* Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, NJ, 1983.

[48] Tom Gilb. *Software Metrics.* Little, Brown, and Co., 1976.

[49] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom '99)*, pages 59–68. ACM, New York, NY, USA, 1999.

[50] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37 – 64, February 2006.

[51] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing (Pervasive '02)*, volume 2414 of *Lecture Notes In Computer Science*, pages 167–180, London, UK, 2002. Springer-Verlag.

[52] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *IEEE Computers*, 34(8):57–66, 2001.

[53] P. Hoes, J.L.M. Hensen, M.G.L.C. Loomans, B. de Vries, and D. Bourgeois. User behavior in whole building simulation. *Energy and Buildings*, 41(3):295–302, 2009.

[54] Bo Huang, Shanzhen Yi, and Weng Tat Chan. Spatio-temporal information integration in XML. *Future Generation Computer Systems*, 20(7):1157–1170, 2004.

[55] Markus C. Huebscher and Julie A. McCann. Simulation model for self-adaptive applications in pervasive computing. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA '04)*, pages 694–698. IEEE Computer Society, Washington, DC, USA, 2004.

[56] Stephen S. Intille, Kent Larson, J. S. Beaudin, J. Nawyn, E. Munguia Tapia, and P. Kaushik. A living laboratory for the design and evaluation of ubiquitous computing technologies. In *CHI '05 extended abstracts on Human factors in computing systems (CHI '05)*, pages 1941–1944. ACM, New York, NY, USA, 2005.

[57] Changhao Jiang and Peter Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In *Proceedings of the 4th international conference on Ubiquitous Computing (UbiComp '02)*, pages 246–263. Springer-Verlag, London, UK, 2002.

[58] S. Jones. Digital era 'will make prime-time television obsolete'. Technical Report September 14th, Financial Times (London), 1999.

[59] Wilfried Jouve, Julien Bruneau, and Charles Consel. DiaSim: A parameterized simulator for pervasive computing applications. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom 2009)*, pages 1–3, Los Alamitos, CA, USA, March 2009. IEEE Computer Society.

[60] Wilfried Jouve, Julien Lancia, Nicolas Palix, Charles Consel, and Julia Lawall. High-level programming support for robust pervasive computing applications. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '08)*, pages 252–255. IEEE Computer Society, Washington, DC, USA, 2008.

[61] Wilfried Jouve, Nicolas Palix, Charles Consel, and Patrice Kadionik. A SIP-based programming framework for advanced telephony applications. In *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, volume 5310 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin / Heidelberg, 2008.

[62] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The Aware Home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture (CoBuild '99)*, pages 191–198. Springer-Verlag, London, UK, 1999.

[63] Julie A. Kientz, Shwetak N. Patel, Brian Jones, Ed Price, Elizabeth D. Mynatt, and Gregory D. Abowd. The Georgia Tech Aware Home. In *CHI '08 extended abstracts on Human factors in computing systems (CHI '08)*, pages 3675–3680. ACM, New York, NY, USA, 2008.

[64] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, October 2002.

[65] Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, January 2002.

[66] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, April 2006.

[67] Craig Larman and Victor R. Basili. Iterative and incremental development: A brief history. *IEEE Computer*, 36(3):47–56, June 2003.

[68] Jumphon Lertlakkhanakul, Jin Won Choi, and Mi Yun Kima. Building data model and simulation platform for spatial interaction management in smart home. *Automation in Construction*, 17(8):948–957, November 2008.

[69] Y. Li, J. I. Hong, and J. A. Landay. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST '04)*, pages 217–226. ACM, New York, NY, USA, 2004.

[70] N. Liogkas, B. MacIntyre, E.D. Mynatt, Y. Smaragdakis, E. Tilevich, and S. Voida. Automatic partitioning for prototyping ubiquitous computing applications. *IEEE Pervasive Computing*, 3(3):40–47, July-Sept. 2004.

[71] Linchuan Liu and Peter Khooshabeh. Paper or interactive?: A study of prototyping techniques for ubiquitous computing environments. In *CHI '03 extended abstracts on Human factors in computing systems (CHI '03)*, pages 1030–1031. ACM, 2003.

[72] F. Lyardet, E. Aitenbichler, G. Austaller, J. Kangasharju, and M. Mühlhäuser. Lessons learned in smart environments engineering. In *UbiComp 2007 Workshop Proceedings*, pages 325–330, September 2007.

[73] Kevin Lynch. *The image of the city.* MIT Press, Cambridge, Massachusettes, 1960.

[74] Kalle Lyytinen and Youngjin Yoo. Issues and challenges in ubiquitous computing. *Communincations of the ACM*, 45(12):62–65, December 2002.

[75] K. McGlinn, E. O'Neill, A. Gibney, D. O'Sullivan, and D. Lewis. SimCon: A tool to support rapid evaluation of smart building application design using context simulation and virtual reality. *Journal of Universal Computer Science*, 16(15):1992–2018, 2010.

[76] Robin Milner. Ubiquitous computing: Shall we understand it? *The Computer Journal*, 49(4):383–389, June 2006.

[77] Michihiko Minoh and Tatsuya Yamazaki. Daily life support experiment at ubiquitous computing home. *The 11th Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU2006)*, 2006.

[78] R. Morla and N. Davies. Evaluating a location-based application: A hybrid test and simulation environment. *IEEE Pervasive Computing*, 3(3):48–56, 2004.

[79] I. Nieto, J.A. Bota, and A.F. Gmez-Skarmeta. Information and hybrid architecture model of the OCP contextual information management system. *Journal of Universal Computer Science*, 12(3):357–366, 2006.

[80] Hiroshi Nishikawa, Shinya Yamamoto, Morihiko Tamai, Kouji Nishigaki, Tomoya Kitani, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. UbiREAL: Realistic smartspace simulator for systematic testing. In *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, pages 459–476. Springer Berlin / Heidelberg, 2006.

[81] Dan O'Neill. Integration engineering perspective. *Journal of Systems and Software*, 3(1):77–83, March 1983.

[82] Eleanor O'Neill. TATUS: A ubiquitous computing simulator. M.Sc. dissertation, Trinity College Dublin, 2004.

[83] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, and Dirk Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM '05)*, pages 60–69. IEEE Computer Society, 2005.

[84] A. Padovitz, S.W. Loke, and A. Zaslavsky. Towards a theory of context spaces. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 38 – 42, March 2004.

[85] Julius Panero and Martin Zelnik. *Human Dimension and Interior Space: A Source Book of Design Reference Standards*. Watson-Guptill, 1979.

[86] Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers (ISWC '98)*. IEEE Computer Society, 1998.

[87] Shwetak Patel, Jun Rekimoto, and Gregory Abowd. iCam: Precise at-a-distance interaction in the physical environment. In *Pervasive Computing*, volume 3968 of *Lecture Notes in Computer Science*, pages 272–287. Springer Berlin / Heidelberg, 2006.

[88] Thomas Pederson and Dipak Surie. Towards an activity-aware wearable computing platform based on an egocentric interaction model. In *Ubiquitous Computing Systems*, volume 4836 of *Lecture Notes in Computer Science*, pages 211–227. Springer Berlin / Heidelberg, 2010.

[89] C. Potts, K. Takahashi, and A. I. Anton. Inquiry-based requirements analysis. *IEEE Software*, 11(2):21–32, 1994.

[90] Colin Potts. Using schematic scenarios to understand user needs. In *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques (DIS '95)*, pages 247–256. ACM New York, NY, USA, 1995.

[91] Karl Quinn. *A Multi-Faceted Model of Trust that is Personalisable and Specialisable*. Ph.D. thesis, Trinity College Dublin, 2007.

[92] Andry Rakotonirainy and Greg Groves. Resource discovery for pervasive environments. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 866–883. Springer Berlin / Heidelberg, 2010.

[93] Daniel Ronzani. The battle of concepts: Ubiquitous computing, pervasive computing and ambient intelligence in mass media. *Ubiquitous Computing and Communication Journal*, 4(2), 2009.

[94] D. Salber, A. K. Dey, and G. D. Abowd. 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 (CHI '99)*, pages 434–441. ACM, New York, 1999.

[95] Ichiro Satoh. A location model for smart environments. *Pervasive and Mobile Computin*, 3(2):158–179, March 2007.

[96] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[97] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pages 85–90. IEEE Computer Society, 1994.

[98] Bill N. Schilit, Marvin M. Theimer, and Brent B. Welch. Customizing mobile applications. In *Proceedings of the USENIX Symposium on Mobile & Location-Independent Computing*, pages 129–138, 1993.

[99] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22 –32, 1994.

[100] Albrecht Schmidt, Kofi Asante Adoo, Antti Takaluoma, Urop Tuomela, Kristof Van Laerhoven, and Walter Van De Velde. Advanced interaction in context. In *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 89–101. Springer Berlin / Heidelberg, 1999.

[101] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893 – 901, 1999.

[102] J. Scourias and T. Kunz. Activity-based mobility modeling: realistic evaluation of location management schemes for cellular networks. In *IEEE Wireless Communications and Networking Conference (WCNC '99)*, volume 1, pages 296 –300, 1999.

[103] Emilio Serrano and Juan Botia. Infrastructure for forensic analysis of multi-agent systems. In *Programming Multi-Agent Systems*, volume 5442 of *Lecture Notes in Computer Science*, pages 168–183. Springer Berlin / Heidelberg, 2009.

[104] Emilio Serrano, Juan Botia, and Jose Cadenas. Construction and debugging of a multi-agent based simulation to study ambient intelligence applications. In *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *Lecture Notes in Computer Science*, pages 1090–1097. Springer Berlin / Heidelberg, 2009.

[105] Emilio Serrano, Juan A. Botia, and Jose M. Cadenas. Ubik: a multi-agent based simulator for ubiquitous computing applications. *Journal of Physical Agents*, vol. 3, no. 2:39–43, 2009.

[106] Wassiou Sitou and Bernd Spanfelner. Towards requirements engineering for context adaptive systems. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02*, COMPSAC '07, pages 593–600, Washington, DC, USA, 2007. IEEE Computer Society.

[107] Morris Sloman, Martyn Thomas, Karen Sparck-Jones, Jon Crowcroft, Marta Kwiatkowska, Paul Garner, Nicholas R. Jennings, Vladimiro Sassone, Eamonn O'Neill, Michael Woodridge, Carsten Maple, George Coulouris, and Dan Chalmers. Discussion on robin milner's first computer journal lecture: Ubiquitous computing: Shall we understand it? *The Computer Journal*, 49(4):390–399, 2006.

[108] Shamus P. Smith and David Trenholme. Rapid prototyping a virtual fire

drill environment using computer game technology. *Fire Safety Journal*, 44(4):559–569, 2009.

[109] Moonbae Song, Kwangjin Park, and Jehyok Ryu. Modeling and tracking complexly moving objects in location-based services. *Journal of Information Science and Engineering*, 20(3):517–534, 2004.

[110] Vincent Tabak and Bauke de Vries. Methods for the prediction of intermediate activities by office occupants. *Building and Environment*, 45(6):1366 – 1372, 2010.

[111] Lei Tang, Zhiwen Yu, Xingshe Zhou, Hanbo Wang, and Christian Becker. Supporting rapid design and evaluation of pervasive applications: challenges and solutions. *Personal and Ubiquitous Computing*, pages 1–17, 2010.

[112] David Tennenhouse. Proactive computing. *Communications of the ACM*, 43(5):43–50, 2000.

[113] David Trenholme and Shamus Smith. Computer game engines for developing first-person virtual environments. *Virtual Reality*, 12:181–187, 2008.

[114] Hung-Ying Tyan, Ahmed Sobeih, and Jennifer C. Hou. Design, realization and evaluation of a component-based, compositional network simulation environment. *Simulation*, 85(3):159–181, 2009.

[115] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[116] Stefan G. Weber, Andreas Heinemann, and Max Mühlhäuser. Towards an architecture for balancing privacy and traceability in ubiquitous computing environments. In *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 958–964, Washington, DC, USA, 2008. IEEE Computer Society.

[117] Torben Weis, Mirko Knoll, Andreas Ulbrich, Gero Muhl, and Alexander Brandle. Rapid prototyping for pervasive applications. *IEEE Pervasive Computing*, 6(2):76–84, 2007.

[118] Mark Weiser. Creating the invisible interface: (invited talk). In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, New York, NY, USA, 1994. ACM.

[119] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, 1999.

[120] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[121] Tatsuya Yamazaki. Ubiquitous home: real-life testbed for home context-aware service. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 54–59. IEEE Computer Society, 2005.

[122] K.F. Yeung, Yanyan Yang, and D. Ndzi. Contextualized mobile information retrieval in hybrid P2P environment. In *Joint Conferences on Pervasive Computing (JCPC)*, pages 543 –546, 2009.

Appendix A

Lighting Case Study

A.1 Models

This appendix presents the models used to define the test specification for the lighting case study in Chapter 6, Section 6.2.2.

A.1.1 Situation Model

Listing A.1: User present and lights are off.

```
rule "Lights Off for User"
  when
    $user : UserStateFact ($userLocationRoom:userLocationRoom)
    not (exists ( EntityStateFact (entityType == "light",
      entityLocationRoom == $userLocationRoom, entityState == "on") ) )
  then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
      "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model, "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "timestamp", new Long (System.currentTimeMillis()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model,
      "location", $light.getEntityLocation());
    modelManipulation.addNode(SARConstants.sar_Model,
      "user", $user.getUserID());
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
      "SAR_output_model.xml");
end
```

Listing A.2: Lights are on when no user is present.

```
rule "Lights On for No User"
  when
    $light : EntityStateFact (entityType == "light",
      entityState == "on", $lightLocationRoom:entityLocationRoom)
    not (exists (UserStateFact (userLocationRoom == $lightLocationRoom)))
  then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
      "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model, "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "timestamp", new Long (System.currentTimeMillis()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model,
      "location", $light.getEntityLocation());
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
                              "SAR_output_model.xml");

end
```

A.1.2 Spatial Model

Listing A.3: Spatial Model.

```
<?xml version="1.0" encoding="UTF-8"?>
<esrm>
  <building id="TCD-Building">
    <vertical-space id="atrium" type="atrium" />
    <vertical-space id="fs-stairwell-south" type="stairwell">
      <entity id="light-stairwell-south"
        type="light" initial-state="off"/>
      <area id="fs-stairwell-south-1" type="stairwell-exit">
        <entity id="ms-stairwell-south-1"
          type="kproxy_motion_sensor" initial-state="active"/>
      </area>
      <area id="fs-stairwell-south-0" type="stairwell-exit">
        <entity id="ms-stairwell-south-0"
          type="kproxy_motion_sensor" initial-state="active"/>
      </area>
    </vertical-space>
    <vertical-space id="fs-stairwell-north" type="stairwell">
      <entity id="light-stairwell-north"
        type="light" initial-state="off"/>
      <area id="fs-stairwell-north-0" type="stairwell-exit">
        <entity id="ms-stairwell-north-0"
          type="kproxy_motion_sensor" initial-state="active"/>
      </area>
      <area id="fs-stairwell-south-1" type="stairwell-exit">
        <entity id="ms-stairwell-north-1"
```

```xml
          type="kproxy_motion_sensor" initial-state="active"/>
      </area>
    </vertical-space>
    <floor id="0">
      <room id="fs-lift-lobby-south-0" type="lift-lobby">
        <entity id="light-lift-lobby-south-0"
          type="light" initial-state="off"/>
        <entity id="ms-lift-lobby-south-0"
          type="kproxy_motion_sensor" initial-state="active"/>
      </room>
      <room id="fs-corridor-south-0" type="corridor">
        <entity id="light-corridor-south-0"
          type="light" initial-state="off"/>
      </room>
      <room id="fs-lift-lobby-north-0" type="lift-lobby">
        <entity id="light-lift-lobby-north-0"
          type="light" initial-state="off"/>
        <entity id="ms-lift-lobby-north-0"
          type="kproxy_motion_sensor" initial-state="active"/>
      </room>
      <room id="fs-corridor-north-0" type="corridor">
        <entity id="light-corridor-north-0"
          type="light" initial-state="off"/>
      </room>
    </floor>
    <floor id="1">
      <room id="fs-lift-lobby-south-1" type="lift-lobby">
        <entity id="light-lift-lobby-south-1"
          type="light" initial-state="off"/>
        <entity id="ms-lift-lobby-south-1"
          type="kproxy_motion_sensor" initial-state="active"/>
      </room>
      <room id="fs-lift-lobby-north-1" type="lift-lobby">
        <entity id="light-lift-lobby-north-1"
          type="light" initial-state="off"/>
        <entity id="ms-lift-lobby-north-1"
          type="kproxy_motion_sensor" initial-state="active"/>
      </room>
      <room id="fs-corridor-1" type="corridor">
        <entity id="light-corridor-1"
          type="light" initial-state="off"/>
      </room>
    </floor>
  </building>
</esrm>
```

A.2 Results

This section presents the full set of charted results, as captured in the Alert Report and test log files.

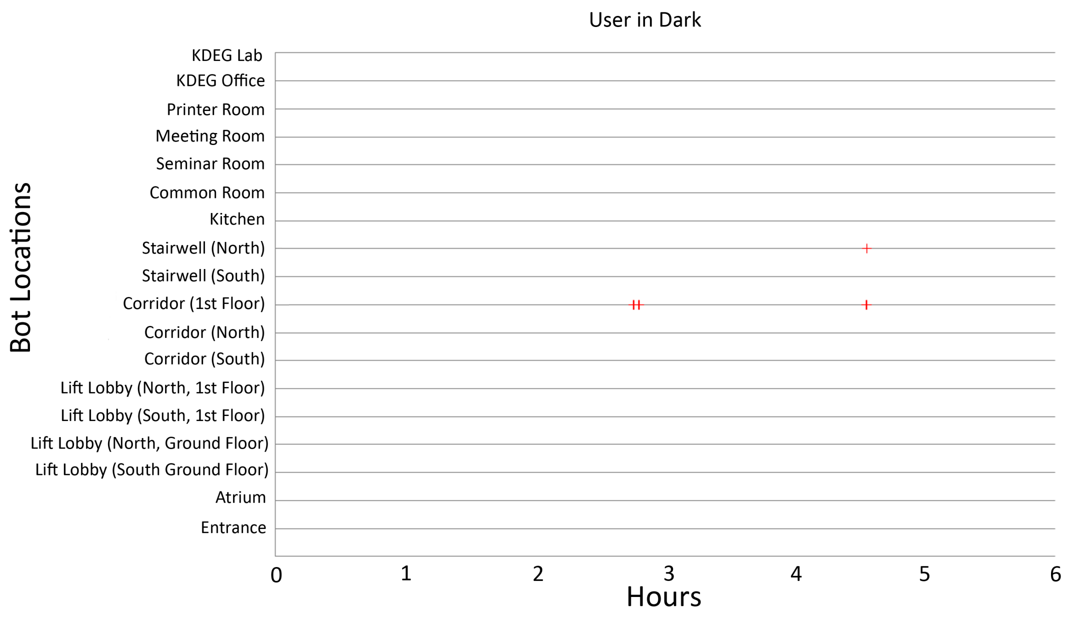


Figure A.1: Situation A.1: Bot in the dark (Hours 0-6).

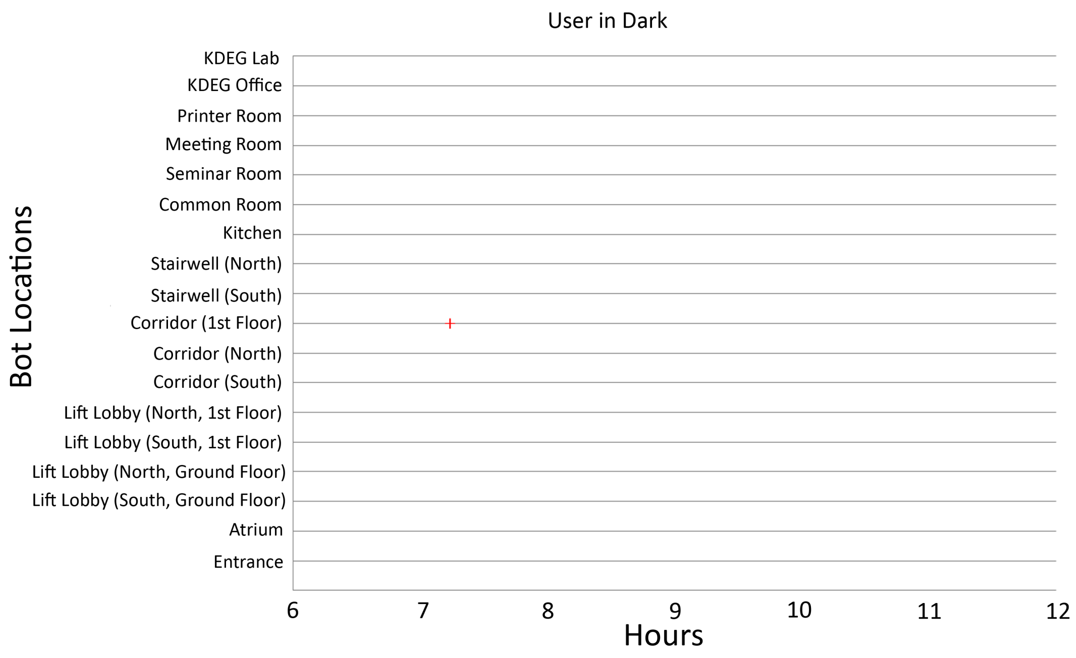


Figure A.2: Situation A.1: Bot in the dark (Hours 6-12).

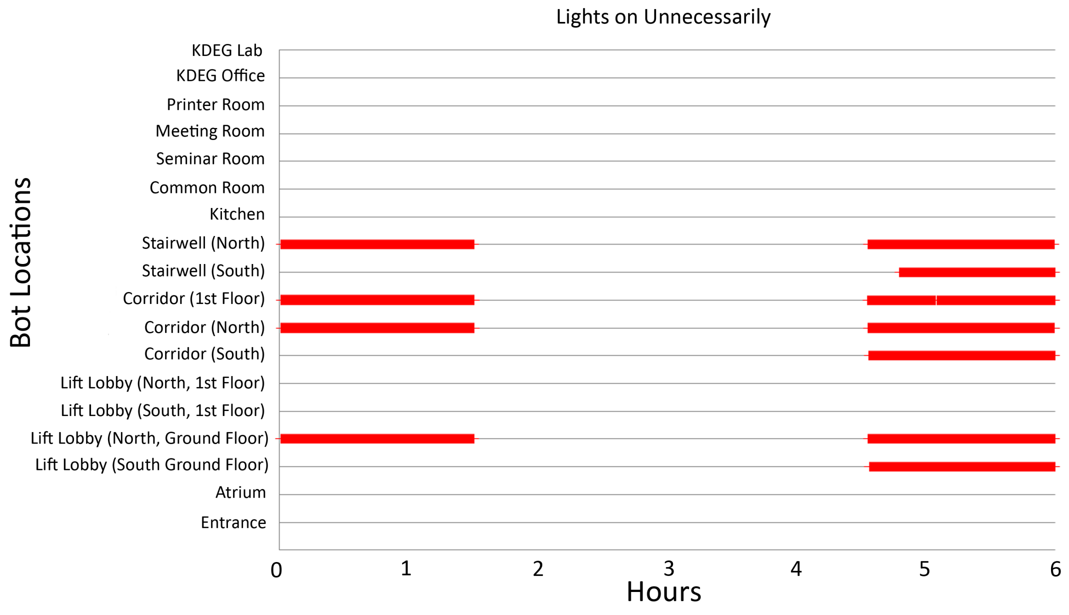


Figure A.3: Situation A.2: Lights on unnecessarily (Hours 0-6).

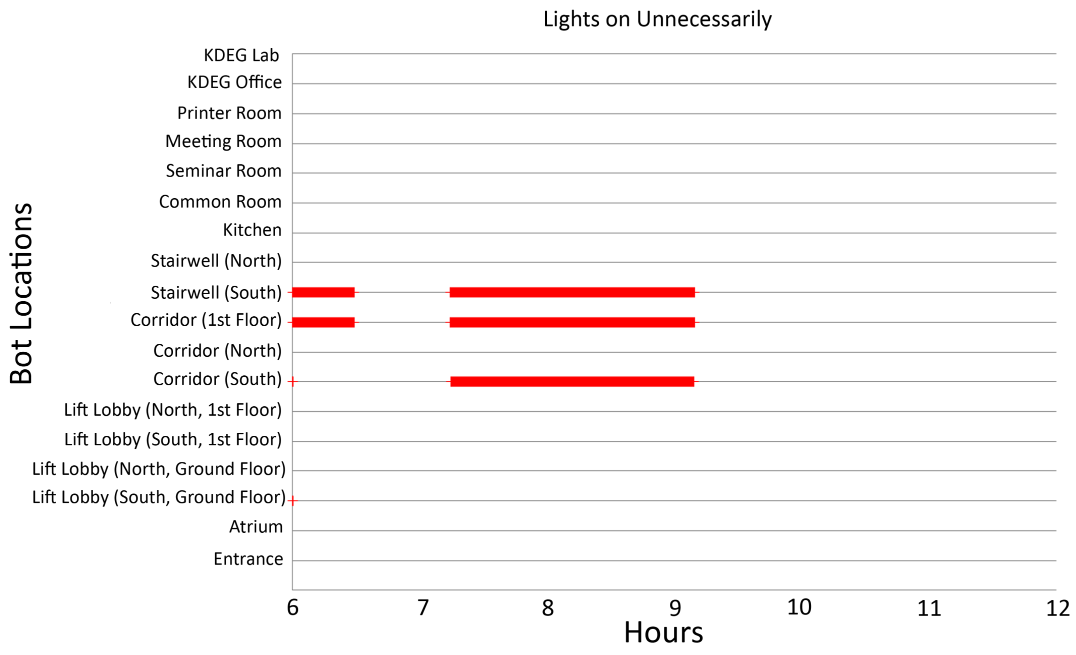


Figure A.4: Situation A.2: Lights on unnecessarily (Hours 6-12).

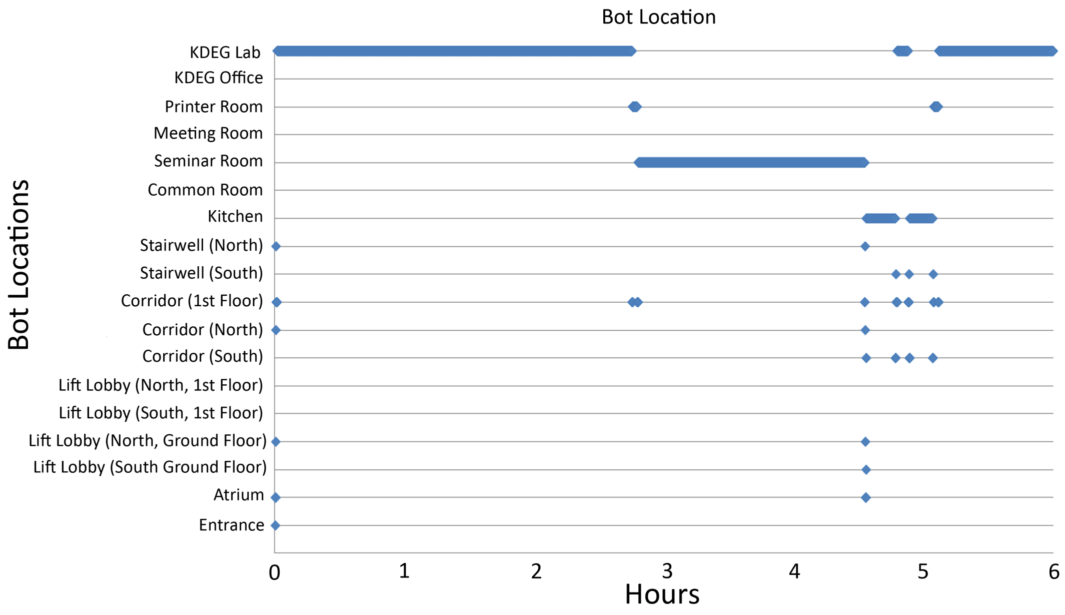


Figure A.5: Trace of bot location over time (Hours 0-6).

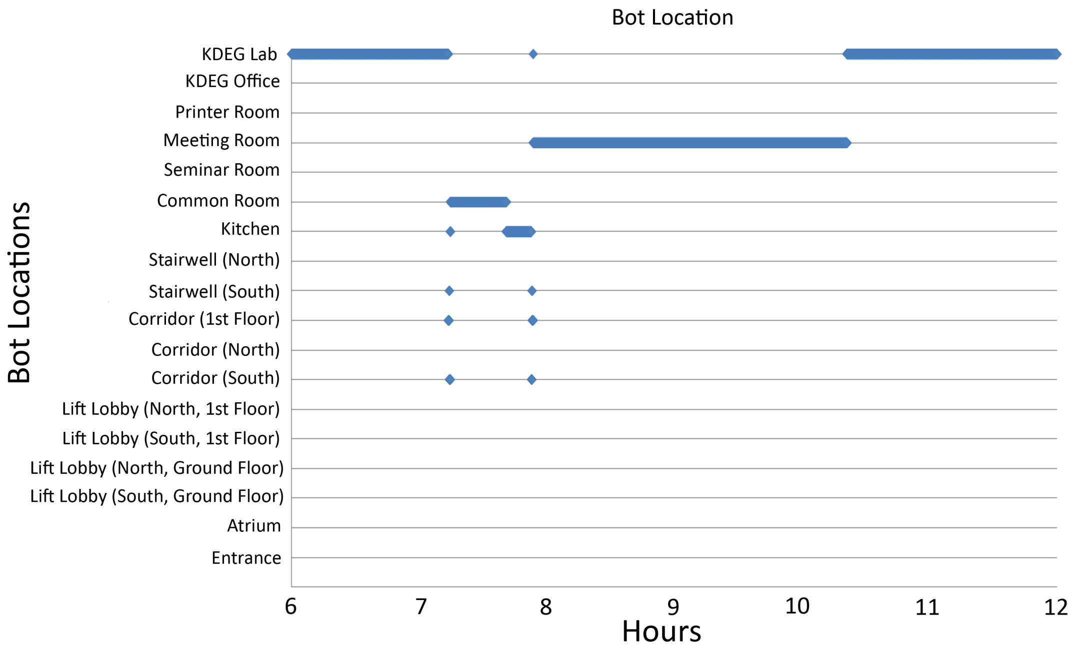


Figure A.6: Trace of bot location over time (Hours 6-12).

Appendix B

Access Control Case Study

B.1 Models

This appendix presents the test specifications and models for the access control case study in Chapter 6, along with some sample results. The full results set are included in Appendix I on the CD accompanying this thesis.

B.1.1 Activity Model

Occupants				
Activity ID	**Location**	**Min Duration**	**Max Duration**	**Next Activity**
Working-In-Office	All Desk Areas	1 minute	4 hours	Lunch-Break, Coffee-Break, Attend-Meeting, Short-Break
Attend-Meeting	Meeting Room	15 minutes	3 hours	Working-In-Office, Lunch-Break, Coffee-Break, Short-Break
Lunch-Break	Common Room	5 minutes	2 hours	Working-In-Office, Attend-Meeting
Coffee-Break	Coffee Room	2 minutes	15 minutes	Working-In-Office, Attend-Meeting
Short-Break	Outside	5 seconds	5 minutes	Working-In-Office, Attend-Meeting,
Activity ID	**Location**	**Min Global Time**	**Max Global Time**	**Next Activity**
Start-Day	Outside	06:00	10:30	Working-In-Office, Attend-Meeting
End-Day	Outside	17:00	04:30	
Visitors				
Activity ID	**Location**	**Min Duration**	**Max Duration**	**Next Activity**
Visiting	All Offices, Coffee Room	15 minutes	60 minutes	Not-Visiting, Visit-Coffee-Room
Not-Visiting	Outside	5 seconds	15 minutes	Visit-Coffee-Room Visit-Offices

Figure B.1: Activity definitions for Access Control Case Study.

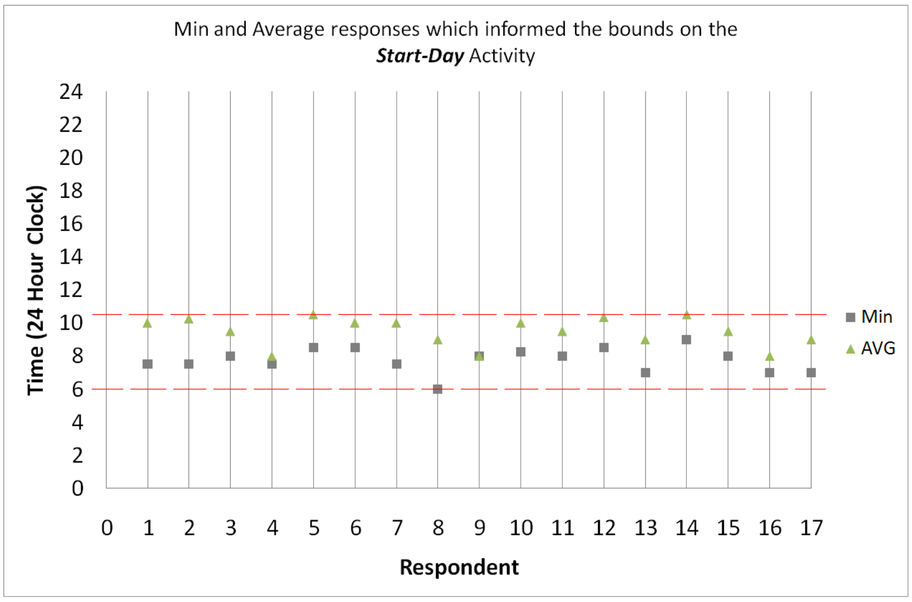


Figure B.2: Times at which respondents reported starting their working day.

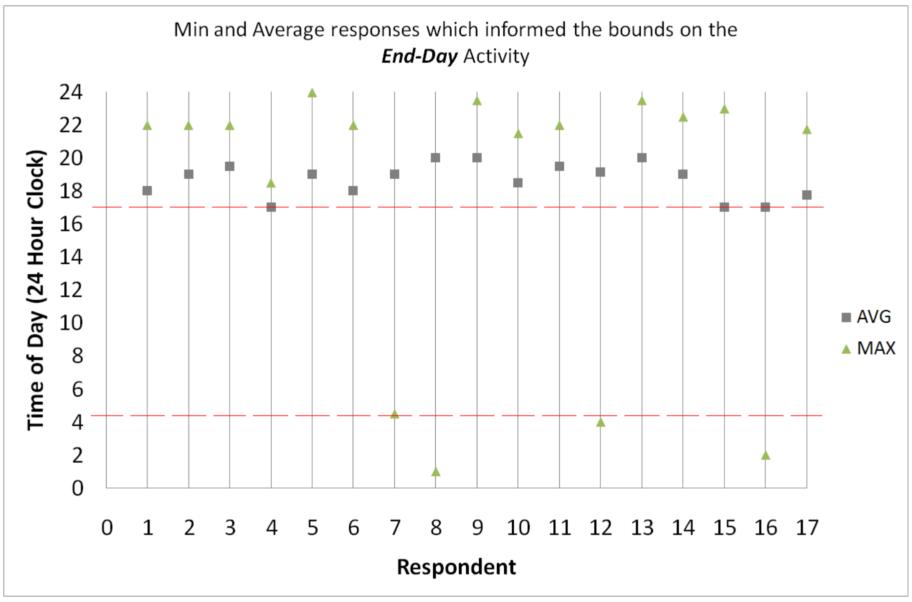


Figure B.3: Times at which respondents reported ending their working day.

B.1.2 Situation Model

Listing B.1: A visitor is present without an occupant.

```
rule "Visitor present without an occupant"
  when
    $visitor : UserStateFact ( userType == "visitor",
      $visitorLocationRoom:userLocationRoom )
    LocationStateFact ( locationID == $visitorLocationRoom,
      locationType == "office",
      || locationType == "coffee-room" )
    not (exists UserStateFact ( userType == "staff"
      || userType == "student",
      userLocationRoom == $visitorLocationRoom ))
  then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
        "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model, "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
        "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model,
        "timestamp", new Long ($user.getUpdateTime()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model, "id",
        drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model,
        "location", $visitorLocationRoom);
    modelManipulation.addNode(SARConstants.sar_Model,
        "user", $visitor.getUserID() );
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
        "SAR_eon_model1.xml");
end
```

Listing B.2: A visitor is unobserved when in the presence of occupants.

```
rule "Visitor unseen by occupants in the same room."
  when
        $visitor : UserStateFact ( userType == "visitor",
                    $visitorLocationRoom:userLocationRoom,
                    $visitorID:userID )
        LocationStateFact ( locationID == $visitorLocationRoom,
                    locationType == "office" ||
                    locationType == "coffee-room")
      exists ( UserStateFact ( userType == "staff" || userType == "student",
                  userLocationRoom == $visitorLocationRoom ) )
        forall ($occupant: UserStateFact (
                  userType == "staff" || userType == "student",
                  userLocationRoom == $visitorLocationRoom)
                  UserStateFact (this == $occupant,
                  usersInView not contains $visitorID ))
    then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
      "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model, "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model, "timestamp",
      new Long ($visitor.getUpdateTime()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model, "location",
      $visitorLocationRoom);
    modelManipulation.addNode(SARConstants.sar_Model, "user",
      $visitorID);
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
      "SAR_eon_model1.xml");
```

```
end
```

Listing B.3: Office space is unlocked and no occupant is present.

```
rule "Office space unlocked and no occupant present"
  when
    $location : LocationStateFact ( locationTag=="space",
                locationType=="officespace",
                $locationID:locationID )
    (and (not (exists UserStateFact (userType != "visitor",
                userLocationSpace == $locationID)))
      (exists EntityStateFact (entityType == "external-door",
                entityState != "locked")))
  then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
      "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model,
      "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model, "timestamp",
      new Long ($location.getUpdateTime()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model,
      "location", $locationID);
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
      "SAR_eon_model1.xml");
end
```

Listing B.4: User waiting to enter space.

```
rule "User waiting to enter space"
  when
    $user : UserStateFact (userType == "visitor",
        $userLocationArea : userLocationArea matches "wait.*" )
  then
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelManipulation.searchModel(SARConstants.sar_Model,
      "system-alerts");
    modelManipulation.addNode(SARConstants.sar_Model,
      "alert");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", new Integer (SarGenerator.getNextAlertID()).toString());
    modelManipulation.addAttribute(SARConstants.sar_Model, "timestamp",
      new Long ($user.getUpdateTime()).toString());
    modelManipulation.addNode(SARConstants.sar_Model, "rule");
    modelManipulation.addAttribute(SARConstants.sar_Model,
      "id", drools.getRule().getName());
    modelManipulation.navigateModel(SARConstants.sar_Model, "..");
    modelManipulation.addNode(SARConstants.sar_Model,
      "user", $user.getUserID());
    modelManipulation.addNode(SARConstants.sar_Model,
      "location", $userLocationArea);
    modelManipulation.navigateModel(SARConstants.sar_Model, "/");
    modelControl.storeModel("data", SARConstants.sar_Model,
      "SAR_eon_model1.xml");
end
```

B.1.3 Spatial Model

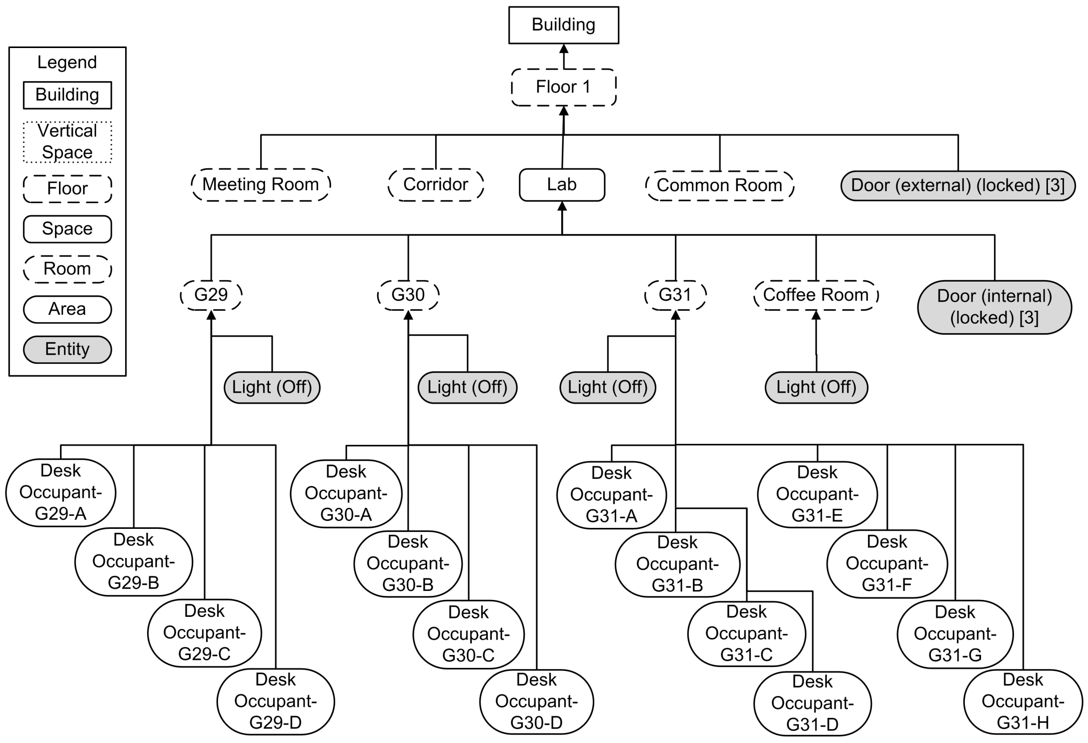


Figure B.4: Hierarchy of Zones and Entities.

Listing B.5: Spatial Model for Access Control Case Study.

```

<?xml version="1.0" encoding="UTF-8"?>
<esrm>
  <building id="OReilly" type="office-building">
    <floor id="floor2" type="floor">
      <room id="OReillyCommonRoom" type="coffee-room"/>
    </floor>
    <floor id="floor1" type="floor">
      <door id="door-Corridor-G31" type="external-door"
        initial-state="locked" entry="Corridor"
        exit="G31"/>
      <door id="door-Corridor-CoffeeRoom" type="external-door"
        initial-state="locked" entry="Corridor"
        exit="CoffeeRoom"/>
      <door id="door-Corridor-G29" type="external-door"
        initial-state="locked" entry="Corridor"
        exit="G29"/>
      <door id="door-Corridor-MeetingRoom" type="door"
        initial-state="unlocked" entry="Corridor"
        exit="MeetingRoom"/>
      <room id="Lobby" type="lobby"/>
        <space id="mainkdeglab" type="officespace">
          <door id="door-G31-G30" type="internal-door"
            initial-state="unlocked" entry="G31"
            exit="G30"/>
          <door id="door-CoffeeRoom-G30" type="internal-door"
            initial-state="unlocked" entry="CoffeeRoom"
            exit="G30"/>
          <door id="door-G29-G30" type="internal-door"
            initial-state="unlocked" entry="G29"
            exit="G30"/>
          <room id="G31" type="office">
            <entity id="rfid_G31toCorridor" type="kproxy_rfid_reader"
              mobile="no" initial-state="active" />
            <entity id="light-G31" type="light" mobile="no"
              initial-state="off" />
            <entity id="proximity_npc1" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc2" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc3" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc4" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc5" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc6" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc7" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc8" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="ms_G31" type="kproxy_motion_sensor"
              mobile="no" initial-state="active"/>
            <area id="desk_npc1" type="desk"/>
            <area id="desk_npc2" type="desk"/>
            <area id="desk_npc8" type="desk"/>
            <area id="desk_npc7" type="desk"/>
            <area id="desk_npc4" type="desk"/>
            <area id="desk_npc3" type="desk"/>
            <area id="desk_npc6" type="desk"/>
            <area id="desk_npc5" type="desk"/>
            <area id="waitareaG31toCorridor_visitor" type="waitarea"/>
            <area id="waitareaG31toG30_visitor" type="waitarea"/>
          </room>
          <room id="G30" type="office">
            <entity id="light-G30" type="light" mobile="no"
              initial-state="off" />
            <entity id="proximity_npc9" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="proximity_npc10" type="kproxy_proximity"
              mobile="no" initial-state="active" />
```

```
      <entity id="proximity_npc11" type="kproxy_proximity"
        mobile="no" initial-state="active" />
      <entity id="proximity_npc12" type="kproxy_proximity"
        mobile="no" initial-state="active" />
      <area id="desk_npc10" type="desk"/>
      <area id="desk_npc11" type="desk"/>
      <area id="desk_npc9" type="desk"/>
      <area id="desk_npc12" type="desk"/>
      <area id="waitareaG30toG29_visitor" type="waitarea"/>
      <area id="waitareaG30toCoffee_visitor" type="waitarea"/>
      <area id="waitareaG30toG31_visitor" type="waitarea"/>
      <entity id="ms_G30" type="kproxy_motion_sensor"
        mobile="no" initial-state="active"/>
    </room>
    <room id="CoffeeRoom" type="coffee-room">
      <entity id="light-CoffeeRoom" type="light"
        mobile="no" initial-state="off" />
      <entity id="rfid_CoffeetoCorridor" type="kproxy_rfid_reader"
        mobile="no" initial-state="active"/>
      <entity id="ms_Coffee" type="kproxy_motion_sensor"
        mobile="no" initial-state="active"/>
      <area id="waitareaCoffeetoCorridor_visitor"
        type="waitarea"/>
      <area id="waitareaCoffeetoG30_visitor"
        type="waitarea"/>
    </room>
    <room id="G29" type="office">
      <entity id="rfid_G29toCorridor" type="kproxy_rfid_reader"
        mobile="no" initial-state="active"/>
      <entity id="light-G31" type="light" mobile="no"
        initial-state="off" />
      <entity id="proximity_npc13" type="kproxy_proximity"
        mobile="no" initial-state="active" />
      <entity id="proximity_npc14" type="kproxy_proximity"
        mobile="no" initial-state="active" />
      <entity id="proximity_npc15" type="kproxy_proximity"
        mobile="no" initial-state="active" />
```

```
            <entity id="proximity_npc16" type="kproxy_proximity"
              mobile="no" initial-state="active" />
            <entity id="ms_G29" type="kproxy_motion_sensor"
              mobile="no" initial-state="active"/>
            <area id="desk_npc13" type="desk"/>
            <area id="desk_npc15" type="desk"/>
            <area id="desk_npc14" type="desk"/>
            <area id="desk_npc16" type="desk"/>
            <area id="waitareaG29toG30_visitor" type="waitarea"/>
            <area id="waitareaG29toCorridor_visitor" type="waitarea"/>
          </room>
        </space>
        <room id="Corridor" type="corridor">
          <entity id="rfid_CorridortoG31" type="kproxy_rfid_reader"
            mobile="no" initial-state="active" />
          <entity id="rfid_CorridortoG29" type="kproxy_rfid_reader"
            mobile="no" initial-state="active" />
          <entity id="rfid_CorridortoCoffee" type="kproxy_rfid_reader"
            mobile="no" initial-state="active" />
          <area id="waitareaCorridortoG29_visitor" type="waitarea"/>
          <area id="waitareaCorridortoG31_visitor" type="waitarea"/>
          <area id="waitareaCorridortoCoffee_visitor" type="waitarea"/>
        </room>
        <room id="MeetingRoom" type="meeting-room"/>
      </floor>
  </building>
</esrm>
```

B.2 Sample Situation Traces

B.2.1 Detected Situation: Visitor present without an occupant

Figures B.5, B.6, B.7 and B.8 illustrate a simple example of instances of the situation *Visitor present without an occupant*, which checks for times that a visitor is present in an office without an occupant.

<u>Situation</u>: *Visitor present without an occupant.*

if

U(type = "visitor") $\longleftarrow$ v

$\wedge$ L((zoneID = v.roomLocation) $\wedge$ ((type = "office") $\vee$ (type = "coffee-room")))

$\wedge$ $\nexists$ (U(((type = "staff") $\vee$ (type = "student"))

$\wedge$ (roomLocation = v.roomLocation))

then

$Output\ O(t) \subset G(t)$

Figure B.5 illustrates all of the recorded instances of this situation and the locations in which they were detected. Figure B.6 zooms in on two instances in particular in order to discuss the surrounding relationships. Figure B.7 confirms that the visitor was present in these locations at the time the situations were detected. This chart also confirms that the visitor entered through the door between the corridor and G29. Figure B.8 shows that Occupant-G30-B opened that door, around the 0.2 hour mark on the time-line, shortly before the visitor entered the space, confirming that the access control system allowed the visitor to enter the space.

For the duration that this situation was recorded, Occupant-G30-B was in G30 while the visitor was in G29, confirming that the visitor was unsupervised. It can also be observed that the situation is no longer detected once Occupant-G29-C arrives at G29 around the 0.45 hour mark on the time-line, in Figure B.7.

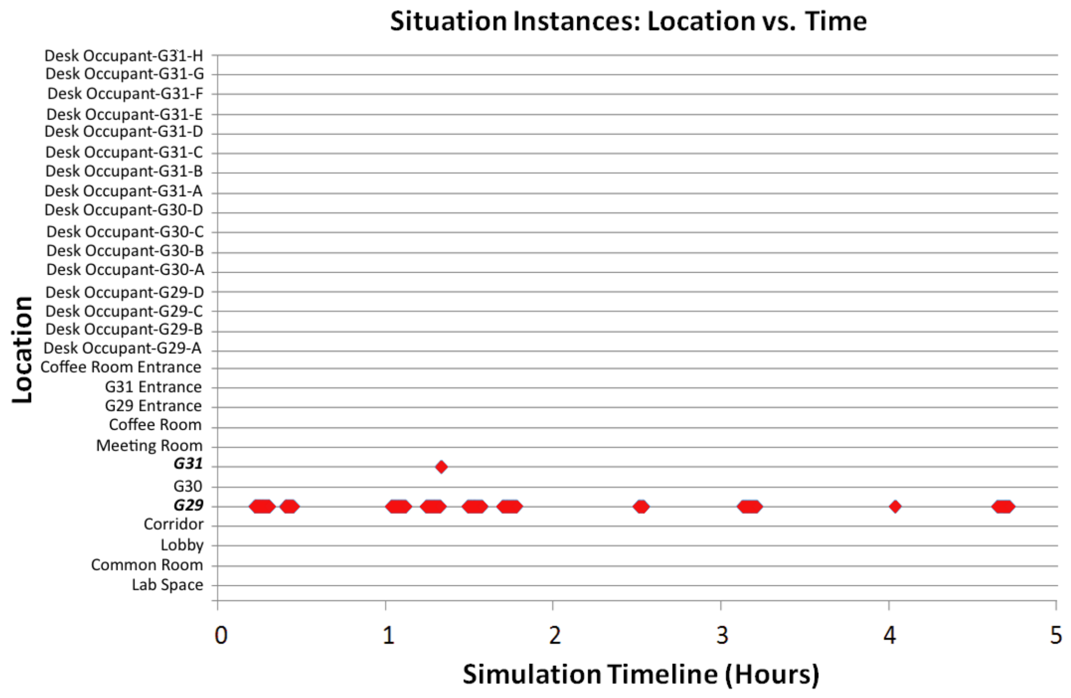


Figure B.5: Detected situational instances.

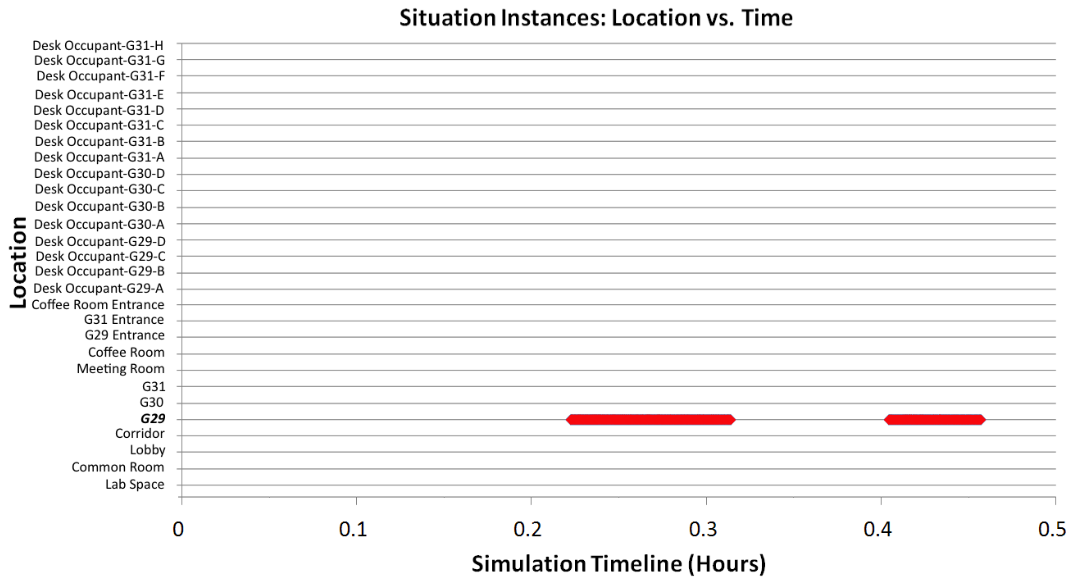


Figure B.6: Zoomed view of timeline.

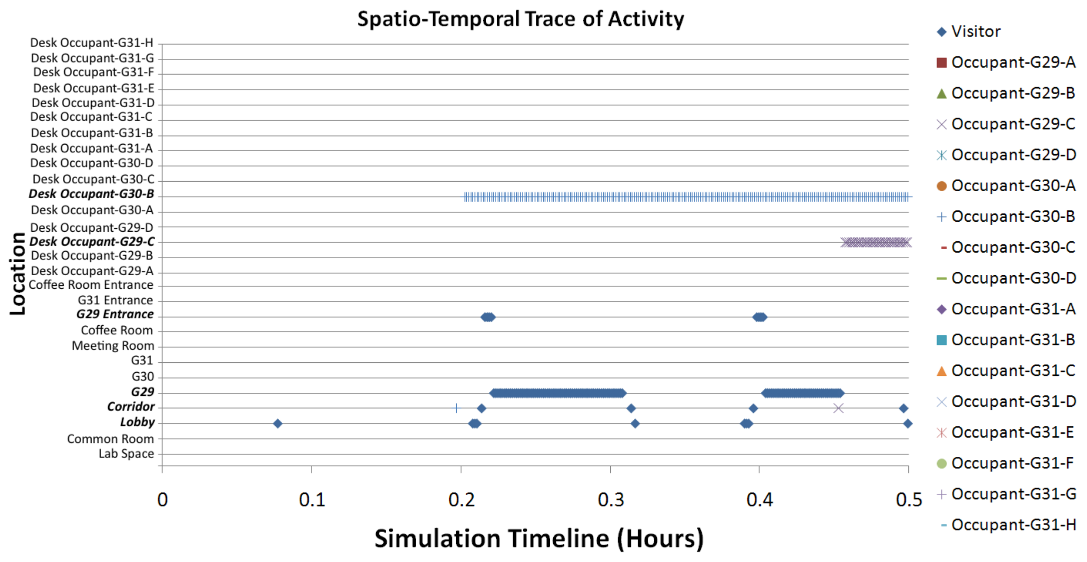


Figure B.7: Spatio-temporal trace of activity.

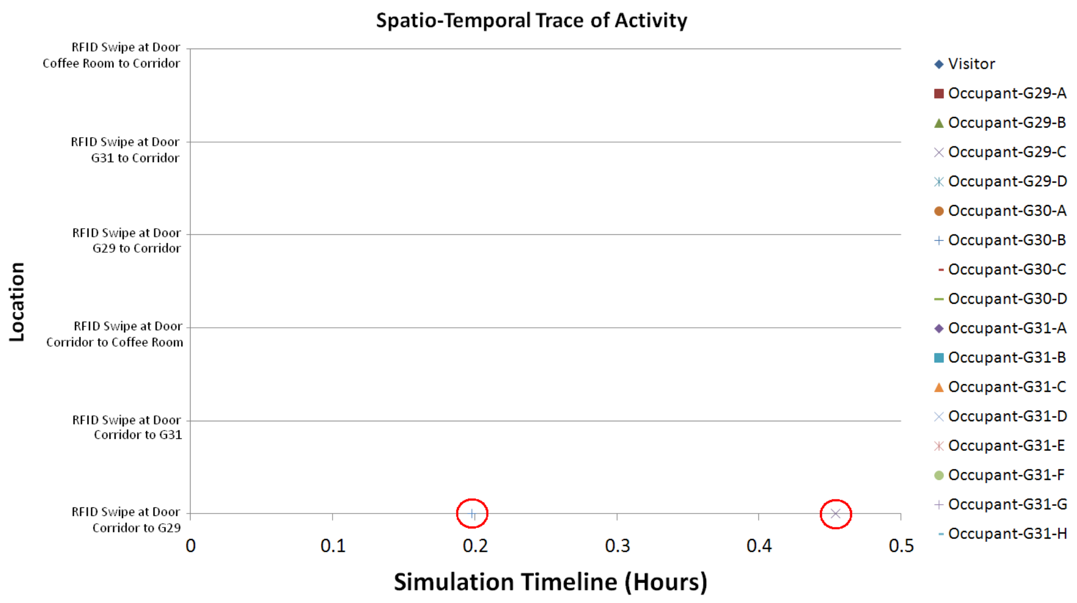


Figure B.8: Log of RFID scans at readers.

B.2.2 Detected Overlapping Situations: Visitor present without and/or unobserved by occupants

The situation specifications, defined below, provide examples of overlapping situations, i.e. *(A) Visitor present without an occupant* and *(B) Visitor unobserved by occupants.* Figure B.9 shows these situations occurring in parallel until some event changes the state of the environment so that only situation B remains.

Figure B.10 shows that these situational instances occurred in room G29. The hierarchy of locations, illustrated in Figure B.4, shows that room G29 contains four desk areas. These mark the zones for desks normally occupied by Occupant-G29-A, Occupant-G29-B, Occupant-G29-C and Occupant-G29-D.

Figure B.11 has been refined to focus on the occupants of G29 and the visitor bot for a clearer trace of the spatio-temporal relationships in these five zones, i.e. G29 and the areas that it contains. This trace of activity in G29 shows that during the third situational instance, which starts shortly after the 0.9 hour mark on the time-line, that although Occupant-G29-B is present at their desk in G29, the occupant is oriented away from the rest of the room, resulting in the visitor being unobserved but not alone.

Situation (A): *Visitor present without an occupant.*
if
U(type = "visitor") $\longleftarrow$ v
$\wedge$ L((zoneID = v.roomLocation) $\wedge$ ((type = "office") $\vee$ (type = "coffee-room")))
$\wedge$ $\nexists$ (U(((type = "staff") $\vee$ (type = "student")) $\wedge$ (roomLocation = v.roomLocation))
then
Output $O(t) \subset G(t)$

Situation (B): *Visitor unobserved by occupants.*

if

U (type = "visitor") $\longleftarrow$ v

$\wedge$ L ((zoneID = v.roomLocation) $\wedge$ (type = "office" $\vee$ type = "coffee-room"))

$\wedge$ $\forall$ (U ((type = "staff" $\vee$ type = "student")

$\wedge$ (roomLocation = v.roomLocation)) :

(v $\in$ FOV))

then

Output $O(t) \subset G(t)$

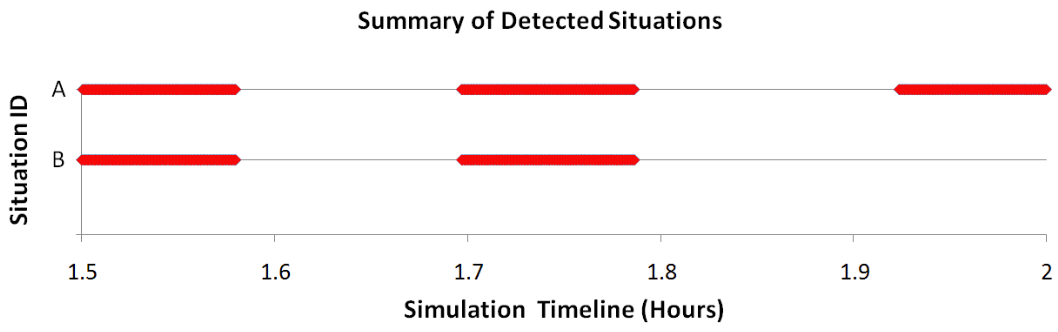


Figure B.9: Detected situational instances.

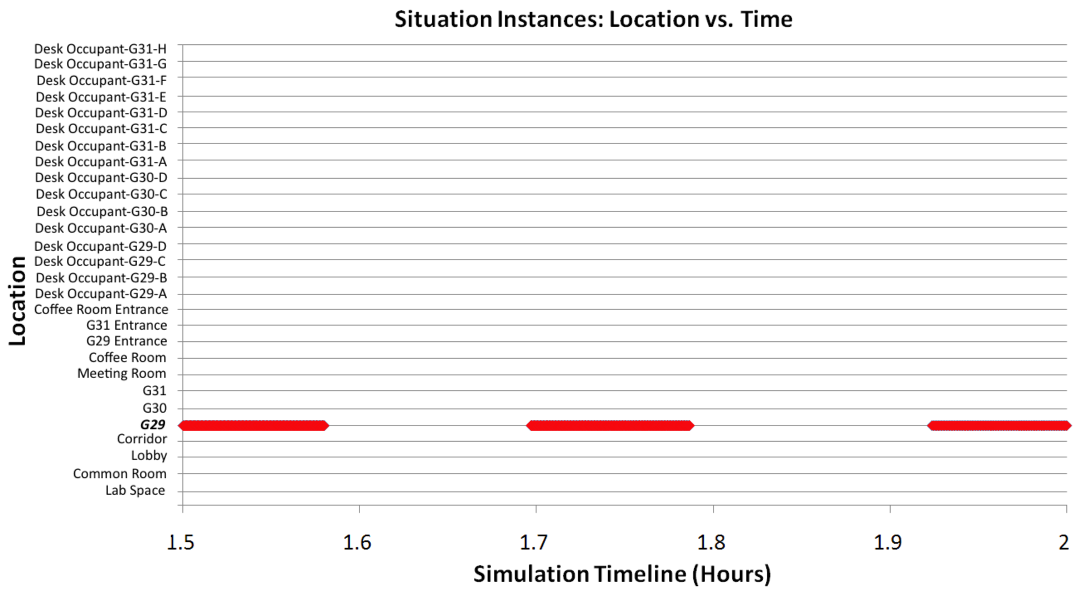


Figure B.10: Location of detected situational instances.

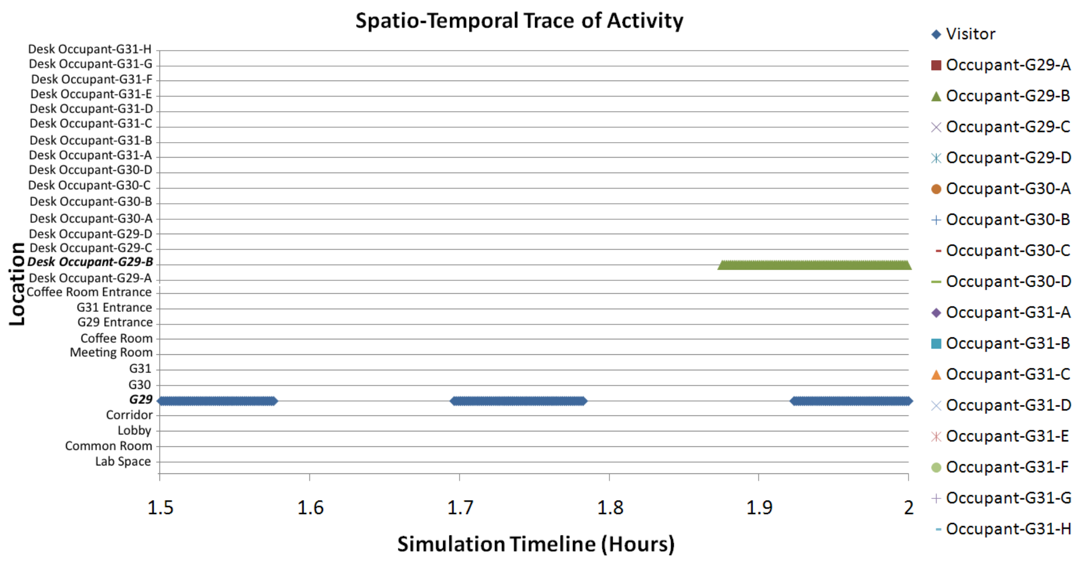


Figure B.11: Trace of spatio-temporal activity in zones contained by G29.

B.2.3 Detected Situation: Visitor locked in office space

This example illustrates the situation that a visitor gets locked in the office space. Although it is a short-lived example, it provides an interesting illustration of tracing user activity using InSitu. Figure B.12 shows that the situation was detected in room G29. Figure B.13 shows that the situation arose when the visitor entered G29 while Occupant-G29-C was present, however Occupant-G29-C left the offices almost immediately, his path can be traced along the corridor and to the lobby where he exits the building. It can been seen that Occupant-G29-B is attending a meeting and Occupant-G30-D is in the common room for the duration that the situation occurred. The situation ends when Occupant-G29-A arrives back in the offices, between 2.54 hour mark and 2.55 hour mark on the time-line.

<u>Situation:</u> *Non-occupant locked inside an office-space.*

if

(U (type = "visitor") $\longleftarrow$ v)

$\wedge$ (L ((id = v.roomLocation) $\wedge$ (type = "office")))

$\wedge$ $\nexists$ (U ((type = "staff" $\vee$ type = "student") $\wedge$ (roomLocation = v.roomLocation))

$\wedge$ $\forall$ (E (type = "external-door") :

(state = "locked"))

then

Output $O(t) \subset G(t)$

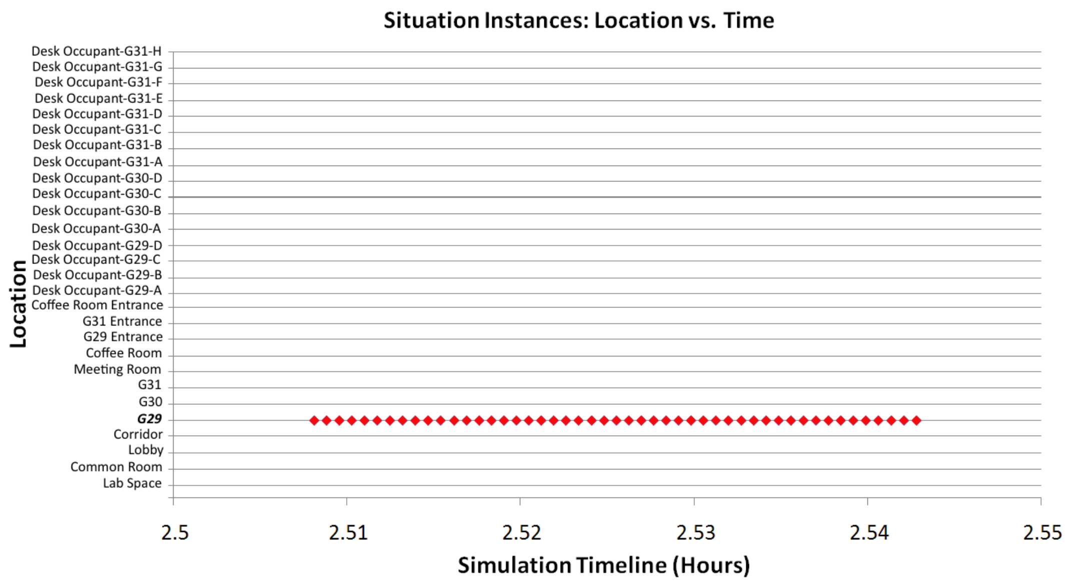


Figure B.12: Location of detected instances.

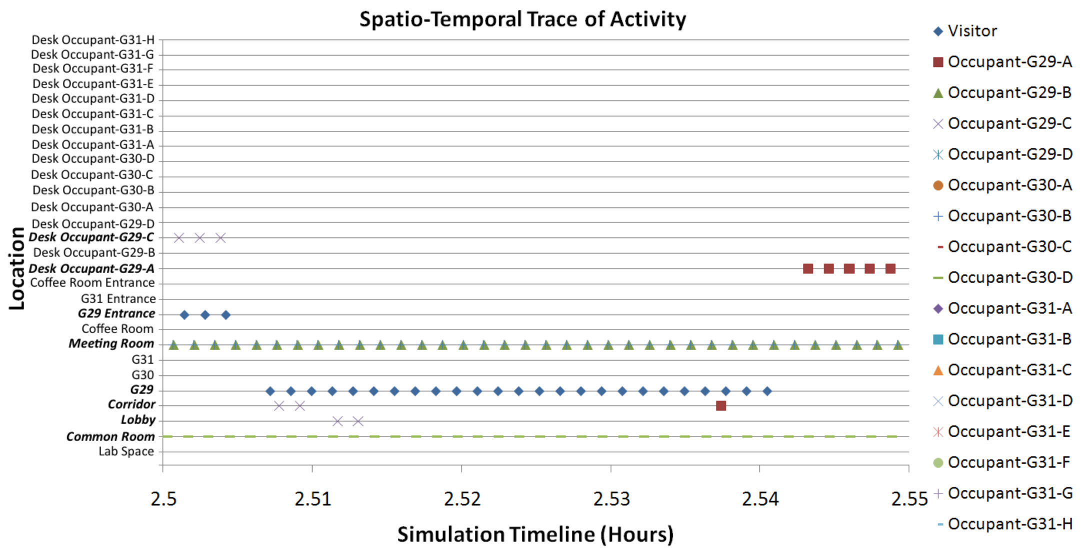


Figure B.13: Trace of spatio-temporal activity.

Appendix C

User Trial Results

This appendix includes charted results for the user trial presented in Chapter 6.

C.1 Participant Profiles

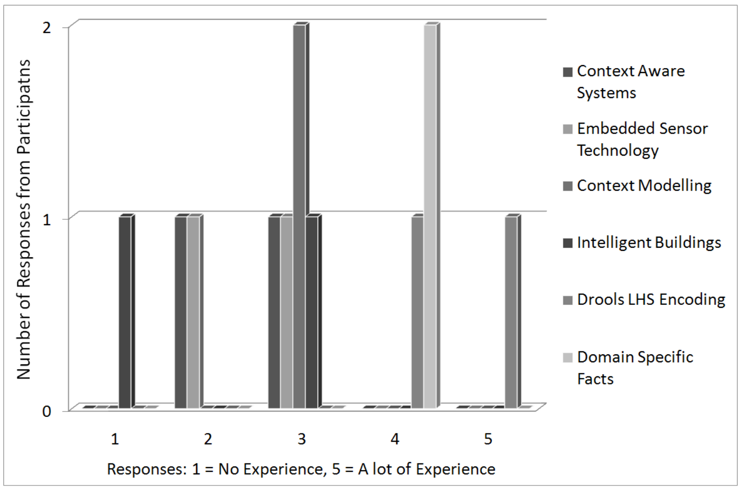


Figure C.1: Summary chart of Team A's prerequisite experience.

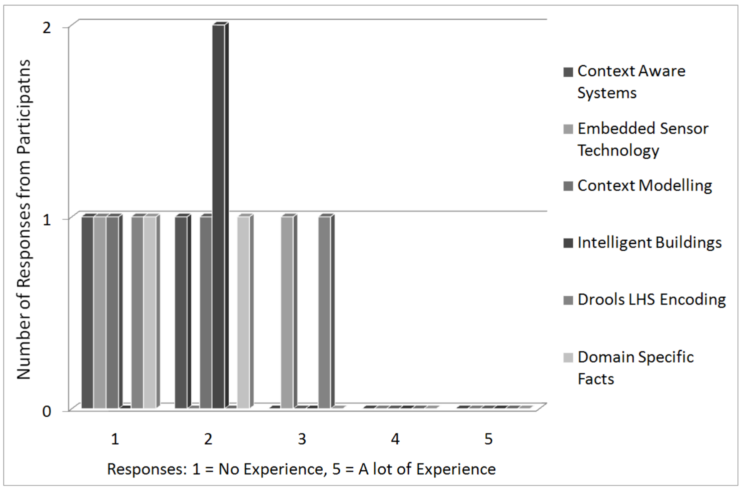


Figure C.2: Summary chart of Team B's prerequisite experience.

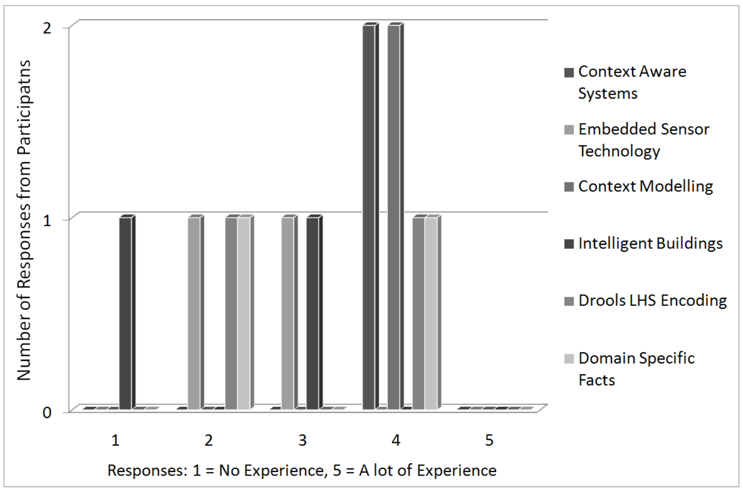


Figure C.3: Summary chart of Team C's prerequisite experience.

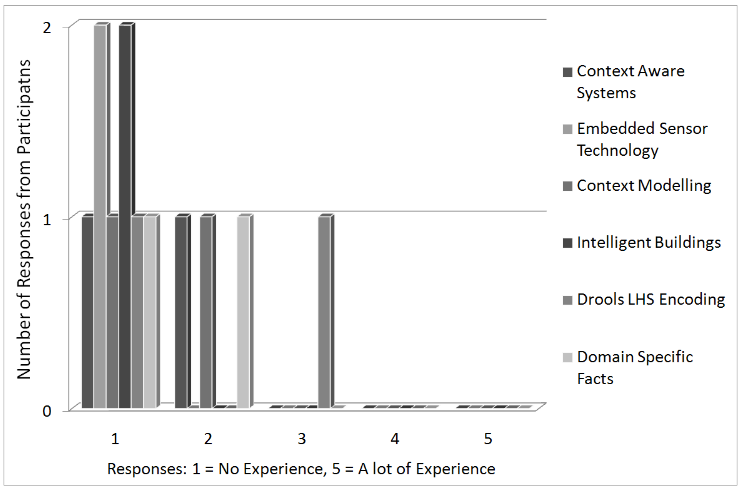


Figure C.4: Summary chart of Team D's prerequisite experience.

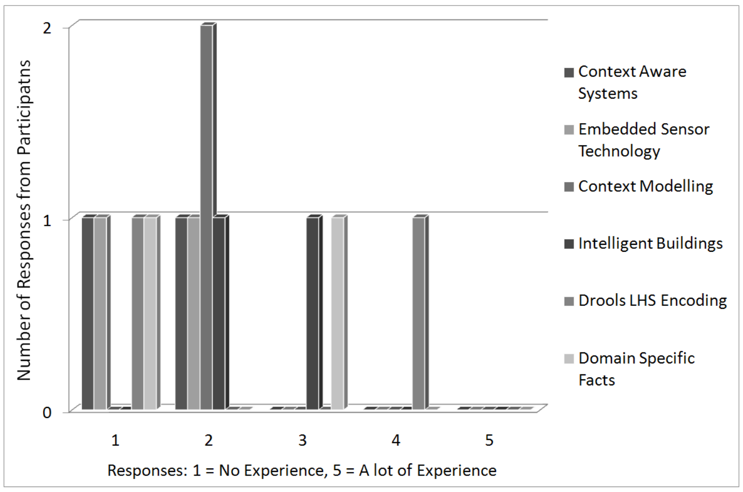


Figure C.5: Summary chart of Team E's prerequisite experience.

C.2 Pre-Experiment Questionnaire

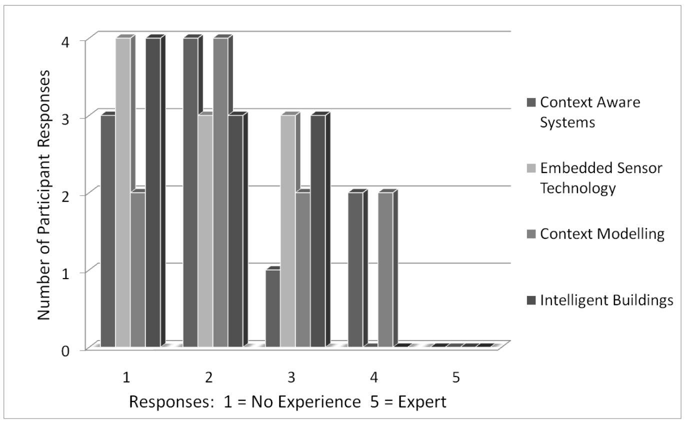


Figure C.6: Charts of participants' prior experience. **Survey Question:** Do you have experience of any of the following? (a) Context-Aware Systems / Context-Driven Behaviours (b) Embedded Sensor Technology (c) Context Modelling (d) Intelligent buildings / Intelligent Control Systems.

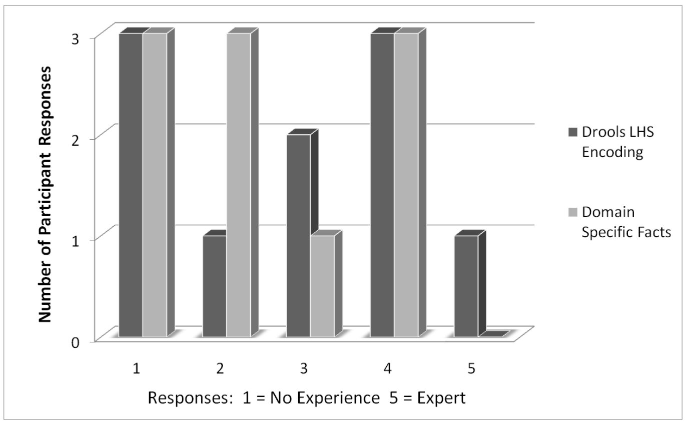


Figure C.7: Charts of participants' prior experience. **Survey Question:** Do you have experience of the following aspects of Drools? (a) Left Hand Side rule encoding (b) Domain Specific Facts.

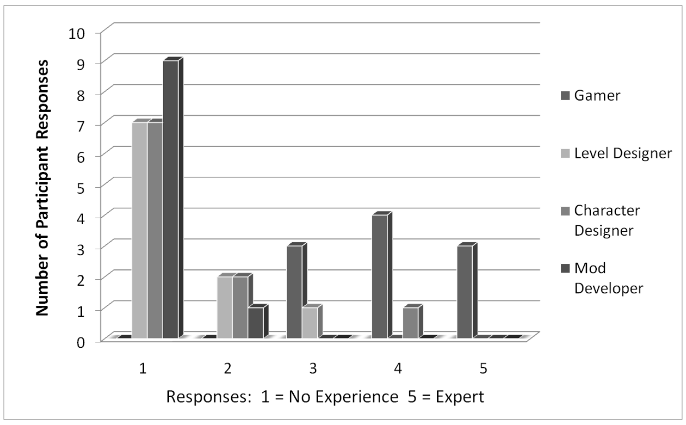


Figure C.8: Participants' prior experience with Online Games. **Survey Question:** Do you have experience of First Person Shooter games in any of the following roles? (a) Gamer (b) Level Designer (c) Character Designer (d) Mod Developer

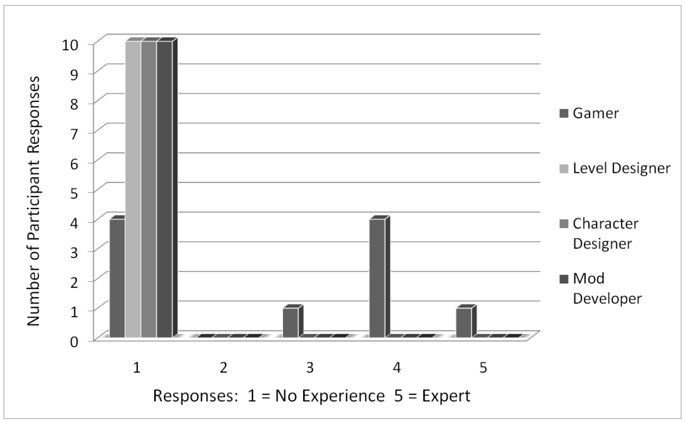


Figure C.9: Participants' prior experience with Half-Life 2 and its SDK. **Survey Question:** Do you have experience of Half-Life 2 or the Source game engine in any of the following roles? (a) Gamer (b) Level Designer (c) Character Designer (d) Mod Developer

C.3 Post-Configuration Questionnaire

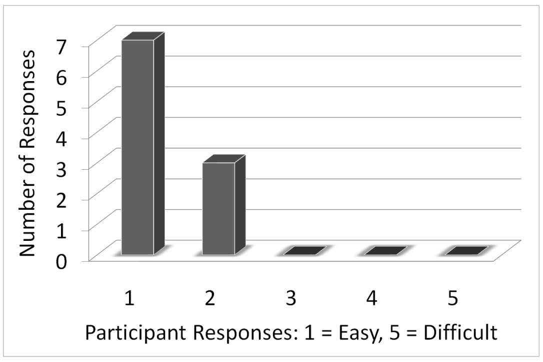


Figure C.10: Response to: *How easy was it to generate pseudocode to describe tests for your simulation?*

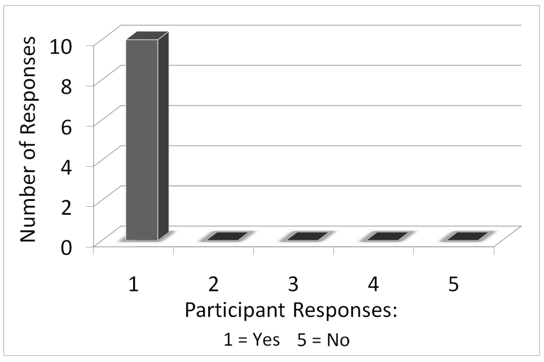


Figure C.11: Response to: *Were you able to complete the task of writing pseudocode?.*

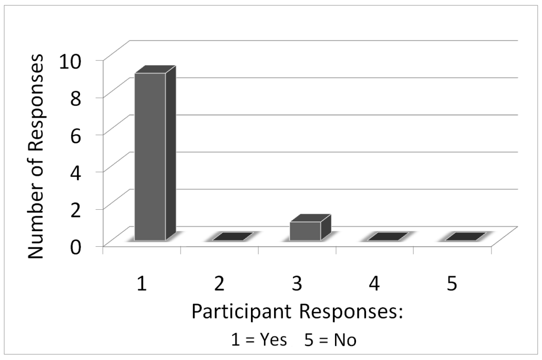


Figure C.12: Response to: *Were you able to complete the conversion of pseudocode to Drools (LHS)?.*

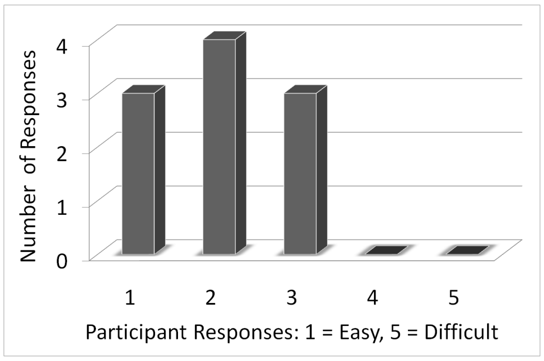


Figure C.13: Response to: *How easy was is convert the pseudocode to Drools (LHS)?*

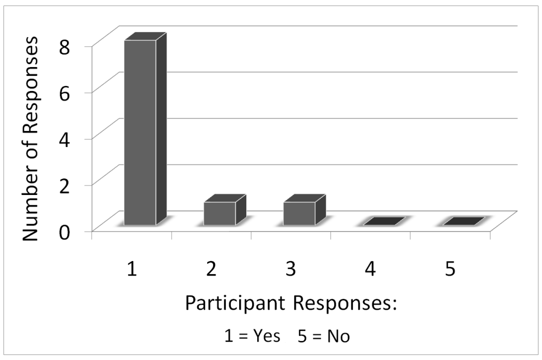


Figure C.14: Response to: *Did you find the Drools LHS sufficiently expressive for your needs?*

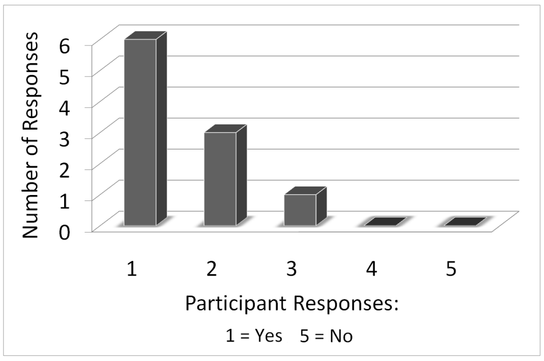


Figure C.15: Response to: *Did you find the Fact Vocabulary sufficiently expressive for your needs?*

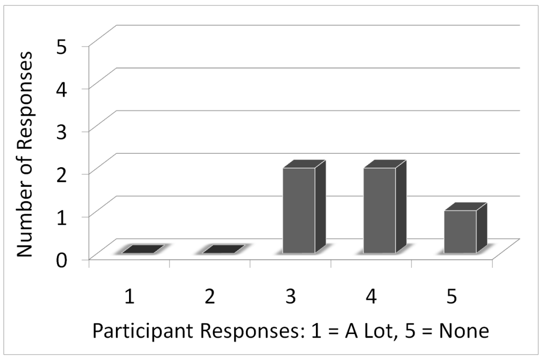


Figure C.16: Response to: *How much assistance did you need from your team partner for the Drools Model?*

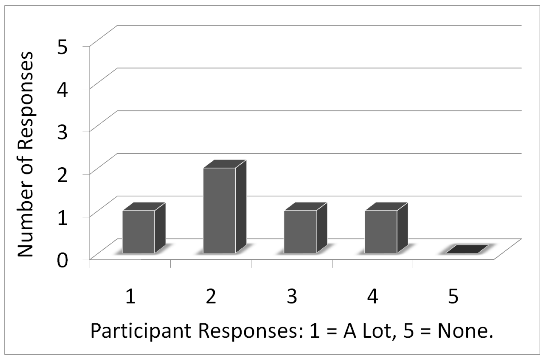


Figure C.17: Response to: *How much assistance did you need to provide to your team partner for the Drools Model?*

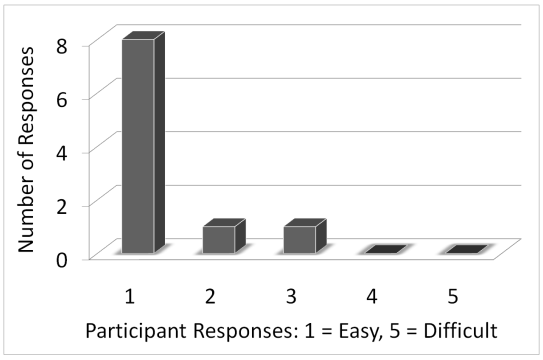


Figure C.18: Response to: *For Drools RHS, how easy was it to write the code to generate an alert?*

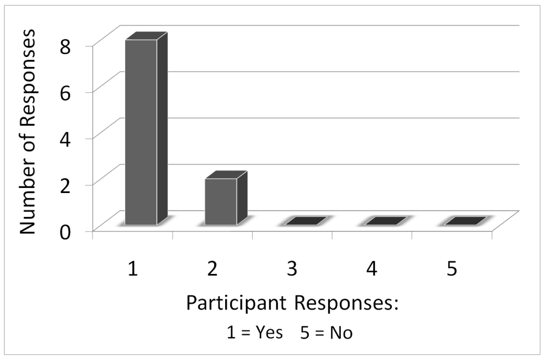


Figure C.19: Response to: *Did you find Drools RHS sufficiently expressive?*

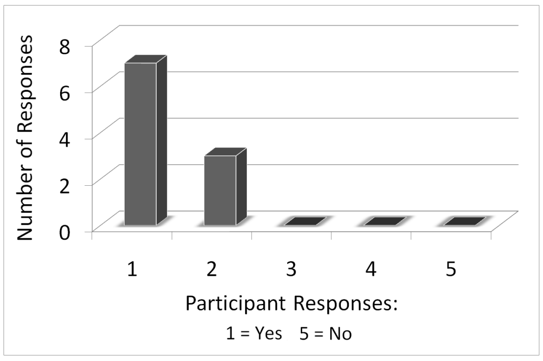


Figure C.20: Response to: *Did you find the process for defining activities using the Excel spreadsheet was systematic (i.e. there was a clear method to complete this task)?*

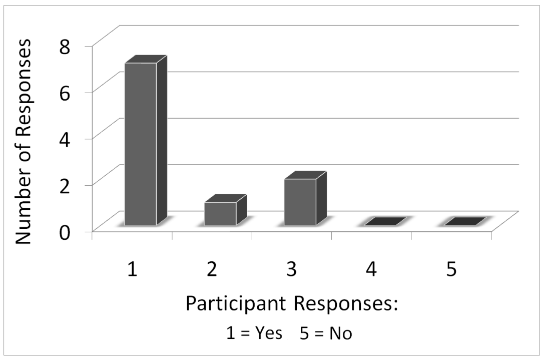


Figure C.21: Response to: *Did you find the process for transferring activity definitions from the Excel spreadsheet to Hammer was systematic (i.e.. there was a clear method to complete this task)?*

C.4 Post-Simulation Questionnaire

The first part of this section presents charted responses to the post-simulation questionnaire collected during the user trial.

In the remainder of this section, three sets of comments from the post-simulation questionnaire are presented, each set categorised as follows:

- **Conclusive Positive Responses**: Responses that conclusively indicated that participants could correctly interpret the charted alert report.
- **Inconclusive Responses**.
- **Conclusive Negative Responses**: Responses that conclusively indicate that participants were not able to correctly interpret the charted alert report.

C.4.1 Charted Questionnaire Responses

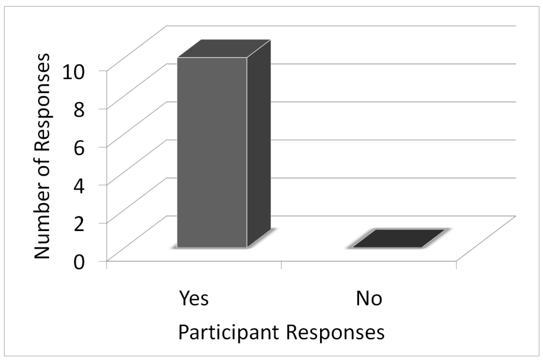


Figure C.22: Response to: *Do your results indicate that problematic behaviour exists in the prototype system?*

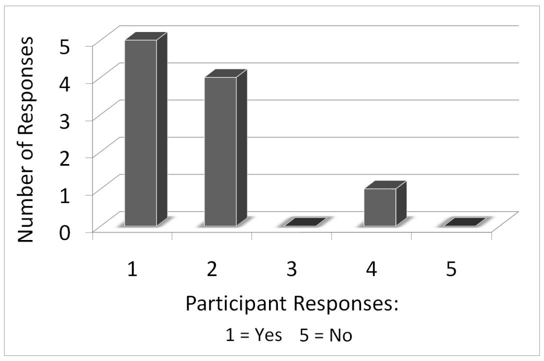


Figure C.23: Response to: *Do you think that lighting system could be improved prior to deployment based on the results from your simulation?*

C.4.2 Comments: Set 1

In response to: *Do your results indicate that problematic behaviour exists in the prototype system? Please specify the evidence from your simulation results to support your opinion.*

Conclusive Positive Responses

1. "There were at least seven instances where the bot was in the darkness, there were also several long periods where lights were left on unnecessarily."
2. "Yes, the lights were on but there was nobody at home (in the corridors and stairwells at least) so to speak. Our rules were used to log this sort of behaviour. There are sections of the building that are empty but are illuminated. This behaviour needs to be improved."
3. "The bot was left in darkness in the 1st floor corridor on 7 occasions and lights were left on for hours on end without a user being present."
4. "There seems to be a lot of light on even though no bot is there"
5. "Lights are unnecessarily on in corridor and stairwell. User in dark in first floor corridor."
6. "Lights were on when user was not present and off when the user was around."

Inconclusive responses:

1. "the blue bars in the problematic behaviour indicate this".
2. "Too many meetings and not enough work"
3. *Blank response.*

Conclusive Negative Responses:

1. “Situations with unnecessary lights on were logged. But situations for situations of user in dark is not logged. I am not sure if this situation never happened during simulation or the was not working”.

C.4.3 Comments: Set 2

Comments in response to: *If your system detected the problem of users finding themselves in the dark, how would you suggest that this the design of the lighting system could be improved? [Your suggestions can involve changes to the hardware / software, where hardware is the sensors and lights, software is the system logic]*

Conclusive Positive Responses

1. "To prevent the bot in the darkness scenario it could be possible to turn on lights in corridors/stairwells when motion is detected in an adjacent room. This would effectively eliminate the darkness problem. This functionality would require the software to understand which rooms are adjacent to each other and turn lights on accordingly. This however could cause more unnecessary lighting."
2. "Change location of motion sensor"
3. "There should be motion detectors at every door so that the lights get triggered when there is movement. By only having a limited range of motion sensors you run the risk of users being left in the dark."
4. "Add sensors down the corridors to detect when people leave rooms then turn the lights on"

Inconclusive Responses

1. "Better conflict management, may also be that hardware sensors can overide software decisions."
2. "That depends on what the problem is: (1) if user presence in that area was not detected, sensors should be adjusted, (2) if presence is detected there might be a problem with the lighting system logic."
3. *Blank response.*

4. *Blank response.*

Conclusive Negative Responses

1. “Don’t know.”
2. “If the system can detect the error, as well as logging the error, the system can take corrective action. Turn on/off the lights in this case.”

C.4.4 Comments: Set 3

Comments in response to: *If your system detected the problem that lights are on unnecessarily and waste energy, how would you suggest that the system could be improved? [Your suggestions can involve changes to the hardware / software, where hardware is the sensors and lights, software is the system logic]*

Conclusive Positive Responses

1. "To reduce unnecessary lighting the system could be redesigned to use different software timers depending on how long the user is present in an area, i.e. short timers could be used when a user is just passing through as is the case with the corridors and stairwells. The longer the user is present in an area the longer the timer should be until the lights are turned off. This approach would reduce unnecessary lighting particularly in the corridors and stairwells which from the graphs seem to be the most problematic areas."

2. "The lighting system should be closely linked to the motion detectors so that if no motion is detected within their zone for more than 60 seconds then they switch off the lights."

3. "The lighting system might need to be timed out such that if there was no user in the past X minutes it should switch off."

4. "Add sensors down the corridors to detect when people are not there and turn the lights off."

Inconclusive Responses

1. "Change location of motion sensor"

2. "Better conflict management, may also be that hardware sensors can overide [sic] software decisions."

3. *Blank response.*

4. *Blank response.*

Conclusive Negative Responses

1. “Look back at the rules for lights to make sure they are correct.”
2. “Fire a simple rule that turns off the lights. This would be the easiest approach I’d imagine but I am unfamiliar with the hardware and networking of the system.”

Appendix D

User Trial Instructions

Note: *The lighting system presented here is a case study of a real world context-driven system. If you are familiar with the problematic behaviour exhibited by this system please inform the experiment observer.*

D.1 System Description

The prototype system assessed during this experiment is an automatic lighting system. The system receives context updates from motion sensors embedded in the environment. Based on the motion sensors detecting presence in the environment, the system chooses lights to switch on. A (software) timer expires to indicate when these lights should be switched off.

The **requirements** which the system must satisfy for effective operation are:

1. Lights should automatically switch on for users according to the user's location.
2. The lighting system should be energy efficient and as such it should switch lights off when no users are present.

Follow the steps in Section D.3 to complete this task.

D.2 Deployment Environment

The deployment environment is outlined in Figures D.1 and D.2. The relationships between motion sensors and the lights that they affect are listed in Table D.1.

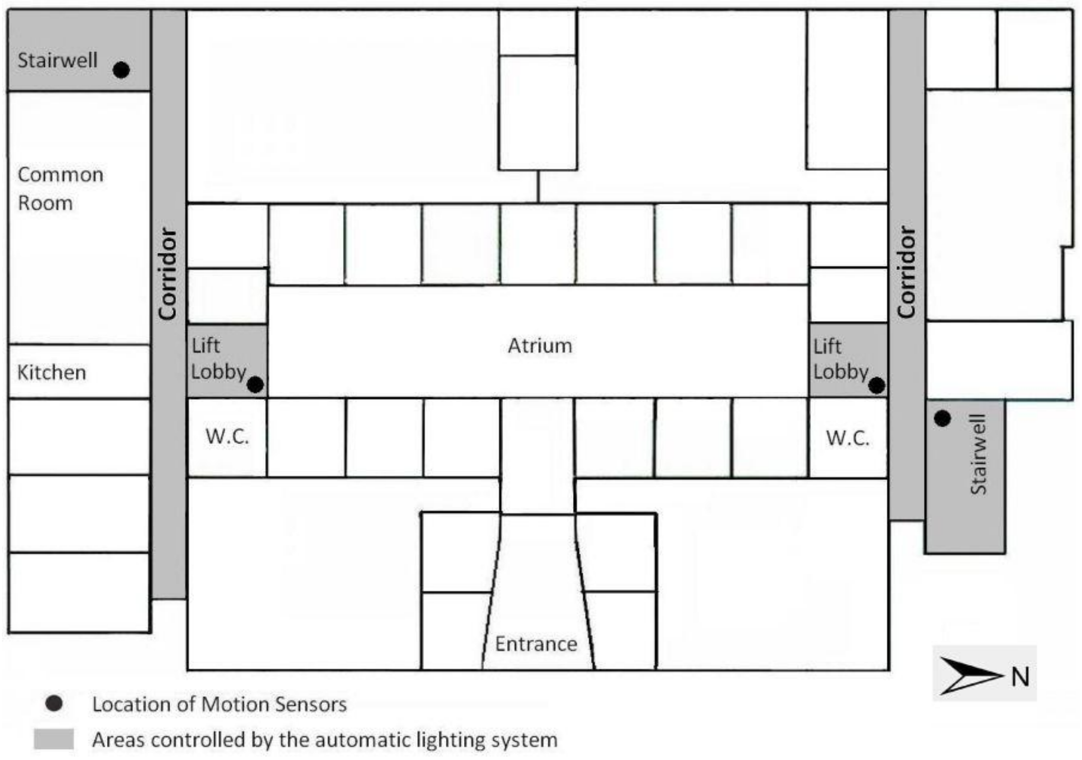


Figure D.1: Ground Floor diagram of the location of motion sensors and the areas where lighting is controlled by the automatic lighting system.

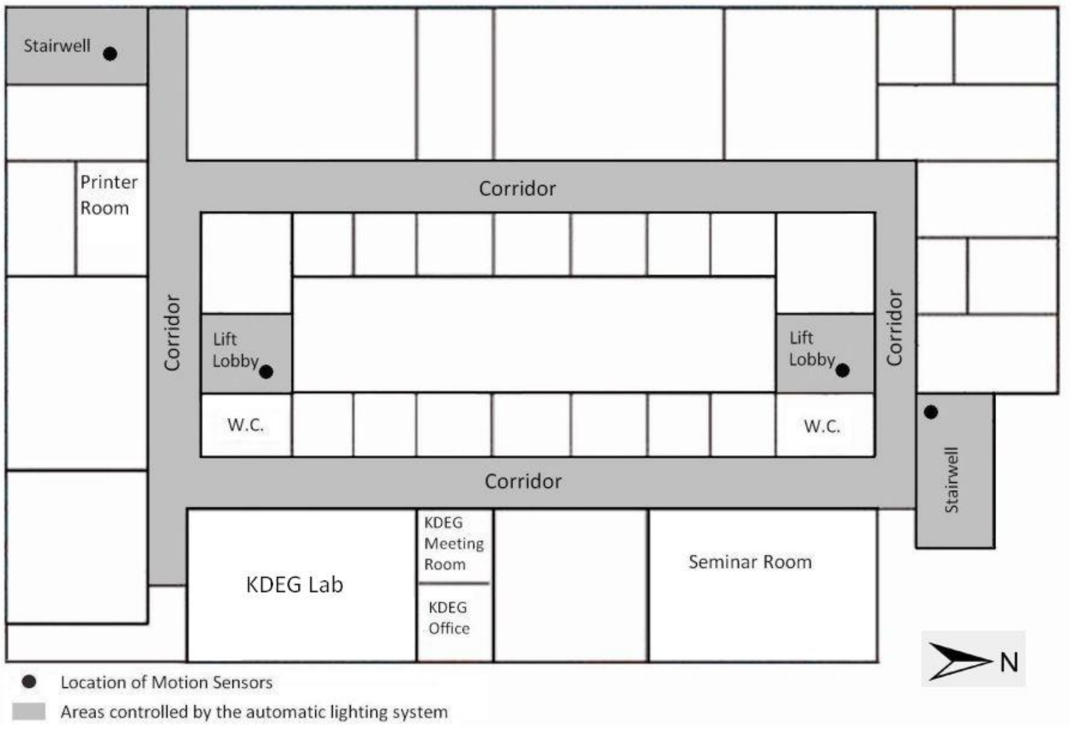


Figure D.2: First Floor diagram of the location of motion sensors and the areas where lighting is controlled by the automatic lighting system.

Motion Sensor	Light Activated
Ground Floor	
South Stairwell	South Stairwell, South Ground Floor Corridor
South Lift Lobby	South Ground Floor Lobby, South Ground Floor Corridor
North Lift Lobby	North Ground Floor Lobby, North Ground Floor Corridor
North Stairwell	North Stairwell, North Ground Floor Corridor
First Floor	
South Stairwell	South Stairwell, First Floor Corridor
South Lift Lobby	South First Floor Lobby, First Floor Corridor
North Lift Lobby	North First Floor Lobby, First Floor Corridor
North Stairwell	North Stairwell, First Floor Corridor

Table D.1: Relationships between motion sensors and the lights they switch on.

D.3 Experiment Process

D.3.1 Part 1: Write Pseudocode

[InSitu Test Process: **ITP1**[1]]

- Write high level pseudocode to test for problematic behaviour the lighting system described in Section D.1.
- Problematic behaviour occurs any time the system does not satisfy the requirements, listed on page 285.
- Tests examine the state of the environment as perceived by an end-user. Do not try to examine how or what the system is doing.

Example Situation: When a door fails to open, an end-user does not perceive the cause of the failure. The end-user perceives their environment and the problem they are faced with i.e. they cannot enter the room. The end-user does not perceive any of the following:

- A motion sensor failing to detect their presence
- Software failing to instruct the door to open
- The mechanical arm on the door failing to complete an actuation instruction from the intelligent control software.

Example Pseudocode: *This example provides a demonstration of how exhibited behaviour is tested for an access control system.*

```
When
user location equals door location
And user access permissions is equal to or greater than door access permissions
And door state not equal open
```

[1]Codes indicate relationship of each step in this set of user instructions to the InSitu Test Process defined in Chapter 4, Figure 4.6.

Then

Log this problematic behaviour.

End

D.3.2 Part 2: Convert Pseudocode to Drools LHS

[InSitu Test Process: **ITP1**]

System Behaviour Model tests are written using the Left-Hand-Side (LHS) of Drools. These Drools rules are tests which define unwanted behaviour for the context aware system.

These rules test context information which is modelled using the Context Fact Vocabulary.

When a test evaluates to true, i.e. unwanted behaviour has been detected, the Right-Hand-Side (RHS) outputs a record of the incident to an XML file.

Encode the pseudocode from Section D.3.1 as Drools rules.

- Guide to LHS is provided on page 290
- Guide to RHS is provided on page 292
- Domain specific fact vocabulary is specified on page 294.

Drools LHS:

Using the Context Fact vocabulary, page 294, write the set of LHS rules to check for unwanted behaviour exhibited in the environment.

Example Rule: LHS

In the example below the designer uses the value "visitor" to test the *userType* for an individual user. The only constraint on literals used in the rule engine is that they must be consistent with the terms used in the VR engine. For example, a user called *Alice* in the VR engine, should be tested by userID == "Alice" in the rule engine. These literals are defined as values of the property sets in the map using the Hammer Map Editor. Tables D.2, D.3 and D.4 denote existing literal values inside curly brackets { }.

Listing D.1: Example Code

```
rule ``ID.3: Access Control''
when
$user: UserStateFact (userType==``visitor'', $userLocation: userLocation)
$location: LocationStateFact (location == $userLocation,
  access == ``staff-only'')
then
// A visitor has accessed a privileged area, the access control system
// may have failed.  Output something to the Alert Report.
end
```

D.3.3 Part 3 : Write Drools RHS

[InSitu Test Process: **ITP1**]

The RHS of each Drools rule is used to output an XML log entry about the problematic behaviour detected by the rule. Three functions are used to manipulate the XML, listed below: *addNode, addAttribute* and *navigateModel.* Use the example on page 293 as a guide.

XML Navigation and Manipulation Functions

addNode (String modelname, String node)

> Adds a new node to the Alert report.
>
> Attributes:
>
> *modelname* [String] Refers to the Alert report model. Always use *sar_model* for this
>
> *node* [String] Value of the tag name to use.

addAttribute (String modelName, String attribute, String attribute_value)

> Adds a new attribute to the current node.
>
> Attributes:
>
> *modelname* [String] Refers to the Alert report model. Always use *sar_model* for this
>
> *attribute* [String] Attribute name to use.
>
> *attribute_value* [String] Value for the attribute.

navigateModel (String modelName, String location)

> Change the current location within the model.
>
> Attributes:
>
> *modelname* [String] Refers to the Alert report model. Always use *sar_model* for this
>
> *location* [String] Refers to the path to take to the new location e.g. “..” OR “/”

Example RHS Code

Note that only the code in italics needs to be changed for new rules. Keep the rest of the code the same to remain compliant with the XML schema for this file. The example at the end of this page shows the XML output by the RHS code. The XML lines in italics correspond to the italicised lines in the code.

rule “ID.3: Access Control”
when
$user: UserStateFact (userType=“visitor”, $userLocation: userLocation)
$location: LocationStateFact (location == $userLocation, access == “staff-only”)
then
nav.addNode(sar_Model, “alert”)
nav.addAttribute(sar_Model, “id”, gen.getNextAlertID());
nav.addNode(sar_Model, “rule”);
nav.addAttribute(sar_Model, “id”, “ID.3”);
nav.navigateModel(sar_Model, ”..”);
nav.addNode(sar_Model, “location”, $userLocation);
nav.addNode(sar_Model, “user”, $user.getUserID());
nav.addNode(sar_Model, “time-detected”, $user.getTimestamp());
nav.navigateModel(sar_Model, “/”); control.storeModel(“data”, sar_Model, “SAR_model.xml”);
end

RHS Code XML Output

<alert id=“1”>
<rule id=“ID.1” />
<location>south_stairwell</location>
<user>test_user</user>
<time-detected>518953635</time-detected>
</alert>

Context Fact Vocabulary

Context is modelled using JBoss Facts. Facts are continually updated at run-time with **all** state information about the environment. A vocabulary of three fact types has been defined to include entities, users and locations.

EntityStateFact types are populated with information about devices in the environment. Table D.2 defines the attributes for entities.

UserStateFact types are populated with information about users in the environment. Table D.3 defines the attributes for entities.

LocationStateFact types are populated with information about the structure and physical configuration of the environment. Table D.4 defines the attributes for entities.

Attribute Name	**Type**	**Description**
entityName	String	Unique identifier e.g. light-lift-lobby-south-0. Values correspond to the Target name of entities in the properties dialog box of Hammer.
entityType	String	{light, kproxy_motion_sensor}. Values correspond to the classname of entities in the properties dialog box so this list is extensible.
entityState	String	{on, off, open, closed, locked} State of a fixed/embedded/mobile device. For example a light is 'on' or 'off'.
entityLocation	String	Location identifier. Corresponds to locationID attribute of the Location Fact type.

Table D.2: EntityStateFact Attributes with Getter/Setter Functions

Attribute Name	Type	Description
userID	String	Unique identifier. Values correspond to the Target name of *npc_citizen* bots in the properties dialog box of Hammer.
userLocation	String	Location identifier. Corresponds to locationID attribute of the Location Fact type.

Table D.3: UserStateFact Attributes with Getter/Setter Functions

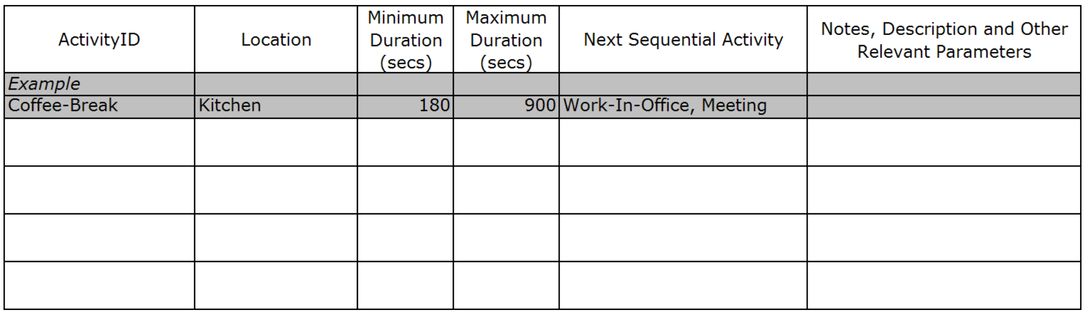
Attribute Name	Type	Description
locationID	String	Unique identifier for a location.
locationType	String	{stairwell, corridor, room, area, floor, building}

Table D.4: LocationStateFact Attributes with Getter/Setter Functions

D.3.4 Step 4: Defining User Activities: Excel Spreadsheet

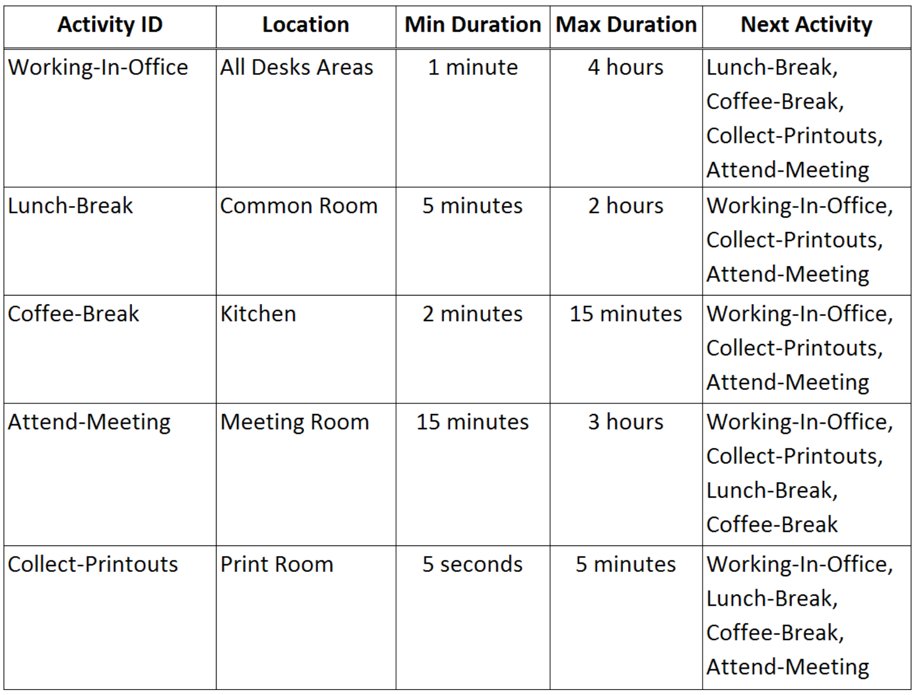
[InSitu Test Process: **ITP1**]

Use the Activities Excel spreadsheet, illustrated in Figure D.3 to define the activities which might take part in this office building. Activities should reflect realistic daily behaviour of users of the building. Figure D.4 contains a set of activities which can be used in this experiment. Table D.5 provides further information about the purpose of each column in the spreadsheet.



ActivityID	Location	Minimum Duration (secs)	Maximum Duration (secs)	Next Sequential Activity	Notes, Description and Other Relevant Parameters
Example					
Coffee-Break	Kitchen	180	900	Work-In-Office, Meeting	

Figure D.3: The Activity Spreadsheet for Activity Modelling. The first line illustrates an example activity.



Activity ID	Location	Min Duration	Max Duration	Next Activity
Working-In-Office	All Desks Areas	1 minute	4 hours	Lunch-Break, Coffee-Break, Collect-Printouts, Attend-Meeting
Lunch-Break	Common Room	5 minutes	2 hours	Working-In-Office, Collect-Printouts, Attend-Meeting
Coffee-Break	Kitchen	2 minutes	15 minutes	Working-In-Office, Collect-Printouts, Attend-Meeting
Attend-Meeting	Meeting Room	15 minutes	3 hours	Working-In-Office, Collect-Printouts, Lunch-Break, Coffee-Break
Collect-Printouts	Print Room	5 seconds	5 minutes	Working-In-Office, Lunch-Break, Coffee-Break, Attend-Meeting

Figure D.4: List of Activities

Column Name	**Description**
ActivityID	A unique identifier for the activity. **Must be unique and only contain hyphens i.e. no symbols and no spaces.**
Location	The name of the room/area in which this activity will be located. **Only one location per cell; for an activity that happens in more than one location, use one line per location**.
Minimum Duration (secs)	The minimum amount of time **(in seconds)** which an activity should last for under normal conditions i.e. not including emergency situations, but instead focusing on reasonable everyday human behaviour.
Maximum Duration (secs)	The maximum amount of time **(in seconds)** to correspond with the previous field.
Next Sequential Activity	A list of the set of possible activities which might be performed by a bot. The game engine randomly selects one of these. **Must correspond to values in the *ActivityID* column.**
Notes, Description and Other Relevant Parameters	A space for the test designers to add notes about an activity. This text box is not directly related to the Activity Models in Hammer.

Table D.5: Column Definitions for Activity Excel Spreadsheet

D.3.5 Step 5: Setting up User Activities in the Hammer Map Editor.

[InSitu Test Process: **ITP1**]

- Open the map in Hammer. Values from each of the columns of the spreadsheet will be used here to set up the properties for the activities.
- For gamers, check that the mouse inversion is set to your preference in the 3D view.
- Throughout this section, where the column name from the Excel spreadsheet appears in italics, it is representing the current value from that column. For example in line 7 of Figure D.3 *ActivityID* refers to the value *CoffeeBreak*.
- During step D.3.5, it may be easiest if one team member reads the instruction sheet while the other team member works with Hammer.

Create New Activity

1. Copy the example Activity group of entities.
2. Move to the location of the new Activity.
3. Point with the mouse crosshair in the 3D view and paste the new Activity in the appropriate location.
4. If entities are hidden be careful not to place an activity in or on furniture.

   Scripted Sequence Settings:

5. Select the **scripted_sequence** entity.
6. Open the properties dialog. **Alt + Enter**
7. Enter the value from the **ActivityID** column as the **Name**.

8. Check that **Target NPC** contains **test_user**.
9. Click **Apply**

<u>Start Timer Settings:</u>

10. Select the **start timer**.
11. Open the properties dialog box, if not already open. *Alt + Enter*
12. Enter **"start-"+ActivityID** as the **Name** e.g. for an activity called lunch, enter **start-lunch**.
13. Check that **Start Disabled** is set to **Yes**
14. Check that **Use Random Time** is set to **No**
15. Set Refire Interval to 5
16. Click **Apply**

<u>End Timer Settings:</u>

17. Select the **end timer**.
18. Open the properties dialog box, if not already open. *Alt + Enter*
19. Enter **"end-"+ActivityID** as the **Name** e.g. for an activity called lunch, enter **end-lunch**.
20. Check that **Start Disabled** is set to **Yes**
21. Check that **Use Random Time** is set to **Yes**
22. Set **Minimum Random Interval** to **Minimum Duration** from the Excel Spreadsheet.
23. Set **Maximum Random Interval** to **Maximum Duration** from the Excel Spreadsheet.

24. Click **Apply**

Logic Case

25. Select the **logic_case**.

26. Open the properties dialog box, if not already open. **Alt + Enter**

27. Enter "case-"+ActivityID as the **Name** e.g. for an activity called lunch, enter **case-lunch**.

28. Click **Apply**

Activity I/O

Start Timer

29. Select the **start timer** and open the properties dialog box.

30. Select the **Outputs tab**.

31. Select the **BeginSequence line**.

32. Select the **eye-dropper tool**. (On the right of the **Target Entities named** box).

33. Click on the **scripted_sequence** in the 3D view using the eye-dropper tool. (Move the properties dialog box if necessary).

34. Click the **Apply** button.

Scripted Sequence

35. Select the **scripted_sequence** entity and open the properties dialog box.

36. Select the **Outputs tab**.

37. Select the **OnBeginSequence line**.

38. Select the **eye-dropper tool** beside the **Target Entities Named** box.
39. Click on the **start timer** entity in the 3D view using the eye-dropper tool. (Move the properties dialog box if necessary).
40. Select the **OnEndSequence line**.
41. Select the **eye-dropper tool** beside the **Target Entities Named** box.
42. Click on the **end timer** entity in the 3D view using the eye-dropper tool. (Move the properties dialog box if necessary).
43. Click the **Apply** button.

<u>End Timer</u>

44. Select the end timer and open the properties dialog box.
45. Select the Outputs tab.
46. Select the Disable line.
47. Select the eye-dropper tool beside the *Target Entities named* box.
48. Click on the end timer in the 3D view using the eye-dropper tool. (Move the properties dialog box if necessary).
49. Select the PickRandomShuffle line.
50. Select the eye-dropper tool beside the *Target Entities named* box.
51. Click on the logic_case entity (the last of the four activity entities) in the 3D view using the eye-dropper tool. (Move the properties dialog box if necessary).
52. Click the **Apply** button.
53. **Repeat steps 1 to 52 for each activity listed in the Excel Spreadsheet.**

Activity Logic

- Revisit the Excel Spreadsheet.
- Use the first activity in the excel spreadsheet.

1. In Hammer, under the Edit menu choose **Find entities** .... and enter the **ActivityID** into the dialog box that appears. Click OK.
2. Select the **logic_case entity** and open the properties dialog box.
3. Open the **Outputs tab**.
4. Click the **Add...** button.
5. In the **My output named:** droplist choose **OnCase01**. (If the activity has more than one possible next step, use a different case here for each possible next step).
6. In **Target entities named:** type **"start-"ActivityID** i.e. the name of the start time associated with the **next step** activity. (Auto-complete will assist here).
7. In **Via this input** type Enable.
8. Click the Apply... button.
9. **Repeat steps 4-8 until all next steps have been added to the logic_case**.
10. Move to the next Activity on the Excel spreadsheet and **return to step 1**.

Set the start trigger for the test:

1. In the Edit menu choose Find entities.... and enter *start_experiment* into the dialog box that appears.

2. Open the properties dialog box and open the **outputs** tab.
3. Choose the **Enable** line.
4. In *Target Entities named* box type *"start-"+ActivityID* for the first activity that should be performed to begin the experiment.
5. Click the **Apply** button.

Compile the Map:

- Save the map as teamID_lloydmap e.g. team1_lloydmap.
- Press F9 and make sure the first four check boxes are ticked. Press the 'Go' button.

D.3.6 Run Test/Simulation Configuration and InSitu Monitoring Engine

[InSitu Test Process: **ITP2**]

D.3.7 Review charted alert report to analyse system behaviour.

[InSitu Test Process: **ITP3**]

D.4 Questionnaire

Please ask the experiment observer for your next questionnaire.

D.5 Experiment Finished

Thank you.

D.6 Pre-Experiment Hammer Tutorial

D.6.1 Hammer Map Editor

D.6.2 1. Opening a Map in Hammer

1. Open Steam.
2. Open the Tools tab
3. Open Source SDK
4. Open Hammer
5. In Hammer, open the file Lloyd_UserTrial.vmf.

2. Navigation in Hammer

2D view:

- **Zoom:** Hold the mouse pointer over a 2D view. Use the mouse scroll button to zoom in and out.
- **Pan:** Use scroll bars to pan across the 2D view.

3D view:

- Hold the mouse pointer over the 3D view.
- Press 'z' to toggle mouse control to the 3D view.
- Press 'w' to move forward.
- Press 's' to move backwards.
- Press 'a' to move sideways to left.
- Press 'd' to move sideways to right.

3. **Editing a Map with Hammer**

- **Select:** Use the mouse in the 3D view to select an entity. Hold the CTRL key to select more than one entity. Tip: When an entity is selected, its name is visible at the bottom of the Hammer window; this is a useful aid to help determine that you have selected the correct entity.
- **Move:** Use drag and drop to move entities/objects in the environment.
- **Copy:** Standard copy and paste will duplicate entities. Use the mouse crosshair, in the 3D view, to point to the location where the Activity should be pasted.
- **Properties Dialog Box:** Use a dialog box for selection entities to edit their properties (Alt + Enter).
- Tip Hide Entities: To clear the map and make it easier to work with the 2D views uncheck the Entities box in the VisGroup list (bottom right corner).

Appendix E

Situation Visualisation

The realism of the virtual environment provides testers and designers with a useful tool for intuitive views of detected situations. This also provides an opportunity to demonstrate system conflicts to end-users where the solution to problematic or inappropriate behaviour is not a straight-forward decision. In these cases, designers can play back problematic situations to end-users of the system to collect their opinion about how the system should operate. This example is taken from the lighting case study, which is presented in full in Chapter 6.

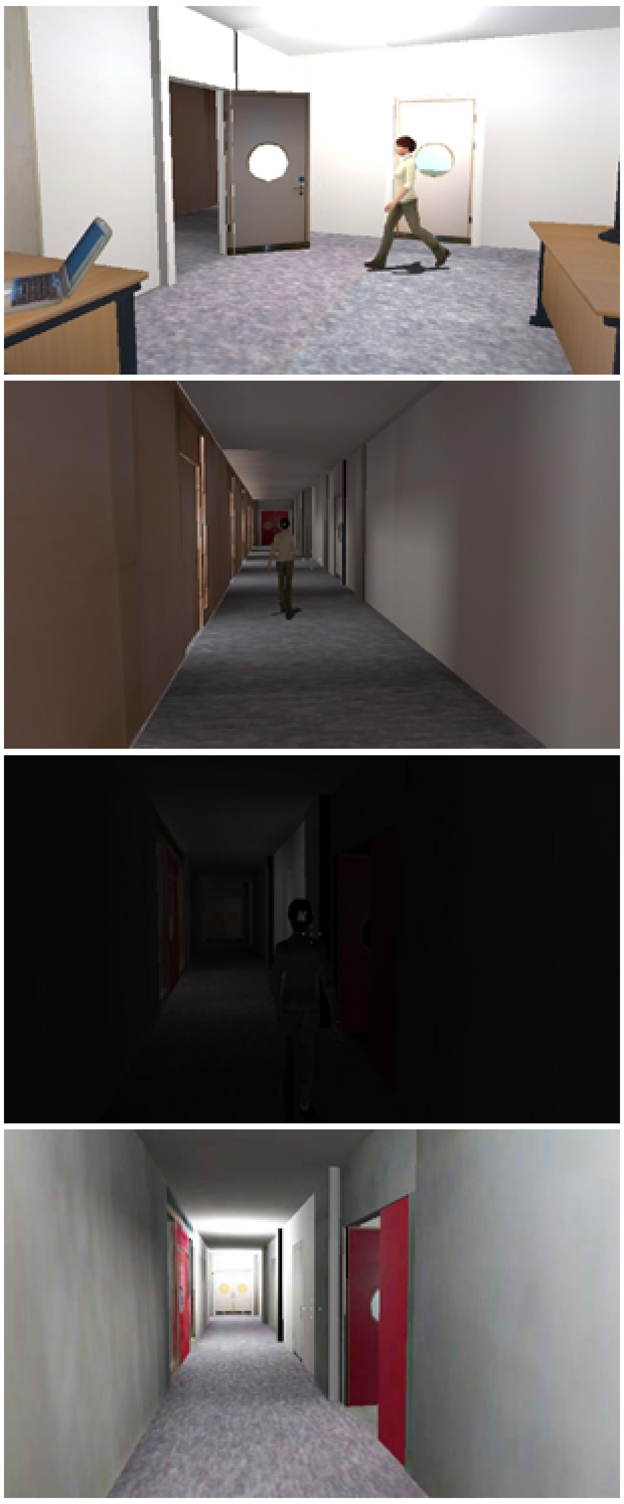


Figure E.1: 1) An occupant finishes work and leaves the office. 2) The occupant enters corridor to move to another location. 3) The lights have been switched off and the corridor is in darkness. 4) The occupant activates the motion sensor in the stairwell and the lights switch on but the occupant is not longer present.

Figure E.1[1] provides an example of a simulated user encountering an inappropriate situation in the deployment environment. Figure E.1 (top) begins by showing a female worker leaving her office. The corridor next to her office is in darkness when she leaves. There is no motion sensor installed outside her office, only the ambient lighting from neighbouring offices illuminates the corridor, as seen in Figure E.1 (second from top). When she reaches a section of the corridor without ambient lighting she is very much in the dark. Looking at Figure E.1 (third from top), it is just about possible to see the outline of her head and shoulders. In this image there is a stairwell exit point immediately to her right, however in the darkness it is almost impossible to see this. She exits through this door and as she steps into the stairwell she activates the motion sensor installed in the stairwell and the lights turn on in both the corridor and stairwell, see Figure E.1 (bottom).

[1]Demo video available through Appendix K or at http://www.youtube.com/watch?v=imvkKnhNdVM.

Appendix F

CD Table of Contents

The back cover of this thesis includes a CD containing supplemental appendices for this thesis. The following details the table of contents for the supplemental appendices included on the CD.

Contents