UNIVERSITY
of
GLASGOW

Contextual Mobile Adaptation

Malcolm Hall

Doctor of Philosophy
Department of Computing Science
Faculty of Information and Mathematical Sciences
University of Glasgow
May 2008

Abstract

Ubiquitous computing (ubicomp) involves systems that attempt to fit in with users' context and interaction. Researchers agree that system adaptation is a key issue in ubicomp because it can be hard to predict changes in contexts, needs and uses. Even with the best planning, it is impossible to foresee all uses of software at the design stage. In order for software to continue to be helpful and appropriate it should, ideally, be as dynamic as the environment in which it operates. Changes in user requirements, contexts of use and system resources mean software should also adapt to better support these changes. An area in which adaptation is clearly lacking is in ubicomp systems, especially those designed for mobile devices. By improving techniques and infrastructure to support adaptation it is possible for ubicomp systems to not only sense and adapt to the environments they are running in, but also retrieve and install new functionality so as to better support the dynamic context and needs of users in such environments.

Dynamic adaptation of software refers to the act of changing the structure of some part of a software system as it executes, without stopping or restarting it. One of the core goals of this thesis is to discover if such adaptation is feasible, useful and appropriate in the mobile environment, and how designers can create more adaptive and flexible ubicomp systems and associated user experiences. Through a detailed study of existing literature and experience of several early systems, this thesis presents design issues and requirements for adaptive ubicomp systems. This thesis presents the Domino framework, and demonstrates that a mobile collaborative software adaptation framework is achievable. This system can recommend future adaptations based on a history of use. The framework demonstrates that wireless network connections between mobile devices can be used to transport usage logs and software components, with such connections made either in chance encounters or in designed multi–user interactions.

Another aim of the thesis is to discover if users can comprehend and smoothly interact with systems that are adapting. To evaluate Domino, a multiplayer game called Castles has been developed, in which game buildings are in fact software modules that are recommended and transferred between players. This evaluation showed that people are comfortable receiving semi–automated software recommendations; these complement traditional recommendation methods such as word of mouth and online forums, with the system's support freeing users to discuss more in–depth aspects of the system, such as tactics and strategies for use, rather than forcing them to discover, acquire and integrate software by themselves.

Contents

List of Figures

Acknowledgements

Having been lucky to be part of the Equator group at Glasgow, I have been fortunate to work collaboratively with a large number of excellent researchers on many exciting projects, and had the opportunity to present work at conferences throughout the world. Specifically, I would like to thank our leader and inspiration Dr. Matthew Chalmers, and group members Dr. Marek Bell, Scott Sherwood, Dr. Barry Brown, Julie Maitland, Dr. Louise Barkhuus, Dr. Alistair Morrison, Paul Tennent, Jose Rojas and John Ferguson. I would also like to extend my thanks to my Equator friends outside of Glasgow with whom I have had the pleasure to collaborate on projects and attend workshops.

I would like to thank Prof Stephen Brewster for inviting me to do a summer project that inspired me to enter academic research.

I would like to thank Dr. Marek Bell, a fellow Ph.D. student who was one year ahead of me and led the adventure, inspired me to succeed, and was a valued friend.

I would like to thank my supervisors Dr Matthew Chalmers and Mr Phillip Gray for their support and guidance through the course of this research.

I would like to offer special thanks to Dr Matthew Chalmers, Dr Marek Bell and Eve Hoggan, for their motivation and proofreading. Without them, this thesis would not have the quality it is—but I take responsibility for all remaining errors!

Finally, I would like to thank my parents Allan and Irene, and brother Michael, who have always encouraged and supported me during my work on this thesis.

The work described within this thesis has been funded by EPSRC grant GR/N15986/01 as well as an EPSRC Doctoral Training Award associated with that grant.

Declaration

The contents of this thesis are the author's personal work. However, many of the systems discussed within this thesis have been designed and implemented as part of the Equator group at the University of Glasgow and have been accomplished, in part, with contributions from others in the Equator IRC.

The author has attempted to make clear when and by whom systems have been designed and implemented with others. However, the author has been one of the main designers and programmers of every system developed by the University of Glasgow Equator group. Specifically, the author was the main designer and programmer of *Far Cry, StreetHawk, Domino*, and part of a group of designers and programmers of *George Square*, *Treasure*, *Feeding Yoshi* and *Castles. Samara* was primarily the work of Marek Bell but the author modified the implementation of the peer-to-peer recommendation system for use with Castles. Contributions from others have been made only to system design and implementation – the work and research related to the thesis itself is entirely the author's own.

The concept of *Seamful Design*, discussed in 2.2.5 is primarily that of Matthew Chalmers. The author's work did not lead to the idea of Seamful Design; rather the author merely applies it to an adaptive system and highlights its importance.

List of Publications

In each of the following publications, the author has been either the primary author or a co-author. Each is related to or has influenced the work in this thesis, and some material from them has been included in this thesis.

Hall M., Hoggan E., Brewster S., *"T-Bars: Towards Tactile User Interfaces for Touchscreen Mobiles"*, Proc. Mobile HCI 2008, Amsterdam, Netherlands.

Barkhuus L., Brown B., Bell M., Sherwood S., Hall M. and Chalmers M., *Friendship, Awareness and Repartee: Sharing location on the go,* Proc. CHI 2008.

Marek Bell, Malcolm Hall, Matthew Chalmers, Phil Gray and Barry Brown, *Domino: Exploring Mobile Collaborative Software Adaptation,* Proc. Pervasive 2006, Dublin, Ireland, pp. 153-168.

Bell, M., Chalmers, M., Barkhuus, L., Hall, M., Sherwood, S., Tennent, P., Brown, B., Rowland, D., Benford, S., Capra, M. & Hampshire, A., *Interweaving Mobile Games with Everyday Life,* Proc. ACM CHI 2006, Montreal, 2006, pp. 417-426.

Malcolm Hall, Marek Bell & Matthew Chalmers, *Domino: Trust Me I'm An Expert*, Workshop on Software Engineering Challenges for Ubiquitous Computing 2006, Lancaster, UK.

Tennent, P., Hall, M., Brown, B., Chalmers, M., and Sherwood, S., *Three Applications for Mobile Epidemic Algorithms*, in MobileHCI 2005, Glasgow, UK.

Matthew Chalmers, Marek Bell, Barry Brown, Malcolm Hall, Scott Sherwood & Paul Tennent, *Gaming on the Edge: Using Seams in Ubicomp Games* Proc. ACM Advances in Computer Entertainment (ACE) 2005.

Brown, B., Chalmers, M., Bell, M., MacColl, I., Hall, M. & Rudman, P., *Sharing the square: collaborative leisure in the city streets,* Proc. Euro. Conf. Computer Supported Collaborative Work (ECSCW), Paris, 2005.

Matthew Chalmers, Marek Bell, Barry Brown, Malcolm Hall, Scott Sherwood & Paul Tennent, *Using Peer-to-Peer Ad Hoc Networks for Play and Leisure*, 3rd UK-UbiNet Workshop, 2005, Bath, UK.

Barkhuus, L., Chalmers, M., Hall, M., Tennent, P., Bell, M., Sherwood, S. & Brown, B., *Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game,* Proc. Ubiquitous Computing (Ubicomp), Tokyo, LNCS 3660, pp. 358-374, 2005.

Bell, M., Chalmers, M., Brown, B., MacColl, I., Hall, M. & Rudman, P., *Sharing photos and recommendations in the city streets*, Pervasive Workshop on Exploiting Context Histories in Smart Environments (ECHISE), 2005.

Chalmers, M., Bell, M., Hall, M., Sherwood, S. & Tennent, P., *Seamful Games*, Adjunct Proc. 6th International Conference on Ubiquitous Computing, UbiComp 2004, Nottingham, England.

Malcolm Hall, Philip Gray, *Mobile Support for Team-Based Field Surveys*, Lecture Notes in Computer Science, Volume 3160, Jan 2004, Pages 431–435.

Brewster, S., Lumsden, J., Bell, M., Hall, M. and Tasker, S., *Multimodal 'eyes-free' interaction techniques for wearable devices,* Proc. CHI 2003, Florida, USA.

Chapter 1 Introduction

This thesis presents a study of dynamic software adaptation in the field of ubiquitous computing (ubicomp). There is an increasing consensus among researchers that software adaptation is becoming increasingly desirable as the use of computers extends beyond work activities, focusing on pre–planned tasks, into ubicomp where they are used increasingly for leisure and daily domestic life. Here, the variety and dynamics of people's activities, contexts and preferences make it especially challenging for the designer to foresee all possible functions and modules, and their transitions, combinations and uses.

By making systems more adaptive in nature they may, in turn, become more accountable, and behave fluidly and appropriately in an increasing variety of situations and environments. A simple but well–known example is of a mobile phone that models its context, reacting when it is in a lecture theatre by switching to silent mode—so that any incoming calls do not disturb the lecture.

In Mark Weiser's paper, "The Computer for the 21st Century" [1], he mentions that, perhaps in the future, the role of portable computers will change. Weiser proposes devices named *Pads* that "are intended to be scrap computers (analogous to scrap paper) that can be grabbed and used anywhere; they have no individualized identity or importance." In order to support such use, any system running on the Pad would have to adapt to the current user and situation, dynamically configuring the components running on the Pad and loading in new ones as necessary to present a customised interface and set of applications. This could be thought of as a sea of generic, but highly adaptive, Pad devices, whose individual usefulness or significance is only realised as waves of users discover and interact with them.

In the not too distant past, the vast majority of people did not own personal or mobile computers, and their use tended to be only in the workplace. Nowadays, it is common for people to own numerous computers, including digital cameras, mobile phones, PDAs and laptops. As the popularity of these mobile devices increases, manufacturers are eager to introduce a continuous cascade of new features in an attempt to further the success of their product. However, in doing so the feature set of the device expands. These manufacturers are, in effect, invading the markets occupied by other device types, which themselves have respectively expanding feature sets. For example, many of the new mobile phones being released in the next few months will have 802.11 wireless capabilities, and there are already 802.11 equipped PDAs that have GSM/GPRS/3G abilities. Furthermore, mobile phones and PDAs can be used as personal music players, and have models available that have integrated digital cameras, which recently have become of adequate quality to produce high-resolution prints.

It is hard to predict when, or if, this cross-convergence of markets will unite and to what extent the different devices, or parts of them, will combine. Furthermore, it is impossible to accurately envision what such an emerging device will look or feel like. However, as Weiser alludes, it may possibly be a generic and highly adaptive device powered by a very intelligent adaptive infrastructure, which elegantly fulfils all the user's requirements and expectations, as well as the constraints and interdependences of the configuration it is used in.

As might be expected, with all of these added features and cross-convergence of markets, the modern mobile device is capable of sensing its surroundings more than ever. Where we go, our interests, what we do, whom we meet in our everyday lives, and much more can be detected by the devices we carry. Basically, the 'who, what, where and when' of computing is becoming exposed to developers and users of mobile devices. Yet, despite the huge advancements in mobile devices, and the obvious advantages of the ubicomp concept, adaptive and dynamic systems are practically non-existent. Barkhuus and Dourish describe both problems and opportunities for adaptation on mobile devices [2]:

> *Amongst the problems are the difficulties of managing power, locating people, devices and activities, and managing interactions between mobile devices. Amongst the opportunities is the ability to adapt to the environment. Recognizing that different places and settings have different properties and are associated with different activities, researchers have become interested in how computational devices can respond to aspects of the settings within which they are used, customizing the interfaces, services, and capabilities that they offer in response to the different settings of use.*

This reinforces the view that, rather than a static design built on presumptions of use by designers, incremental adaptation and ongoing evolution under the control of users is appropriate [3] [4] [5]. There is now a wealth of high performance mobile devices on which to run ubicomp systems, and concepts as to how these systems should behave (incremental adaptation and ongoing evolution), yet there are no systems currently in existence fulfilling these needs for adaptation of deep software structure during use. That is, there are no mobile systems that sense a user's context, and dynamically adapt their features and functionality, adding or removing computational software modules, in accord with that model of context and the history of how users' contexts have changed in the past.

The lack of such systems implies that there may be other underlying obstacles present in the development of software for these mobile devices. As previously mentioned, mobile users can be in a variety of different situations when using their devices. The fact that designers of mobile systems cannot possibly foresee every possible situation, environment, or individual user needs further strengthens the argument that software has to be designed differently, allowing it to adapt to these ever-changing factors. Mobile devices are so widely used and woven into people's lives that it is a good starting point for ubicomp development.

Existing mobile applications in general use are often cut down or limited versions of desktop software. They are static and inflexible and unaware of the user's current situation. These mobile device applications do not offer anything unique over what a desktop PC can already do. For example, on Windows Mobile there is Pocket Word, Pocket Excel, Calendar and Tasks, all of which are just cut-down versions of full Windows software. There is a task manager to switch between or quit running applications. Similarly the iPhone has cut-down versions of iCal, Safari and Mail. All these applications, running on mobile versions of Windows and OS X, provide at best the same, and at worst reduced, functionality when compared to their desktop counterparts. None of them take into account users' locations, surroundings or other context. Instead of making the applications adapt to fit the users' tasks and workflow, they rely on the user to alter his or her behaviour around the device and application. Symbian does not have such applications as the operating system is not based on a desktop operating system but, as with the others, its interaction methods are still rooted in desktop concepts. Most desktop applications are designed for use in static environments. If mobile device applications are to become more adaptable and dynamic, static desktop designs are not appropriate. The research operating system TinyOS[1] is an open-source operating system designed for wireless embedded sensor networks. Experimentation with adaptive component-based TinyOS architectures targeted at ubicomp systems are under development [6], however this work ran only in simulations, ran into the issues of finding appropriate bindings (i.e. matching dependencies) for new components, and focused more on dynamic architectures on such highly constrained devices such as sensor networks rather than user-oriented research.

Another obstacle in the path of context aware, dynamic and adaptive mobile systems development, is the fact that telecom operators supply the majority of mobile phones, and it is in their interest to restrict the functionality of their devices to secure additional revenue streams, besides the monthly line rental fees. For this reason, operator-branded phones are often locked down. The opinion of most operators is that if users were able to install VOIP applications they could bypass phone call charges, and if people could share ringtones and games freely the operators would lose out on the additional revenue they gain from those products. These restrictions imposed by mobile phone operators are a large part of the reason why mobile application innovation is being constrained.

[1] http://www.tinyos.net

An additional important issue is that the most useful information in determining context is often secured and unavailable for programmers to use, or if it is available, it is presented in a device specific way. For example, obtaining cell tower identification numbers in a range of mobile phones is a task that can be very difficult. If this information was accessible, it could give way to many location aware applications. However, operators commonly wish it to be locked away from the general developer, so that they—the phone operators—can charge for any location-aware functionality they supply themselves.

A further setback with mobile devices is that they heavily vary in hardware; as a result, it can be difficult to choose a platform and language that is easy to develop for, even though mobile devices are ubiquitous enough. Fortunately, solutions to this problem may come in the near future, with the recent announcement of a new consortium whose goal is to develop open standards for mobile devices, which it is hoped, will allow developers to work more collaboratively to develop more flexible user experiences. Google Inc., T-Mobile, HTC, Qualcomm, Motorola and others have collaborated on the development of the Android platform through the Open Handset Alliance[2]. As there is no Android device available yet, it is not yet known if the operators will impose any constraints on the devices to secure revenue as mentioned above.

Lastly, and perhaps most importantly, there are many issues with the tools available to mobile application developers. There are few guidelines for developing mobile experiences and thus the full potential of the mobile device has not been explored. Developers resort to porting desktop applications to mobile devices and most are redundant when the user has a desktop or laptop available. Mobile software or services have the potential to be far more unique and novel, and to actually complement existing desktop software rather than deliver simple, weak, awkward to use cut down versions of desktop applications. One example might be a device that logs context information when the user is mobile, and then automatically synchronises with the user's desktop, on which it can be visualised or processed. A mobile device is often a very personal device, which one carries everywhere. It thus has potentially available a wealth of information that could be recorded and put to good use. Ongoing use can be logged and combined with past use in various ways. For example, the use of graphical user interface component can be logged by recording when the user clicks on the component or when the component is visible. For non-user interface components alternative ways of logging are possible. For example, for a process that discovers libraries of shared music on a network it would be possible to log the times at which it discovers a music-sharing peer device. These kinds of applications could be, potentially, as ground breaking and useful as the original purpose of mobile phones, making phone calls and sending SMS. The Domino framework of Chapter 5 is an attempt to address these issues and meet the need for adaptive dynamic solutions. Domino has been designed to be a flexible framework capable of processing and responding to various different types of log data. An example system described in Chapter 6, Castles, demonstrates one subset of these techniques.

[2] http://www.openhandsetalliance.com

1.1 Research questions

The Equator style of work, described in Section 1.2, demonstrates several aspects of system design and use, ranging from technical design to user experience. Rather than focus on just one of these in a single thesis statement, the work of this thesis followed several interwoven threads. This thesis aims to identify key aspects of existing systems and practices that, when combined, can potentially provide the basis for a more dynamic and adaptive approach to mobile software in ubicomp scenarios. By investigating common themes in related areas of research, and outcomes of some initial experience projects, this thesis presents the design and implementation of *Domino*, a flexible, context-aware, mobile, peer-to-peer application framework. The research presented in this thesis primarily focuses on a study of using histories or logs of software module use in a mobile adaptive system as a basis for recommendations of modules for use in similar contexts. Furthermore, this study concentrates on the specifics of using these history logs as a means of discovering the most effective technique for linking new software modules to existing sets of modules. This work can be framed as a series of research questions, as follows:

RQ1 *What aspects of existing research and systems can be applied to the design of an adaptive infrastructure for ubicomp?*

RQ2 *How can ubicomp researchers design more dynamic and adaptive systems?*

RQ3 *How do users react to adaptive and dynamic ubicomp systems?*

1.2 The Role of Equator

All of the work discussed in this thesis has been conducted as part of Equator, a six-year, ten million pound Interdisciplinary Research Collaboration that involved eight UK universities. The Equator Group at the University of Glasgow has been at the forefront of international ubicomp research for the past six years, creating new ubiquitous devices, establishing distributed software platforms to knit many such devices together, working with external partners to demonstrate innovative and creative applications of ubicomp, and studying these and then generalising the lessons learned into new design concepts and frameworks. Through Equator and related projects we successfully laid the foundations for a new interdisciplinary approach to ubicomp research that involves taking emerging technologies out of the laboratory and studying them 'in the wild'. In so doing we have placed the UK in a world-leading position in this field.

Equator had a large number of resources and researchers, and was a unique environment for a student. Being part of this project enabled me to take part in the creation of several systems throughout the PhD, working at a scale and pace that might not have been possible without the support of Equator. Furthermore, Equator has been responsible for numerous trials of these systems covering technical, architectural, interaction and user issues. In common with Equator, neither Artificial Intelligence (AI) nor Intelligent User Interfaces have been the approach used in this work. In Equator's sociotechnical approach, people are part of the systems being designed and studied, as opposed to AI's tendency to focus on computers acting more autonomously.

The approach used in this PhD research is based on the Equator approach, and took advantage of Equator's iterative and reflective research process. Each system built and trialled during the project produced many findings that were fed into the design of the next system. Domino and Castles are therefore based on many findings from earlier Equator systems. For instance, the much simpler model of data distribution used successfully in FarCry (Chapter 4) influenced the adaptation feature of Domino. Similarly the history based recommendation technique applied to content in George Square (Chapter 2), was used for software component recommendations in Domino. All systems developed as part of this thesis are part of the Equator project:

- George Square – a tourist co-visiting system, demonstrating history based contextual recommendations of content and a modular architecture (2.2).
- Treasure – a location based game applying Seamful Design to WiFi and GPS (3.2.1).
- Feeding Yoshi – a game used to research how mobile games can be played over a relatively long period of time fitting with the everyday life of the user, which applied a Seamful Design to the use of ad hoc WiFi networks (3.2.2).
- FarCry – an application demonstrating how peer-to-peer WiFi connections can support epidemic distribution of media files (4.2).
- Domino – the backbone system of this thesis, an architecture for the design of highly adaptive ubicomp systems (5.2).
- Castles – a game demonstrating one possible use of Domino that was trialled to ascertain users reactions to such a system (Chapter 6).

Throughout this thesis, the part the author played in each of the collaborative work of creating these systems will be made clear.

1.3 Approach

RQ1 will be addressed through a literature review, detailed analysis of systems and user studies in the areas of context awareness, seamful design, mobile peer-to-peer applications and adaptive systems (Chapters 2–5). An answer to RQ2 given in this thesis will be based on the design and implementation of Domino, an adaptive infrastructure for ubicomp presented in the second half of Chapter 5 and was specifically designed in response to the findings from studies of the systems investigated to address RQ1 (as discussed in the earlier chapters). RQ3 will be addressed through the implementation and findings from a user trial of a Domino-based adaptive dynamic ubicomp system, Castles, where users found recommendations and associated system adaptation generally useful, and yet were also subjects of individual interpretation and social discussion.

1.4 Thesis Walkthrough

Four themes were identified as relevant to the design of adaptive ubicomp systems: context awareness, Seamful Design, mobile peer-to-peer systems and adaptation. These form four chapters containing both background research in the area, and the selected systems listed above. Findings from these chapters build the rationale for the Domino adaptation architecture, presented at the end of the adaptation themed chapter. Following this is an investigation into the Domino-based adaptive ubicomp system, Castles.

Chapter 2 examines existing literature in the area of context awareness and explains how it is the first step towards adaptive ubicomp systems. The George Square system is presented, and features history-based, contextual recommendations of places and photos to tourists using a mobile application.

Chapter 3 examines Seamful Design and presents two systems, Treasure and Feeding Yoshi. The Seamful approach is proven to be successful in allowing users to understand complex technologies in meaningful ways, and suggests a similar technique may be successful in the design of an adaptive ubicomp architecture, which is likely to be highly complex.

Chapter 4 has a review of existing literature in the field of mobile, peer-to-peer systems. The FarCry, peer-to-peer Recer and Samara systems are presented in order to demonstrate that mobile peer-to-peer applications can exploit social proximity between co-located people and high bandwidth, ad hoc wireless connections, and to suggest that adaptive ubicomp systems could exploit these attributes too.

Chapter 5 examines existing literature in the field of adaptation. An architecture for a mobile software recommendation and adaptation framework is presented,that encompasses all positive features of the mobile systems identified in the background litereature and the author's early systems.

Chapter 6 presents the implementation of an application based on the Domino platform and an investigation of its use. A multiplayer mobile strategy game, Castles, is presented along with an evaluation that covers technical aspects as well as user experience. Also the chapter presents the findings from the development of Domino, its outcomes and future directions.

Finally, Chapter 7 concludes by reviewing the thesis and its contributions which address the research questions raised in the introduction.

Chapter 2 Context awareness

Context awareness is a key issue relating to how systems can become more adaptive. This chapter discusses background work in the field of context awareness research and then focuses on a context aware system built during this PhD.

2.1 Background

The aim of this thesis is to address the issue of how to make ubicomp systems more dynamic and adaptable, which is widely regarded as a necessary step towards the ideals of ubicomp. A greater variety of contextual information, representing more of the user's activity, can be obtained from a mobile device than a desktop computer. The ability to sense the surrounding environment opens up possibilities for allowing software to adapt so as to behave better under constantly changing circumstances and thus become much more useful. Certain types of contextual information can be logged and distributed between large groups of people to allow more powerful adaptation techniques, for example generating recommendations based on comparisons between people. Therefore, this section reviews some of the literature in context-aware systems.

Dey, Abowd and Salber define context as [7]:

Any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves

Thus context can be regarded as the situation or environment, both physical and social, in which a computational system is present. Context-aware systems are designed to allow constituent devices to acquire and utilise information about the context, for example the place, time, or people nearby. Dey and Abowd also give a definition of a context-aware system [8]:

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

Dey and Abowd [8] conducted a survey of existing literature on context-aware systems and attempted to make clearer definitions of context, as well as re-examining the categories that are most vital to context-aware systems. They started by reiterating why context awareness is of fundamental importance to mobile systems:

The increase in mobility creates situations where the user's context, such as the location of a user and the people and objects around her, is more dynamic. Both handheld and ubiquitous computing have given users the expectation that they can access information services whenever and wherever they are. With a wide range of possible user situations, we need to have a way for the services to adapt appropriately, in order to support the human-computing interaction.

Schilit et al. reiterated the value of context in mobile computing and suggested a new class of software that adapts to changes in environment [9]:

One challenge of mobile distributed computing is to exploit the changing environment with a new class of applications that are aware of the context in which they are run. Such context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment.

Schilit et al. begin to consider the types of contextual information that might be useful in driving potentially useful adaptations. Dey and Abowd identify what they believe are normally the four most important items of context [8]:

> *There are certain types of context that are, in practice, more important than others. These are* location*,* identity*,* activity *and* time.

Hull et al. [10] identified that a further crucial item of context was to sense the presence of other people using the same system. *Hummingbird* was a small device equipped with a short-range radio transceiver, through which it broadcast its identity and received information about other Hummingbirds in the vicinity [11]. Another system, *Jabberwocky* was designed to extend the 'familiar stranger' relationship between strangers in public places by providing unobtrusive awareness notifications [12]. Other such awareness systems are examined in Section 4.1.1, which discusses software designed to exploit proximity between co-located users.

Hull et al. discuss the importance of context-awareness in wearable computers; clothes augmented with small computers and various sensors. Wearable computers are typically designed to support the natural workflow of the user conducting their task; that is, wearers typically do not need to interrupt their normal tasks to make use of the wearable computers abilities, or even interact with it. In this way, wearable computers have the potential to support users moving around as they normally would. It appears that the more potential for system mobility, the more crucial appropriate adaptation behaviour is [10]:

> *Situation awareness is particularly valuable for wearable devices. Desktop computers live in a very static environment. Even notebooks mostly only make the trek to office to home and back. However, wearable computers will (potentially) go everywhere with their owners, into a wide variety of situations in which appropriate behaviour for a given situation might be essential.*

Hull et al. suggest mobile devices, especially wearables, have a unique opportunity for capturing an extremely rich level of context. It is both appropriate and necessary for mobile devices to be continually *sensing* their environment and *reacting* to it.

Pascoe points out that context-aware systems should not make the mistake of concentrating solely on observing the external environment [13]:

> *Context-awareness is the ability of a program or device to sense various states of its environment and itself.*

Pascoe also reiterates the sentiments of Hull et al. on the potential value of context information to mobile devices [13]:

The intimate association of user and computer in a wearable system leads to the computing resources being accessed in a diverse array of situations, unlike a static desk-bound computer. It is this multitude of dynamic contextual factors that allows context-awareness to be exploited particularly well in wearable computers.

Pascoe defines a set of core generic capabilities suggested as vital to context-aware systems [13]:

1. *Contextual sensing: the ability of a system to detect various environmental states and to feed information about the current state or changes to it back to the user.*
2. *Contextual adaptation: the ability of a system to tailor itself to the current situation.*
3. *Contextual resource discovery: the ability of a system to detect and to take advantage of the resources it discovers in its environment (e.g. peer devices).*
4. *Contextual augmentation: the augmentation of additional information to elements in the environment. For example, this might be embodied in a tour system which provides additional information on interesting buildings or statues in the environment.*

These four generic context-aware capabilities are highly relevant to the work in this thesis and are discussed throughout. Context sensing, resource discovery and contextual augmentation are a major part of many of the systems implemented as part of the research described in this thesis such as *George Square*, *Treasure* and *Feeding Yoshi.* In addition to those three systems, the *Domino* adaptation architecture covered in 5.2 also exploits the use of contextual adaptation to support the development of more adaptive ubicomp systems.

Pascoe believes contextual adaptation to be of core importance:

Applications can leverage this contextual knowledge by adapting their behaviour to integrate more seamlessly with the user's environment. Rather than providing a uniform service regardless of the user's circumstances, the context-aware computer can tailor itself to the current situation. For example, adapting behaviour for a particular user.

The user experience in a context-aware, mobile system is directly linked to the adaptation of the underlying computer system. Mobile devices can, and should, adapt to both the current situation and the current user. Whilst Pascoe implemented a trial system—which uses some context features such as contextual sensing—for supporting ecologists conducting fieldwork and found that 'the context-aware features were attributed as a critical part of the system's success', it proved too challenging to implement contextual *adaptation* in the first iteration. Pascoe explains:

> *Contextual adaptation was not used at all in the current prototype. This capability could provide the fieldworker with assistance by automating actions in certain contexts.*

Pascoe believes contextual adaptation can be of great value but did not have time to implement it in his prototype because of difficulty. This may explain why such adaptive systems have not been developed so far, and provides motivation for the design of useful architectures or toolkits to provide this functionality.

After examining the literature, it is apparent that Weiser's famous vision of ubicomp must rely on context awareness [1]:

> *Hundreds of computers in a room could seem intimidating at first, just as hundreds of volts coursing through wires in the walls did at one time. But like the wires in the walls, these hundreds of computers will come to be invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks.*

It is clear that if mobile devices are to achieve Weiser's vision for ubicomp—fading into the walls, the background, until needed—then it is crucial that they become more adaptive and flexible than they currently are, so as to be able to respond appropriately, and only when needed.

If mobile systems that take full advantage of contextual awareness are to be powerful or even popular, system design must drive adaptation down to their core level. The literature mentions that factors such as sensing, adaptation, nearby resource discovery, location, identity and time are important and must be included in a successful system that takes full advantage of contextual awareness. This thesis continues to examine other literature that suggests what the features of a particularly adaptive architecture would be. Some of the earliest system design work of the research described in this thesis is a system that took a first step in expanding the usual model of context used in ubicomp by applying the historical view of context to a collaborative ubicomp system, thus revealing important architectural requirements for later work in this thesis: George Square, Section 2.2.

There is clearly a large volume of research into context and context-aware systems. However, it is important to note that there remains an apparent tension between two views of context. On the one hand, Dey and Abowd and Schilit view context as a flow of static elements which, when referred to in the past, maintain the same definition. Chalmers [14] has an alternative view that when contextual information is recorded and logged, the overall contextual interpretation changes with time because an essential and important part of the significance of current events is their relationship to past events. In this view, context is dynamic and historical rather than static in meaning.

Later work in this thesis, Section 2.2, utilises this second view of contextual information. By comparing currently sensed context with the history of the system's context information, this dynamic view is can be made manifest in a system implementation. The *George Square* system, produced as part of the work for this thesis, makes good use of the four context types identified by Dey and Abowd (*location*, *identity*, *activity* and *time*), whilst leveraging Chalmers's dynamic and historical view of context. In the George Square co-visiting system, historical context comparisons are used to automatically recommend relevant media content to tourists in a way that adapts to their movements.

An important but unexplored aspect of an application's context is the other software running or available at the time. If a system is to make the correct decisions and adaptations for its current environment then it must monitor not only the external environment but also itself; it must be aware of its current state and its history of use. By logging what software is being used with what other software and when, a good foundation is created for building systems to recommend new and potentially useful software to other users previously unaware of its existence.

Although the benefits of context-awareness have been written about and highly researched, there are few systems that are context aware and flexible at a deep structural level. There is a consensus in the literature that context-awareness is a significant aspect of mobile and ubiquitous computing systems, and has the potential to be a valuable source of information for suggesting adaptations in software. It was shown that context information from both the running system and the environment it is operating in may be useful. Most importantly for this thesis, an adaptive system based on context-awareness must monitor the external environment and also its internal structure. The Domino system introduced in Chapter 5 is one of the first examples of such a system.

2.2 George Square

The *George Square* system, named after the location it was trialled in, was a system that allowed city visitors to share their experiences with other visitors across the Internet. George Square is an example of 'collaborative ubicomp'—a system that supports users working together at a distance, sharing their interactions and information as one or more of them moves and changes context. Using a flexible peer-to-peer architecture, it supports the sharing of voice, photos, location and history between visitors. Through the use of a modular structure, the system allows for the dynamic and flexible reorganisation of system components. The project investigated the design of collaborative ubicomp systems that are flexible enough to work in unpredictable outdoor environments, yet rich enough to support successful and enjoyable collaboration.

Collaborative ubicomp systems have the potential to support valuable shared experiences between users, however this requires more complex infrastructures than are traditionally found in ubicomp systems. Distributed systems involving wirelessly connected devices and sensors present a range of technical and usability problems, which have been investigated in early mobile computing. However, less attention has been devoted to distributions of *people* and rich interaction between them through the medium of a ubicomp system.

George Square provides collaborative resources to users to support their interaction during their tourist visit. Distant visitors, each using a tablet or desktop PC, are connected together. Mobile users are assumed to use tablet PCs. Each mobile user's location is tracked using GPS and displayed on a shared map, whilst each non-mobile user can move an avatar around the map by clicking. Users can share photographs taken with attached web cameras, and can also talk to each other using a headset. Each user has access to a web browser to look up web pages of information during the visit. Each user's activity is recorded and compared to logs of past users. This drives a subsystem that recommends places to visit, web pages and photos.

George Square has a sophisticated modular peer-to-peer architecture that uses a distributed tuple space to supported dynamic reconfiguration as it runs. Multiple processes share sensor data, user activities and historical data in a location–transparent way through a distributed tuple-space, allowing the system to be reconfigured to new contexts dynamically. This architecture is suited to collaborative ubicomp using multiple mobile devices, as well as supporting more advanced information models that traditionally used in ubicomp, although it did have some limitations as described below.

Primarily, Matthew Chalmers, Barry Brown, Ian MacColl and Marek Bell designed *George Square* at Glasgow University. Ian MacColl and Marek Bell implemented the majority of the code for the system and I was involved at a later stage in preparing the system for trial. That required the implementation of a better map and improvements to various other parts of the user interface.

2.2.1 Motivation: supporting collaborative ubicomp

Collaboration is an implicit part of most ubicomp systems, in that multiple users often co-ordinate their activity around a system even if that system is designed for a single user's explicit interaction. Face-to-face collaboration has been an important topic since the early days of ubicomp, for example in the DigitalDesk research where users shared a single display surface and discussed and arranged documents [15]. However there are also opportunities for exploring how ubicomp could better support collaboration between users at a distance, and how technologies could support collaboration involving contexts that span objects distant and near. Location awareness is one part of this, but more broadly the general context of users can be shared.

George Square was designed to overcome limitations of its predecessor—the *Lighthouse* shared museum-visiting system [16]. That system had similarities in its motivation for supporting collaboration. The *Lighthouse* allowed multiple users, each in a different location, to share a museum visit weaving together online information with a physical museum experience. A visitor in the museum held a PDA whilst tracked by an ultrasonic positioning system, and collaborated with online visitors who were navigating a web-based or VR-based version of the same museum. The system supported collaboration around both physical exhibits and digital information in the form of web pages, images and videos. The experience was synchronous in that users were paired up and carried out their visit at the same time.

The user trials of the *Lighthouse* explored how visitors could weave together digital information with the more traditional exhibits available in the museum. Users talked and interacted amongst exhibits in the museum that had both a physical presence in the museum and an online presence in the system in such a smooth way that led to the concept of 'hybrid objects'. This synergy in the interaction was a key finding, and highlighted the successful collaboration around both places and digital information.

While successful in its aim of creating a shared visit among visitors using different resources in different places, the *Lighthouse* system's design lacked flexibility. The system supported only voice communication and location sharing between users; there was little sharing of contextual information beyond that. The system was fixed to one exhibition room, and was reliant on fixed infrastructure, for example the ultrasonic positioning system, and thus lacked scalability.

Another significant problem with the *Lighthouse* was that it required a considerable and extensive amount of content to be authored about the location it was to run at. The visitors were unable to produce their own content, instead consuming only existing static pre-authored content. Straightforward rules triggered actions based on current activity, such as displaying a web page when a user came near an exhibit. However, collaborative ubicomp systems can be more powerful if they also make use of past journeys, experiences and activities. This requires complex context logging of patterns of use and information models. For example, systems might build up and share histories of activity data, and relate ongoing activity to that history. Thus dynamic content generation can be driven by usage within the community of users. This is a more adaptive model of information presentation, generating and collecting based on previous and current activity, supporting uniquely tailored experiences, and is exactly what *George Square* was designed to explore.

2.2.2 System overview

George Square allows tourists to share their visiting experience with other people near by or remote. The tourist is given a mobile hardware based setup consisting of a WiFi Tablet PC that has a stylus for touch screen input and has a USB 'stalk' web cam and a Compact Flash GPS attached (
Figure 1). He or she also wears headphones that are connected to the Tablet PC that has a microphone built in. Remote users' hardware requirements are reduced since there is not any mobility requirement for them, so they could use the same tablet system but could alternatively be at a typical home desktop PC, with mouse and keyboard, webcam and headset.

Figure 1: *George Square* tourist with the tablet PC and headset.

The system provides several features involving location, information, communication and media to assist the user through the city. These features are designed to enrich the visit and enable the visitor to closely share his or her experience with others travelling with them or with remote users.

The *George Square* user interface (Figure 2) looks the same for both the visitor mobile in the square and remote 'online visitors'. It displays a large map with icons representing locations of interest, web pages, and photographs. GPS is used to show an icon representing the mobile visitor's physical location in the square and provides navigation support. It is also possible for users to override GPS and move their icons by tapping the map. Mobile visitors might do this when GPS is showing the wrong location, which frequently happens when buildings obscure the sky. Remote users have location markers too, and they simulate movement also by clicking the map. Maps are automatically downloaded over the Internet from a map server, allowing the system to be run anywhere with map data available.

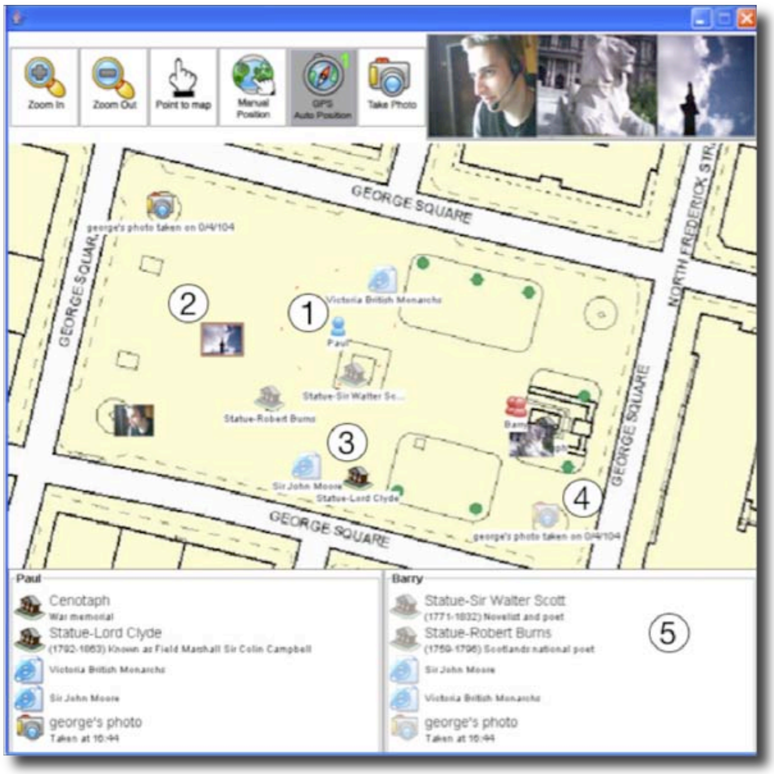


Figure 2: The *George Square* user interface showing user's location (1), photographs (2), locations (3), photograph links (4) and recommendation lists (5).

The “Take Photo” button allows visitors to take photographs using the attached USB camera, and the photographs are automatically geo-referenced using the GPS. A thumbnail of the photo appears in the corresponding location on the map. Clicking on the thumbnail on the map opens the full-size image. Photographs are shared with co-visitors, so that those in remote locations are able to view and discuss the images as they appear. The ‘filmstrip’ in the top right of the interface shows the last three images taken by and all visitors, which conveniently let visitors, see the images taken by others. It animates left to right as new photos appear, and clicking the images opens the full-size image.

Another major part of the system is recommendations. *George Square* contains a variety of information: locations which might be worth visiting, web pages that may be relevant to items nearby, and photographs for pictures taken by previous visitors. A content filtering algorithm, Recer [17], is used to filter the information so only the most relevant to the user’s current context is displayed in the form of recommendations. The recommendations are for web pages, photos and locations, and appear as icons on the map (Figure 3) and in a list at the bottom of the interface. Clicking a recommendation will, respectively, open the relevant web page or photograph, or highlight the location on the map. One type of contextual information used by Recer is location. The location of the user’s icon, generated either by clicking the map or from GPS, allows Recer to recommend nearby points of interest, and thus as he or she walks around icons will appear on the map close to where he or she is. In addition to one’s own recommendations, one sees the recommendations currently being shown to other visitors as ghosted icons on the map and in the list. In *George Square's* predecessor, *Lighthouse*, it was found that by displaying what others see supports a smoother discussion and understanding between concurrent users of such systems. For example, in *George Square,* it is typical for one user to suggest something like, "let's try my third recommendation" and by glancing at the other's list, one can easily follow what is being referred to.

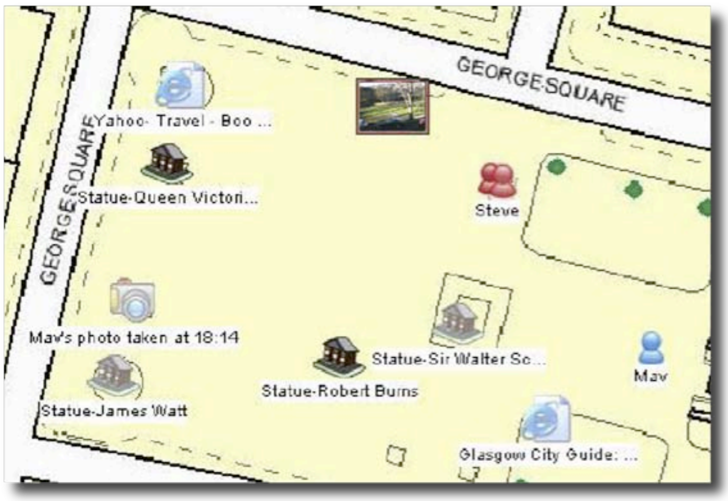


Figure 3: The *George Square* map showing recommended places, photos and webpages. One's own recommendations are shown in full colour while one's co-visitor's are shown 'ghosted'.

Logging plays an important role in *George Square* because the recommendations mentioned above are not pre-authored: they are generated from the logs of previous users' activities. The path the visitor walks, the webpages viewed and photographs taken are all logged by the system. Recommendations for a user currently visiting are generated from the histories that all the previous users of the system built up during their visits to the area. This dynamic generation of content allows the recommendations to remain relevant, and adapt as the city or events in the city change.

Visitors can talk to each other in real time using their headset. We used a VOIP application that utilises the WiFi connection. VOIP has a relatively low bandwidth requirement (<15K/s) and so even though the lowest WiFi spec of 802.11b was used, call quality was excellent even at maximum range.

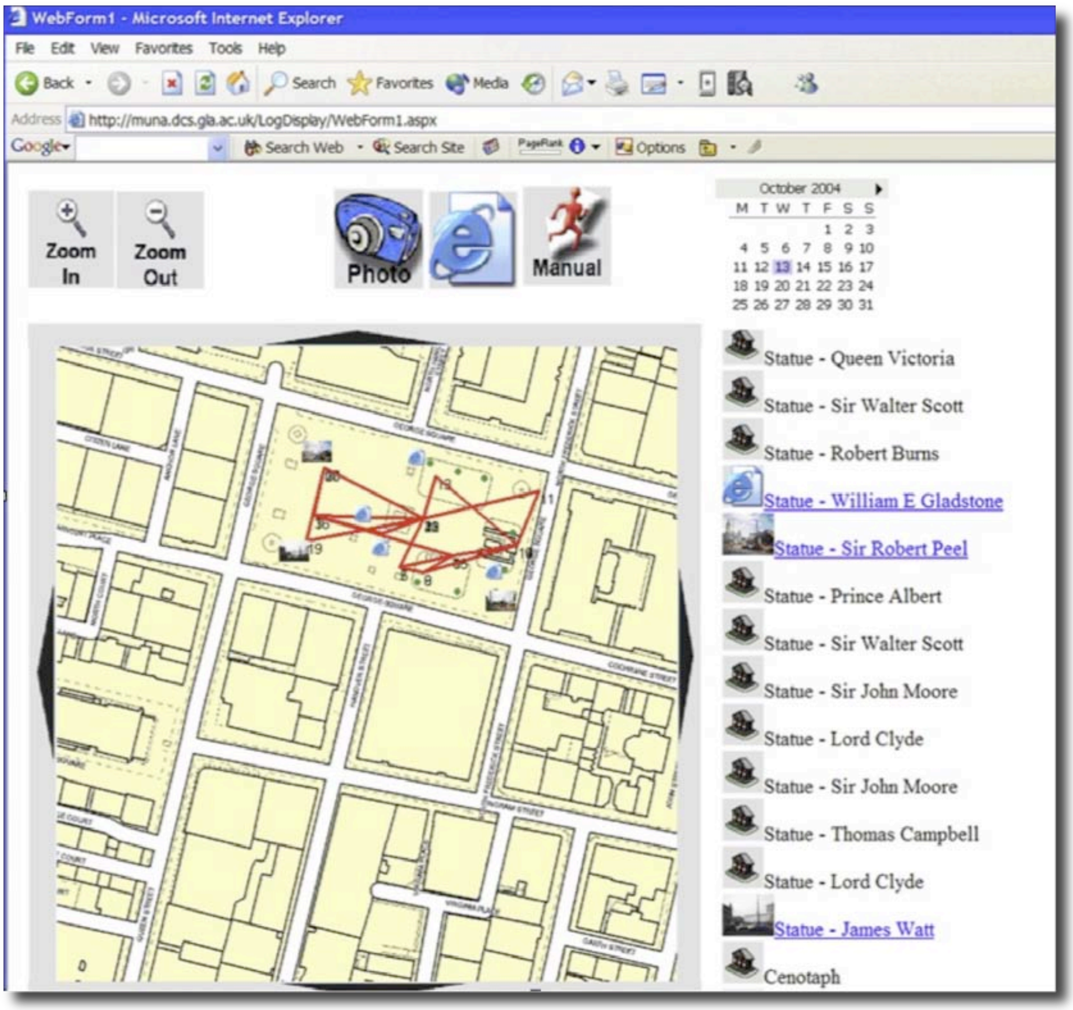

WebForm1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://muna.dcs.gla.ac.uk/LogDisplay/WebForm1.aspx
Google Search Web Search Site Options
Zoom In
Zoom Out
Photo
Manual
October 2004
Statue - Queen Victoria
Statue - Sir Walter Scott
Statue - Robert Burns
Statue - William E Gladstone
Statue - Sir Robert Peel
Statue - Prince Albert
Statue - Sir Walter Scott
Statue - Sir John Moore
Statue - Lord Clyde
Statue - Sir John Moore
Statue - Thomas Campbell
Statue - Lord Clyde
Statue - James Watt
Cenotaph

Figure 4: *George Square* blog interface.

The database logs generated from earlier visiting are used to generate a post-visit ‘travel weblog’ — a web page displaying a summary of each visit (Figure 4). The pages are available on the Internet to the visitor at any time. The pages display a temporally–ordered list of all the photos, web pages and places that he/she visited, and offer a similar map to the one used during the visit. The path he/she walked is summarised by a spatial representation: a red line.

2.2.3 Recer

Recommendations for web pages, photos and locations were generated in George Square using Recer [18]. Recer is a simple yet powerful collaborative filtering algorithm designed to be very flexible in the type of data it uses. It works by monitoring and logging the user’s actions and almost any kind of event can be recorded. Some possibilities include, for example, accessing a file, visiting a URL for a web page in a web browser, sending an instant message, or the title of the application brought to the front of a GUI. Each action is logged as a plain text entry in a database, a user id, and a timestamp for when it occurred, and the table is indexed by time. Later versions of Recer added another plain text entry for the type of action, for efficiency. Over time, the entries create a chronological representation of the user's activity, which may be thought of as one unique history among the unlimited number of possible variations among users' logs. The recommendation algorithm works by comparing groups of recent entries in one's log, with all past windows of time within the database of logs.

Recommendations for a user can be based on history from that single user, however in this situation, no novel recommendations will be generated. The real power of the recommendation algorithm is when the histories of all users are combined, and the recent entries of a single user are compared to the histories of all others. In this situation, interesting and novel recommendations can be made to the user, and the steps taken to achieve this are explained next.

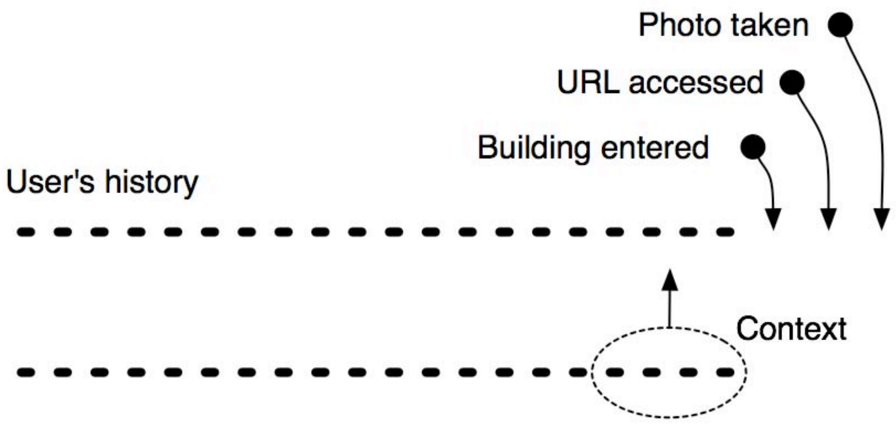


Figure 5: The Recer algorithm is used to identify context from the items logged in the user's history (shown here as a dotted line), such as URLs and locations. The user's context is modelled and extracted as a set of the most recent items.

Firstly, the recommendation algorithm requires the extraction of the most recent subset of entries from the user's own history. Recently logged entries are assumed to be those relevant to the user's current activity; this forms the user's *context* (Figure 5). Next, each individual item in the context is taken, and in turn, the combined histories of all users are searched for instances of the individual item. Many matches may be found but, for each matching item, the time of its logging is examined and two further timestamps are calculated representing a short time in the future and the past of when the item was logged; this forms a selection *window.* The items found within each such window are extracted and grouped with the results from other windows, then ranked in order of number of occurrences (Figure 6). The items ranked highest are the final recommendations.

Matching in this context-specific way distinguishes the collaborative filtering algorithm from most others, which tend to match people on the basis of all the data logged for each user rather than attempting to concentrate on specific windows of history data that are most likely to relevant.

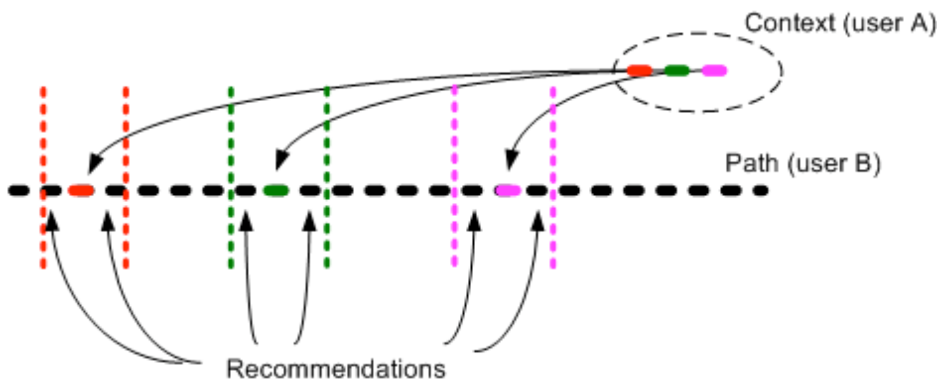


Figure 6: The Recer algorithm generates recommendations by selecting items from a user's (user A) context are identifying them in another's (user B) history log. Items adjacent in time to these are then extracted, tallied and ranked as recommendations for user A.

Versions of the Recer algorithm were implemented by Matthew Chalmers in Java, Aran Lunzer in Visual Basic, and by Marek Bell in C# and PHP (to allow it to be used on websites). The Java and PHP versions run only on desktop (or laptop) machines running Windows, Linux or OS X. However, the C# version is implemented in the .NET Compact Framework and can run on both mobile devices with Windows Mobile and desktop Windows machines. When running on desktop machines, Recer can use Oracle, Microsoft SQL Server, Postgres or mySQL as its database back end, and on mobiles it was designed for Microsoft SQL Server CE. However in testing, this mobile version did not perform adequately and the bottleneck was found to be in the database and communication between it and the application, rather than solely because of the constrained resources on the device.

To overcome this, the author created an optimised implementation of Recer for a mobile device using the SQLite database that overcame the deficiencies and performed well, and was used in the projects described later in this thesis. The length of the context, size of the windows and number of recommendations returned can be set using the API in order to adjust the algorithm for a range of circumstances. However, the default API values of five minutes for context and two-minute windows (one minute on either side of a located item) generally provide good results, and these values have been used in the user trials of all the systems implemented so far.

In Recer, like all collaborative filtering algorithms, the quality of the recommendations depends on the quality of information logged from other users. Recer depends on the history information from other users, and in order to improve relevance and reduce randomness, a method of spreading this information in an epidemic manner within a peer-to-peer community with no reliance on central servers. Peer-to-peer Recer is described in Section 4.3.1 and a system utilising it to recommend places for tourists to go, *Samara,* is described in Section 4.3.2.

2.2.4 Modular design using Equip

The implementation challenges for George Square were typical of collaborative mobile systems, in that the system needed a dynamically changing set of devices to work together as peers, without continuous reliance on a central server, while supporting users and devices joining or leaving at any time.

George Square used a modified version of the Equip distributed tuple space to support the storage and sharing of data within the mobile environment it ran in [19]. Equip features a peer-to-peer communication model between networks of sensors and devices via stores (or 'dataspaces') of records (or 'tuples'). Equip allows applications to subscribe to a dataspace and be notified of specified events, similar to the popular Elvin system [20]. Applications place events into a dataspace, which Equip uses to trigger methods in other applications that have subscribed as listeners to those events. Thus, if one peer has subscribed to receive another's GPS events, it will be continually notified as the other peer generates and stores GPS data as its user moves around. Equip is essentially a transport mechanism, which supports transferring data between both system components and remote devices.

Previous Equip-based systems, for example the *Lighthouse* system described earlier in Section 2.2.1, used a single dataspace running on a single server. However, dataspaces can potentially be distributed across different machines, with events triggered between devices across networks. Utilising a peer-to-peer architecture, each individual Equip space can be used on any device without reliance on a central server. The event–based architecture allows devices and users to leave or join an existing network of users at any time, with dynamic and automatic reconfiguration. Special processes called *PeerProbes* selectively listen for events and data in other dataspaces connected over the network. When events are discovered they are copied into the local Equip dataspace and this effectively produces a distributed tuple space peer-to-peer system.

During the implementation of George Square, three different classes of component for processing and displaying events were created: *probes*, *gizmos* and *services*. Communication between them was moderated through Equip. A *probe* is like an input sensor, for example the GPS or camera. A *gizmo* is able to register an interest in *probe* events and extract the information, for example output it to the screen. Hence, a 'photo gizmo' can extract photographs that a 'camera probe' had put into the dataspace, and display them on the user interface, for example on *George Square's* map.

Since all network communication takes place through Equip, including the discovery of peers, the architecture is modular and services can run in a peer–to–peer configuration on any device or set of devices. However, in some cases it may only be suitable to run a certain service on one particular device. For example, a computationally expensive service requiring more resources only a powerful device can offer, such as an extremely large database store. In *George Square*, the database service that logs history events to a database and ran the Recer algorithm generating recommendations ran on a single machine. Since services operate through the distributed dataspace, machines can change service roles while the system is running, allowing for dynamic reconfiguration of any setup at runtime.

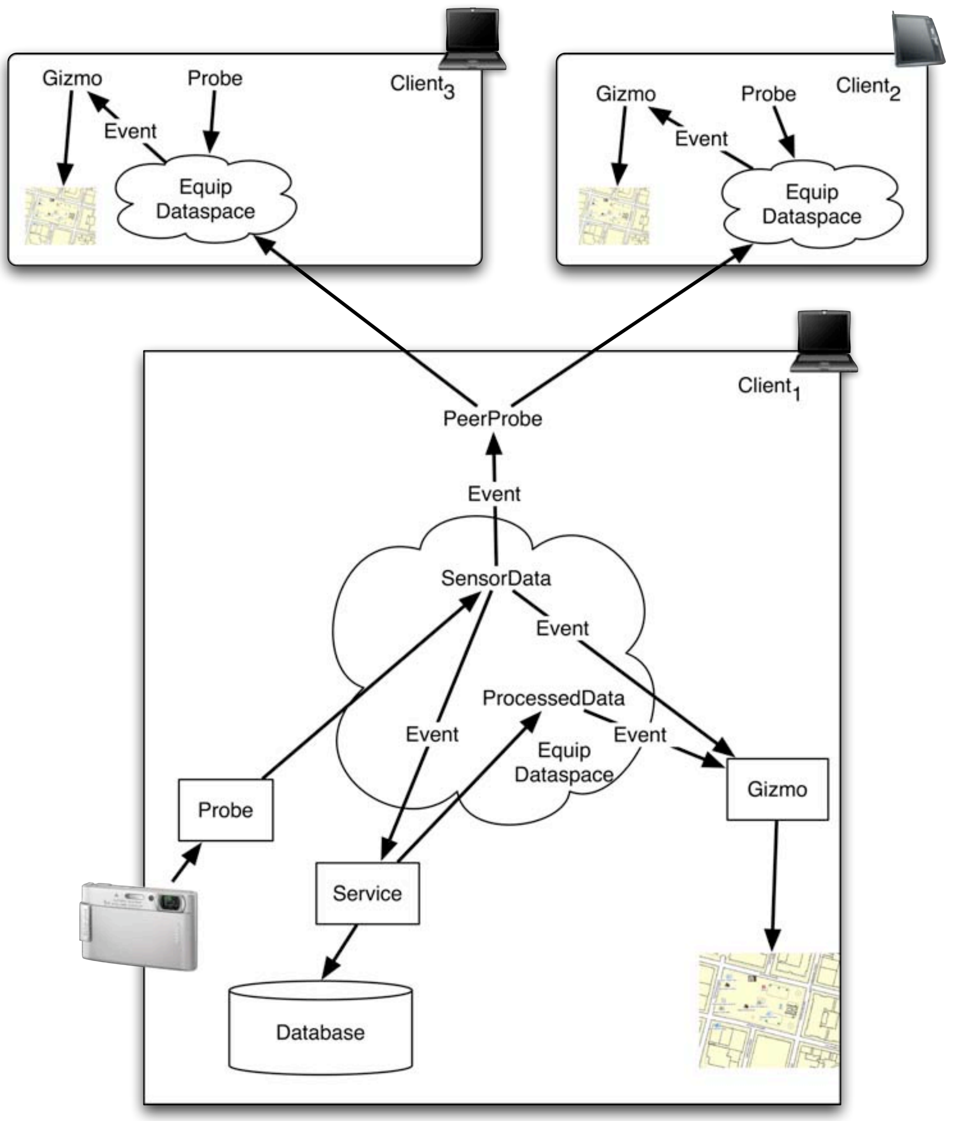


Figure 7: Diagram of *George Square's* Equip-based infrastructure.

As shown in Figure 7, in George Square a set of probes on each user's machine collected sensor data, such as GPS and photographs from the camera, and placed these items into the local dataspace. The PeerProbes would copy these events across the network to other machines. The gizmos running on each machine were then triggered by this information and would display this information to the local user, for example, GPS co-ordinates of his or her own location and other users' positions displayed on a map.

Lastly, the system used three services that carried out further processing on system events. The FocusAdapter service took GPS events and matched GPS co-ordinates to FocusPoints: discretely named polygonal extents on the map. The FocusAdapter thus mapped from continuous coordinates to discrete logical locations (sometimes termed 'semantic locations'), and then injected FocusEvents back into Equip where they triggered a second service, RecerService. RecerService listened for FocusEvents, which were then added to users recent history, and used to generate recommendations from the database of all past histories.

In George Square, Recer was used to deliver photograph, location and web recommendations to tourists visiting the Square. As explained above, the Recer collaborative filtering algorithm works by matching a user's recent activity with similar past periods of activity, and then draws recommendations from these periods. In George Square, a RecerService was responsible for recommending to each visitor a number of web pages, places and photographs, drawn from a database of past visits. Another service, RecerDB, running on each client device was responsible for logging user activity as they explored the area, logging the photographs, locations and web pages each visitor investigated. RecerDB then sent this information to the RecerService, which stored it in its database as required. To preserve the serverless architecture, all Recer services can in actuality run on each local machine, with the database duplicated on each device. Thus, if users moved away from the network and other devices, recommendations could still be delivered to them using the pre-cached information on their device. Recer was previously discussed in Section 2.2.3 where an improved, decentralised version of the recommendation algorithm is used in *Samara.*

2.2.5 Flexible content generation

The *Lighthouse* and GUIDE systems both depended on static, pre-authored content. The Lighthouse museum co-visiting system, mentioned earlier (Section 2.2.1), relied on a large amount of digital content to be entered into the system, to be viewed on a PDA by the user in the museum, and in the web browser of the remote visitor. Textual information had to be searched for, video clips had to be recorded, and photographs taken. The production and entry of this information was found to be one of the most time-consuming parts of building the system. Also, the static nature of the system's content limited its mobility. Mobile systems can, by definition, roam to any location, and it is impractical to produce content for every possible location. Static content systems require an initial committed effort from developers and may require continued maintenance whilst the system is deployed.

Avoidance of reliance on location-specific, pre-authored content can be achieved through the selection of self-generating content type architectures, and George Square utilised this extremely flexible approach.

In a self-generating content system information is entirely generated by the client applications themselves. This information may then be shared to the community, either after being uploaded to a server that peers then read from, or spread directly through the community utilising peer-to-peer, ad hoc connections. For George Square, recommendations are generated from logs of the system itself simply being used and, as the community of users increases, the amount of logged data, and therefore quality of the recommendations, grows. A key feature of a self-generating content system is that it is not necessary for any data to be entered before the system is first run, as it will build the data up as the system is used. Self-generating content systems are probably the most appropriate for use in peer-to-peer mobile systems as they can be configured to require absolutely no support, and connections to, any devices or servers outside the community.

A side effect of utilising user-generated content is the lack of recommended content for the initial use of the system. To overcome this ‘boot-strapping’ problem in George Square, the first use of the system was made by professionals from the near by tourist information office. They used the system as normal and not in a special content authoring mode, the only thing lacking was recommendations. Thus, the George Square system allows both user-generated content and official content to be automatically gathered without requiring input from a designer or specific content author.

Another issue is that the amount of user created content can be unmanageable, and when spread peer-to-peer it is outside local control. This might not be desirable in certain circumstances such as critical applications. An advantage of the recommendation system is it can alleviate this issue, as lower quality content should be viewed less and resulting in lower rankings.

Self-generating systems relieve the entire weight of authoring content from the developer. They can also provide substantial value to the end-user community as self-generated data can be continually updated at no further cost thus providing a system that always stays relevant.

2.3 Conclusion

This chapter has investigated existing context awareness research relevant to the work in this thesis. In particular, it is clear that context awareness is of critical importance to the type of mobile systems proposed throughout this thesis.

It is apparent that making a system aware of its context—the external environment around it, such as peers and other devices; as well as any internal context such as files or software configuration on the device itself—may greatly increase its functionality, usability and appropriate behaviour. Systems such as Jabberwocky—with its ability to highlight ‘familiar stranger’ relationships—and George Square would simply not be possible if they were not able to continually sense, interpret and make use of contextual information.

The George Square system proved an excellent test bed to examine how an advanced context aware system may operate. The findings from the George Square system are concrete demonstrations of an answer to RQ1. George Square demonstrated how the past could be used as a resource to generate relevant information from recommendations, based on comparing the current user’s context to the histories of past activity and context generated by previous users. Furthermore, by sensing an adequate amount of context, George Square is able to appropriately filter the data that is delivered to the user, avoiding information overload. George Square also made use of contextual information to connect or associate complex or heterogeneous types of data; in this case, matching online content to the contextual information on specific places in the city.

Adaptation was also explored in *George Square,* both in terms of the content authored in the course of system use—or, more precisely, associations created in the course of system use among items of mostly pre–existing publicly available content—and a reconfigurable modular architecture. However, the architecture of the George Square software infrastructure was limited by its reliance on central servers. From its trials, several key design features for mobile, context aware, adaptive systems may be highlighted. Firstly, it is clear that authoring content to drive a large mobile system can be costly. For example, the Lighthouse system relied on a wealth of content that had to be selected and edited by museum experts, and meticulously input into databases and files in the system. Whilst the Lighthouse system was available in only one museum room, George Square can be run anywhere, and thus the information it may deliver is far greater. Despite this, George Square requires virtually no cost in order to deliver its content. By utilising adaptive dynamic content George Square supports the gathering of information by users' own use of the system. Through sensing a user's actions and their context, George Square is able to autonomously generate content that it can later deliver to others. Thus, one key design feature that should be adhered to when creating mobile, context-aware adaptive systems may be:

- Avoid reliance on content (and associations among it) pre-authored by system designers by using content and associations that adapt dynamically with use.

Secondly, George Square highlights the importance of the peer-to-peer community mobile systems can operate within. If George Square devices and users were isolated from one another the system would quickly stagnate; it would be unable to generate recommendations for anything other than that for items that the current user had already viewed, or actions that they have already conducted. It is only by sharing information between a community of users that many mobile systems can gain a critical mass of data to make them useful and interesting. As this collection and sharing of information is so vital, it is apt to make use of not only information that is currently available (such as the current peers in range), but also to record and store past information, and to deliver it to others for consumption. This was indeed the case in George Square, were historical logs of data formed a critical part of each devices core. The realisation of the importance of collecting and sharing data leads to another vital design feature of such systems:

- Information should be shared among and stored by multiple peers, to offer efficiency and robustness.

Finally, as stated George Square suffered several disadvantages due to relying on a centralised architecture. As users moved around during their normal use of the system the 802.11 connections relied upon to make the connection to the central server would often break or suffer from interference. When this occurred the system would be unable to continue generate novel recommendations or receive updates about other users locations and photographs until the connection to the server was available once more.

This greatly limited the mobility of users who, after realising the disadvantages of being disconnected, were hesitant to leave the relatively small area in which they knew they were likely to achieve a reliable connection. Thus, a final design recommendation can be made:

- Centralised peer-to-peer architectures should be avoided whenever possible, primarily for reasons of efficiency and robustness.

As will be discussed in Chapter 4, decentralisation can be achieved by employing a greater use of peer-to-peer and MANET techniques. Furthermore, negative affects experienced due to limits in 802.11 connections and other necessary mobile infrastructure can be alleviated through the appropriate application of seamful design. This is discussed in detail in the following chapter.

Chapter 3 Seamful Design

This chapter describes a new area of research in ubicomp called Seamful Design. Its origins are in the work of Mark Weiser, a key figure in the creation of the idea of ubiquitous computing. The Equator group at the University of Glasgow has explored this new research area during the time of this PhD. This chapter presents the background work involving Seamful Design, and two systems designed to demonstrate the design technique. The chapter demonstrates how applying the concept of Seamful Design to an adaptive system can have very positive benefits.

3.1 Background

As envisioned by Mark Weiser, a design goal of ubicomp is invisibility [21]:

> *A good tool is an invisible tool. By invisible, I mean that the tool does not intrude on your consciousness; you focus on the task, not the tool.*

Tools in ubicomp can be multiple heterogeneous devices varying in sensing and output techniques. Weiser believed focus should be on the overall result of interaction, rather than the individual tools used to achieve it.

The need for tool invisibility does not necessarily translate into seamless integration between tools, which Weiser saw as a misguided concept. For example Mainwaring et al. state:

Ubiquitous computing (ubicomp) is a vision of infrastructure. Indeed, it is a vision of multiple infrastructures – some new, some existing; some virtual, some physical; some technical, some social – all coming together in a seamless way.

Weiser suggested a drawback of making things seamless is sacrificing their individual character and direction, in order to promote smooth compatibility. This loss of richness in the integrated tools led him to an alternate view, a *seamful* approach with “beautiful seams”. Inspired by this, seams are defined by Chalmers [22] as:

a break, gap or ‘loss in translation’ in a number of tools or media, designed for use together as a uniformly and unproblematically experienced whole. Seams often appear when we use different digital systems together, or use a digital system along with the other older media that make up our everyday environment.

Seamfully integrated tools would retain their individual characteristics. We would let the tools be ‘themselves’, making their features apparent in our overall interaction with them. Weiser continued:

The unit of design should be social people, in their environment, plus your device

This is embraced by current ubicomp systems, in that devices sense and model the context of people and objects in the environment. Taking into account the physical characteristics of devices that build up the system led Chalmers to suggest a novel realisation [23]:

Letting a ubicomp system be itself means accepting all its physical and computational characteristics—that may either be weaknesses or strengths. A user’s activity is influenced by what they perceive and understand of sensors, transducers and other I/O devices, and the system’s internal models and infrastructure.

Taking mobile phones as an example, signal strength to the cell tower is an inherent property of the infrastructure that makes up the overall system, and is sensed and shown on the phone’s user interface. Its display has many uses, for example to know when and where calls or data connections are possible, perhaps mediating frustration, but if interruptions do occur it provides a mechanism for comprehending why—which may provide some alleviation of frustration, as noted by Chalmers et al. [23]:

This is an elegant ambient or peripheral presentation of potentially useful information that is characteristic of the phone as a physical sensor and the phone network as a cell structure.

Digital systems all have limitations, for example GPS depends on a clear view of the sky and works best in countries closer to the equator where satellite coverage is optimal. Digital cameras are limited by resolution and the light sensitivity of their imaging sensor. This is particularly relevant in the mobile environment, as mobile devices frequently utilise many sensors, and often rely on wireless communications, either in peer-to-peer or infrastructure mode connectivity. This leads to a vulnerability particular to mobile systems, and can increase chance of failures and the breakdown of the ideal of seamlessness. Users may become frustrated or confused if a suitable level of information about the underlying technology is not exposed.

These physical limits of technology usually lead to defensive design tactics:

> *As designers, we can be defensive or negative about such seams in devices and infrastructure, and try to design them out.*

Chalmers explains this approach is not always practical, affordable or possible — systems are always limited in certain aspects by the fact the resources available to build them are finite. The reality is that seamless perfection is rarely if ever attained in practice.

A complementary outlook on the characteristics of digital tools and media involves embracing their limits and variations, and taking a pragmatic or even positive view rather than succumbing to their negative limitations.

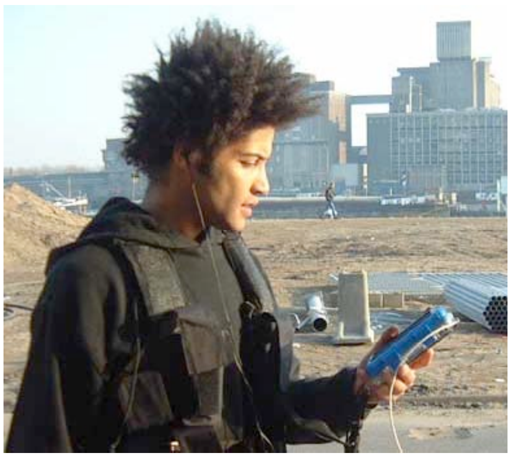
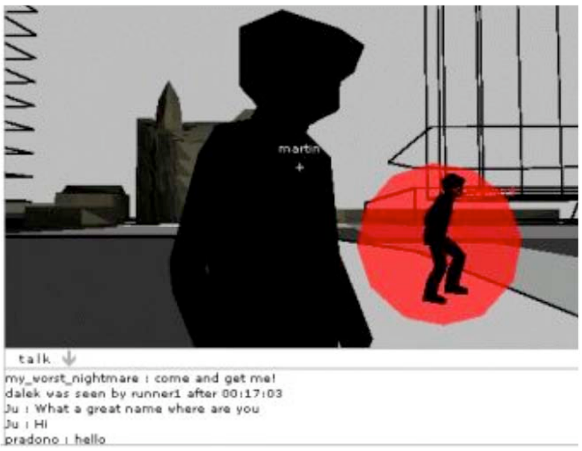


Figure 8: A 'runner' playing *Can You See Me Now* holding a PDA.



my_vorst_nightmare : come and get me!
dalek vas seen by runner1 after 00:17:03
Ju : What a great name where are you
Ju : Hi
pradono : hello

Figure 9: The online players view of the *Can You See Me Now* game – the virtual representation of Figure 8.

An example of how a ubicomp infrastructure limitation can be used positively by users was observed during trials of the location-aware multi-player game *Can You See Me Now* [24]. The game consists of two types of player: runners (Figure 8), who are physically present on the streets of a city and carry PDAs with GPS, and online players using a web interface to control an avatar in a virtual model of the city (Figure 9). The game is similar to *tag*: the runners must chase and capture the online players by moving their slow-moving avatars around the map. The runners' real GPS locations are used to project avatars into the virtual model being viewed by the web players.

A limitation in this game was the accuracy of the GPS. Whilst variation in GPS accuracy is often thought to a problem, the runners in *Can You See Me Now* found ways to accommodate or even take advantage of it [24]:

> *Over the course of the two days play the runners became increasingly aware of the effects of GPS inaccuracy and also where on the city streets it was most likely to be experienced. By the second day's play, they had begun to exploit this knowledge as part of their tactics…*

At the end of the first day's play, the runners had learned the areas where GPS worked well. That evening, they developed a strategy: to wait for online players to enter them, or even to trick them into entering them. Runners could hide nearby, in areas of little or no GPS accuracy, and then move quickly out to catch players. This led to more successful catches in the areas where GPS worked well. It is an interesting aspect of mobile systems that often users employ unexpected methods when using them, and this case is no exception. Whilst knowledge of the GPS infrastructure and its failings was advantageous to the runners in *Can You See Me Now*, lack of this knowledge set online players at a significant disadvantage – as is evident from the drop in success rates from the first day to the second. Flintham et al. pose an interesting question [24]:

> *As we have seen, this ended up placing online players at a disadvantage as runners in the street exploited this difference. This raises the question as to whether the technology should have made them more aware of the characteristics of GPS?*

If the online players had knowledge of GPS inaccuracies, it is likely they would have been able to better predict how the runners' avatars might move, as well as identifying likely areas on the map which had almost no GPS coverage and so were possible 'hiding spots' for runners.

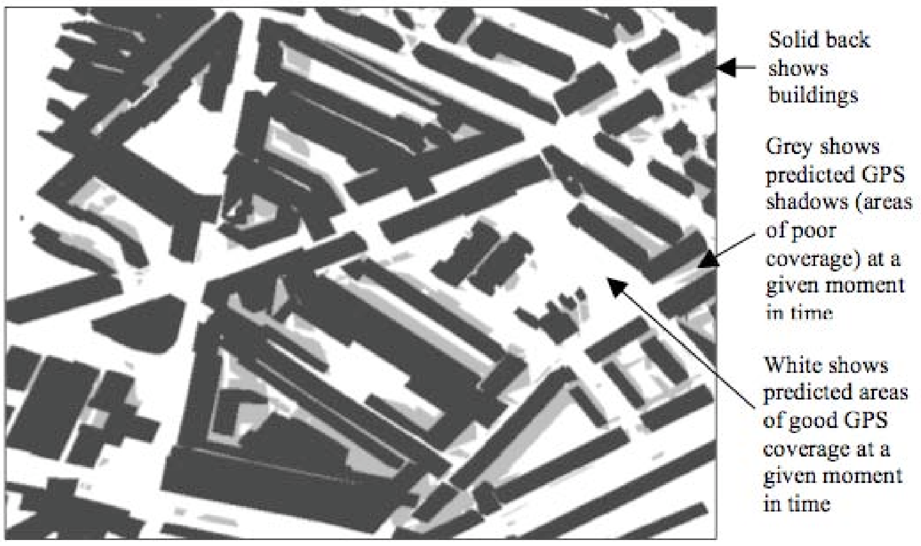

Solid back shows buildings
Grey shows predicted GPS shadows (areas of poor coverage) at a given moment in time
White shows predicted areas of good GPS coverage at a given moment in time

Figure 10: Visualisation of GPS shadows conveying availability [24].

It has been found that GPS coverage can change throughout the day, due to the shadows cast by buildings moving due to satellite movements (Figure 10) and the authors of the game experimented with how this information could be visualised and exploited:

> *The purpose of these visualizations is to help deal with uncertainty by revealing the gaps and breaks – or the 'seams' – in game play and to make them available as resources that the runners and players might exploit to make sense of the technical circumstances effecting interaction.*

The concept of deliberately revealing information about physical and software infrastructure is a powerful one, particularly in the mobile environment, that can be exploited to allow users to overcome or take advantage of problems or failings when they occur. Similar benefits, realised by Gaver et al., were found when approaching ambiguity from a similar angle [25]:

> *Ambiguity can be frustrating, to be sure. But it can also be intriguing, mysterious, and delightful. By impelling people to interpret situations for themselves, it encourages them to start grappling conceptually with systems and their contexts, and thus to establish deeper and more personal relations with the meanings offered by those systems.*

Although Gaver et al. are concerned with ambiguity, they make an important point related to *seamful design*—that systems can be designed to actively encourage users to better understand the concepts and infrastructure behind them and thus, in turn, to better understand how they come together to provide a working system. Flintham et al. and Gaver et al. theorise that users can be trusted to understand and appropriate information about infrastructure to allow them to overcome breakdowns in novel or known ways.

In summary, seamful design is about exploiting characteristics that are apparent in use and interaction, and which contribute to users' practical understanding and use of a system as they experience it in their everyday lives. Users and designers can understand the system better and therefore find ways to use it that fit with or exploit how it really works.

3.2 Seamful games

To reiterate, Seamful Design is concerned with exploiting characteristics of the underlying software and hardware infrastructure that are not made readily apparent by the designer to the user in traditional systems. This can permit users greater practical understanding of the system and enrich their interaction with a system as they experience it in their everyday lives. By employing seamful design, designers can recognise seams in software and hardware, and expose them where appropriate to help users better understand a system's behaviour and thus better integrate a system into their lives and everyday behaviour.

At Glasgow, we have developed systems and infrastructure to explore this concept, initially concentrating on 802.11 networks and the seams inherent in wireless technologies. For our first explorations, a multiple access point WiFi network was set up around the university campus, and a specialised WiFi driver that exposed more information and more control in Windows mobile devices than available by default was developed along with GPS and mapping software. Taken together, these tools formed a Seamful Design test-bed on which multiple systems could be implemented and tested.

This section describes two games, developed at Glasgow, that exploit Seamful Design concepts and ideology. The first, *Treasure,* reveals the underlying WiFi infrastructure to the players through an enjoyable short game in a fixed geographic area. The second, *Feeding Yoshi,* differs in several ways. It is designed to fit into users' everyday lives and be played at many different times and for periods of play. Thus, it is played over a long time—weeks or months—and can be played anywhere. Thus Feeding Yoshi additionally supported an exploration of how to tackle issues such as varying location and time constraints in people's daily lives, and an investigation of how this impacts on the feel of play.

3.2.1 Treasure

Marek Bell, Matthew Chalmers and I designed the idea and game concept behind *Treasure*. Marek Bell and I did the overwhelming majority of the implementation, although several undergraduate students did contribute small pieces of code to the game client as part of a summer research project. In contrast to the use of seams by users in Can You See Me Now, inadvertently made available by CYSMN's system designers, Treasure involved the deliberate use of seams by system designers. We designed the system so as to help make seams into a resource for users.

Treasure was primarily designed to explore seams in WiFi infrastructure, although GPS was also explored. WiFi seams may include, for example, patchy distribution of access points over a geographic area and highly variable signal strength even when inside WiFi network coverage. The aim of Treasure was to expose these limitations of the technology as a feature of the game, to examine if players could understand them and learn to take advantage of them as part of the fun of the game. In this section the game will be described, then the technology used and details of the implementation will be discussed. Finally, a report on the user trial is delivered.

The aim of the game is to collect *coins* scattered over an urban area and then get them in to the *treasure chest* (technically, a server) in order to score points. This concept of having apparently physically located objects viewable on digital devices is common to mobile games, and appears in games such as *Six in the City* [26], Benford et al.'s *Unearthing Virtual History* system [27], *PacManhattan*[3] *and* Newt Games' *Mogi Mogi*[4]. Indeed, it is an example of mixed reality, which is common in many ubicomp systems, and is often referred to as 'augmented reality'. A similar concept is a hobby called *Geocaching*[5], an outdoor treasure-hunting game in which the participants use GPS or other navigational techniques to hide and seek containers (called 'geocaches' or 'caches') anywhere in the world.

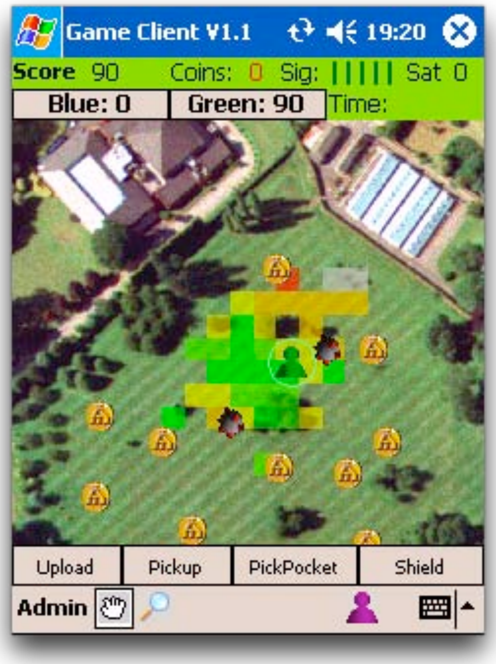


Figure 11: The Treasure user interface displaying a map, coins, mines, the player's location and a semi-transparent WiFi signal strength layer.

[3] http://www.pacmanhattan.com
[4] http://www.mogimogi.com
[5] http://www.geocaching.com

In Treasure*,* two teams of players compete against each other outdoors. Although Treasure can support any number of players in each team, the trials typically used two players in each team. The game is played for a set period of time (20 minutes in the trials), and a clock counts down on the client display to let users know how long they have remaining. The team with the most coins in their treasure chest at the end wins the game. The players carry a PDA that displays a map showing the location of the coins and players (Figure 11). The server randomly creates new coins at random locations within the geographic game area and these coins periodically appear on the map. Players must then physically move to the location of these coins to collect them. When a coin is nearby, i.e. within the five-metre circle surrounding the player's avatar, the *Pickup* button allows the coin to be collected. Coins are dropped periodically in all parts of the arena and a player can pick up coins at any time, as long as he or she is within sufficient range of them.

During the game, a player's location is retrieved using a GPS device attached to the PDA, and the coordinates derived from this are used to display their avatar on the map. The game map seen in the user interface has been geo-referenced so players' locations appear as close to their true physical locations as possible. Limitations in the accuracy of GPS often mean the location of the avatar is not exact. However, when the sky is clear and the GPS unit has a good fix, it is often accurate to within a few metres, and the avatar on the user's display allows for the clear and smooth interpretation of walking or running according to the user's movements.

Wireless network connectivity is a major feature of the game play in *Treasure*. Players' devices can only receive coin updates when they are connected to the wireless network and have a good signal, as this allows the server's broadcast packets including the coin updates and player positions to be received. Thus, users are only notified of new coins, and see the map updated with coins and player positions, when they have a reliable connection to the server in this manner. After coins are picked up they need to be carried back to a location in which there is wireless coverage. At this point the user can click the upload button to send their coins to their treasure chest and gain points. For this to happen reliably the users must again have a strong WiFi connection so that the upload packets reach the WiFi node, and subsequently the server. If the WiFi signal is not available, or is weak, there is a high change the upload packets may be dropped, for example, not reaching the server and being lost instead. Thus, users quickly become acutely aware that they must have a good WiFi signal to reliably upload coins, as a failed upload due to weak signal strength can result in a loss of many coins, and hence points. Also, they need to be aware when player locations are being updated live, to watch out for pickpockets. Thus, to play the game successfully, players must learn the advantages of being both in network coverage and being outside network coverage, and manage their time appropriately in each. By taking part in the game players come to learn the wireless network layout in the area, its characteristics, and discover how it can be both advantageous and problematic.

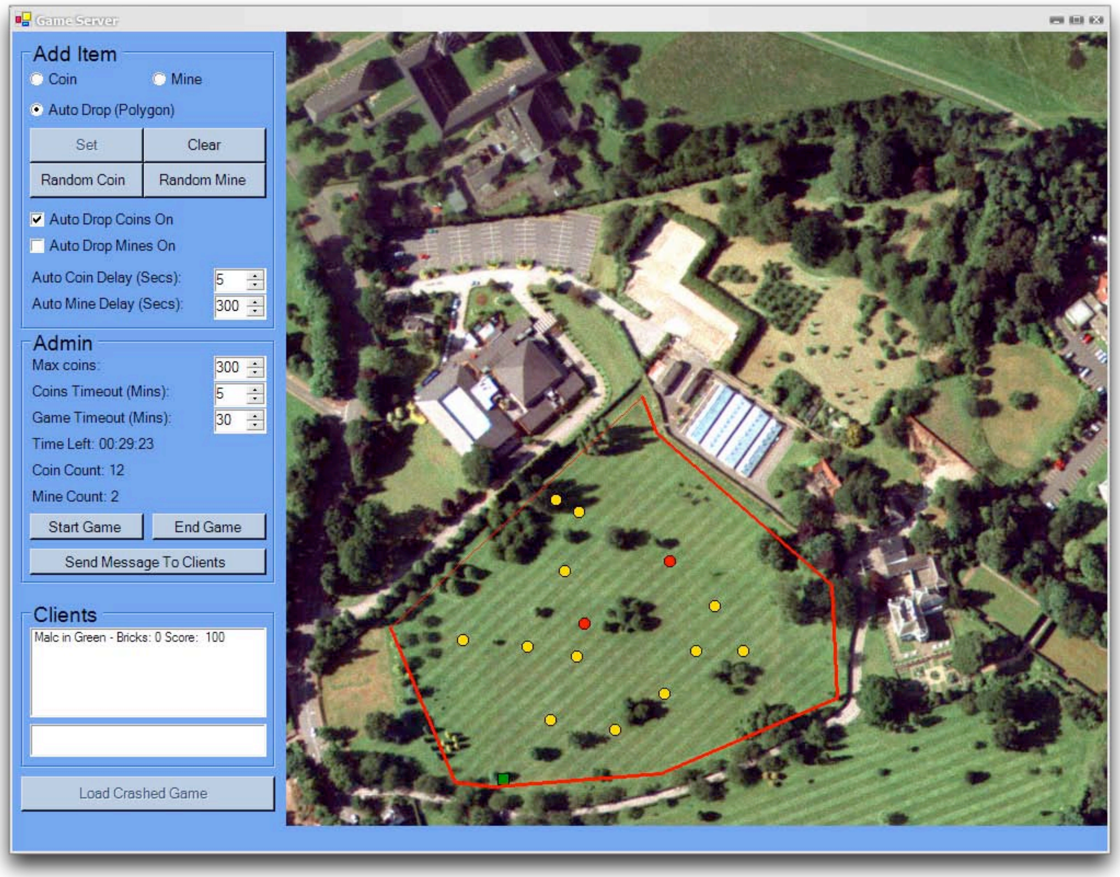
As the game is played, players' PDAs display a map of coins and the positions of other players, and this becomes overlaid with a semi-transparent coloured grid representing the signal strength coverage in the area as the users move around and sample it. This sampling is done automatically and continuously, so the overlay showing signal strength is created without the user having to perform any particular task, other than moving through the game area. When players leave the network area to collect coins, they lose connectivity to the server and updates on the progress of the game. When they return to the network, their PDAs start receiving updates again.

As stated, the chances of a successful coin upload increase with the strength of the wireless network at that location. The current signal strength of the wireless network is shown in the upper right corner of the screen as small bars; when there are few bars and they are red the signal is low; when there are many bars and they are green it is high. By moving in and out of areas of network coverage players also survey the wireless network they are playing in, building up a collective map of signal strength. The fluctuations in signal strength are logged and visualised on the map using semi-transparent coloured grid of squares representing the signal strength coverage. A high level of signal strength is shown by green squares, a moderate level by yellow, low by red, and zero by a light grey.

Scattered amongst the gaps in the field of coins are dangerous mine objects. These mines are randomly placed by the server and shown as black icons on the map interface. If a player hits or walks over a mine, he or she loses all the coins on the PDA, and the interface is disabled for 20 seconds, displaying a black countdown screen. Thus, players often have to negotiate a path around mines to reach the coins they wish to collect.

Players in the same team can upload coins simultaneously and gain double points for their coins. For example, if two players each have four coins and upload them simultaneously they will gain 80 points each rather than 40. To achieve this they have to press the Upload button at almost the same time (technically, within three seconds of each other). Players in opposing teams can also interact by pick pocketing one another. In order to make a successful pickpocket, the player must move within 5 metres of their opponent while both are within wireless network coverage and click the Pickpocket button. He or she then receives all the coins that the other player is currently carrying (but not the coins that have already been uploaded). If the player wants to prevent pick pocketing, he/she can raise a shield by clicking *Shield* that lasts one minute but is able to be recharged. Alternatively, players can move outside of network coverage to avoid pickpocket attempts, as they are safe whilst their device is not in contact with the server.

The game was managed via a user interface for the game server (Figure 12). This displayed the map and allowed a game operator to mark the area where coins should be dropped and control the frequency of drops. Additionally, it allowed the game to be started and stopped, and messages could be sent to users about game events such as where to meet up the other players when the game is over.



Coin ( ) Mine ( ) Auto Drop (Polygon) (•)

Auto Drop Coins On [✓]

Auto Drop Mines On [ ]

Auto Coin Delay (Secs): 5

Auto Mine Delay (Secs): 300

Max coins: 300

Coins Timeout (Mins): 5

Game Timeout (Mins): 30

Time Left: 00:29:23

Coin Count: 12

Mine Count: 2

Malc in Green - Bricks: 0 Score: 100

Figure 12: The Treasure server user interface for managing the game.

The Treasure game was the first system in which seamful design was applied. In the game players learn about WiFi infrastructure, and are given the opportunity to understand its characteristics. Players had to seek out areas with strong WiFi signals to ensure their upload of coins was successful. They also exploited low coverage areas by using them as places to hide from pick pockets. By simply participating in the game, players learn about the WiFi capabilities and characteristics in the area.

Figure 13: Player running through trees and bushes whilst playing *Treasure* at Ubicomp 2004.

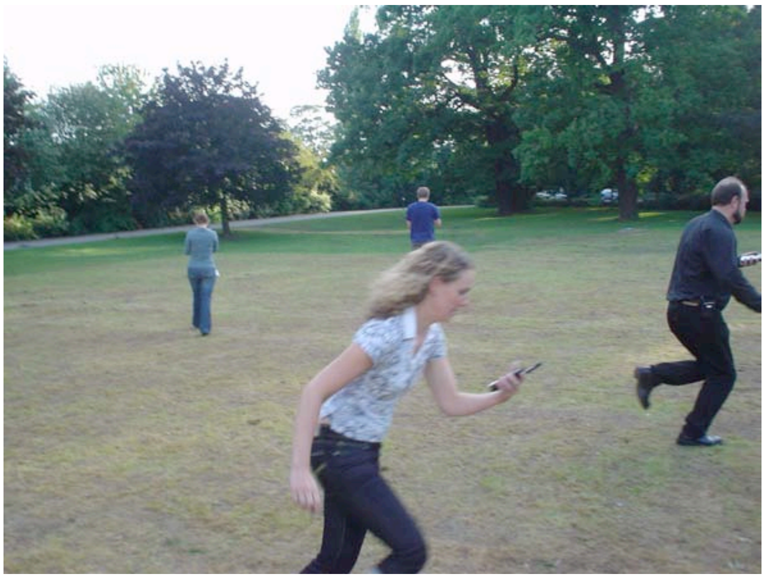


Figure 14: Playing *Treasure* at Ubicomp 2004.

In addition to our own trials at Glasgow University, *Treasure* was demonstrated at Ubicomp 2004, MobileHCI 2004 and WMCSA 2004, with large numbers of participants showing that it was, overall, successful in both game enjoyment and as a demonstration of a new ubicomp design approach.

The following year the findings of an extended user trial of *Treasure* were published at Ubicomp 2005 in [28]. Players reported that the game was a very fun experience, exemplified by the fact many of the players were extremely active during the game; running for long periods of time as they collected coins and chased other players in attempts to pick-pocket them (Figure 14). In the course of multiple games, it was possible to discover how game play changed as players' experience of the game and knowledge of the surrounding WiFi access points in the area influenced their tactics and strategies from one game to the next. Mobile multi-user systems can be hard to record and evaluate in detail, and multiple plays can increase evaluation cost and workload, although in this case it was worthwhile. By looking at video of users' behaviour, as well as detailed logs of system data, we obtained insights into the emergence and success of different strategies, and how features of the system and the setting were used in players' interaction with each other. As their understanding of the game grew, they used game features in different proportions, combinations and patterns. These changes in their game play did not always result in scoring more points, but they did generally lead to more excitement and engagement in the game. It is important to note that players' development of game play did not stem solely from the space the game was played in, the system design alone, or the space and the system together. Instead, players' use was a mix of old and new media developed through a historical and social process. Over time, people affected and were affected by each other, and system and space served as resources as well as constraints on interaction.

This finding suggested that system designers might do more to support the development of tactics and strategies by recording data on system use and reusing it within the user experience—extending the use of historical data beyond that of George Square. Overall, the game was successful in as a research vehicle—it successfully displayed that variations in ubicomp infrastructure were presented and used in ways that were crucial to user activity and enjoyment. It adds to the evidence that ubicomp technical infrastructure can be considered as a resource for users' interaction—with the system and with each other, thus interacting in a *seamful* way.

Another interesting feature of *Treasure* is that it highlighted the benefits of the design of 'games with by-products'. As a result of a game being played, a WiFi coverage map similar to those often required by WiFi network administrators was created as a beneficial side effect. Although that was not the main aim of the system, the map created can have other potential uses. Another game, ESP [29], was a web–based multiplayer game in which remote players were paired up, and typed in words to describe photos they saw. The beneficial by-product here was image labelling by humans, which could be used to subsequently power an image search engine. In this way, traditionally monotonous tasks, which currently only humans can do efficiently and quickly (such as identifying images or negotiating obstacles to cover a geographic area), can be achieved using people's motivation for fun and entertainment. The ESP authors believed that if the game was played as much as other popular online games, most of the images on the web could be labelled "in a few months" making this approach potentially very powerful indeed.

During the early development of *Treasure* we encountered issues with WiFi connectivity on the devices being used. These stemmed from the fact that players were constantly moving around, disconnecting and reconnecting to the network. It was discovered the wireless driver on these devices was not able to reliably join networks, even when in regions of high signal level, and frequently would prompt the user with interface messages about connectivity problems, or offers to connect to other networks. It was clear something was wrong in its connection algorithm, or that it was simply not designed for use in this way. A further problem was that the default driver does not update its internal representation of signal strength or expose it programmatically frequently enough to give adequate feedback to players.

By creating and utilising our own driver for *Treasure,* players experienced no problems leaving and re-entering network coverage. The driver allowed the game to be programmed to automatically scan for network coverage, detect the network used for the game and instantly connect, configure and, importantly, stick to a particular network for as long as it was available.

The implementation of the driver used for *Treasure* was developed by myself, and was of significant importance in this game and many other future projects. The code required to implement it is at a low-level and is complex, using undocumented methods and APIs. Furthermore, it was designed in such a way as to be compatible with as many devices as possible. WiFi scanning software is a useful utility, and at the time there was no existing software on Windows Mobile which worked in a device independent way. *WiFiFoFum*[6], a publicly released application subsequently developed using this driver allowed WiFi scanning and surveying using GPS on virtually any Windows Mobile device with almost any 802.11 hardware (both internal and external). The software is one of the most popular Windows Mobile applications, has had over 600,000 downloads and has an active community of users.

The driver used for *Treasure* was further developed and augmented subsequently. It was not capable of intelligently selecting which network to join or of discovering peers. To fulfil this missing functionality this core driver was augmented with extra functionality described in the following section and tested in another mobile game, *Feeding Yoshi.*

3.2.2 Feeding Yoshi

Feeding Yoshi was the second system, after *Treasure,* in the Seamful Games project in Equator at the University of Glasgow, however this time there were two other Universities involved as partners. At Glasgow it involved Matthew Chalmers, Marek Bell, Louise Barkhuus, Scott Sherwood, Paul Tennent, Barry Brown and myself. Duncan Rowland of the University of Lincoln, and Steve Benford and Alastair Hampshire of the University of Nottingham were also involved. Chalmers led the project but everyone was involved in the design. Sherwood, Bell and myself built the system, and then along with Barkhuus and Brown we trialled it, with Barkhuus primarily doing the analysis. The main paper on the system and trial of *Feeding Yoshi* was published at CHI in [22] and presented by me.

Feeding Yoshi is an example of seamful design because the characteristics of the technologies used in the game, such as the coverage and security of WiFi networks, are deliberately exposed as key features of the interface and user experience. Feeding Yoshi differs from our previous seamfully designed game,

[6] http://www.aspecto-software.com/rw/applications/wififofum

Treasure, in that it is played over a long time and a wide area. In the main user study, the game was simultaneously played in the cities of Glasgow, Derby and Nottingham (and surrounding areas) for one week. The results showed the ways in which the players embedded system use into the patterns of their daily lives, and we observed the impact of varying location on the feel of play.

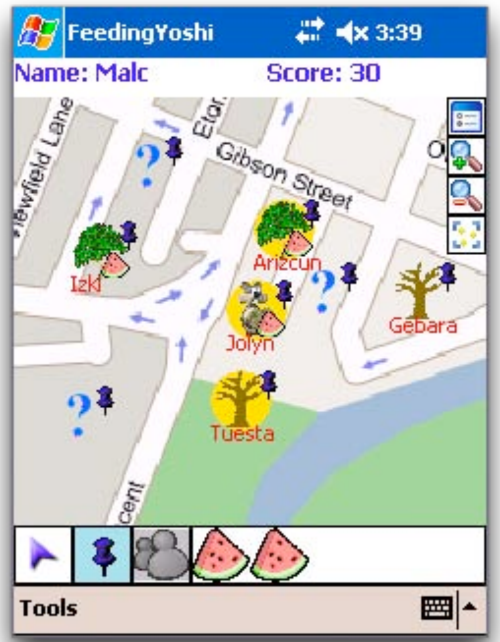


Figure 15: The *Feeding Yoshi* map screen. Yoshis and plantations are shown as icons and navigation controls are on the right. Near the bottom is a row containing (from left to right) a button for selecting icons, pinning an icon onto the map, initiating a swap with another player (greyed out), and the basket of up to five fruit: in this case two melons.

The aim of Feeding Yoshi is for each team of players to collect as many points as possible, by feeding Yoshis the fruits they desire. Yoshis are creatures that players find scattered around the city and which are constantly hungry for five fruits, of seven varieties. In order to collect fruit, players must first collect seeds from the Yoshis themselves and each Yoshi always has a seed for the fruit it most often enjoys. These seeds can then be sown at plantations that can be found scattered around the city, just as Yoshis are. Once a seed is sown, the plantation will begin to generate fruit, which can then be picked and used to feed Yoshis. Feeding a Yoshi one of his desired fruit scores 10 points, but feeding several fruit simultaneously gives more points, for example, feeding all five desired fruit at once scores 150 points. Feeding the Yoshi a fruit it does not want results in the player losing ten points.

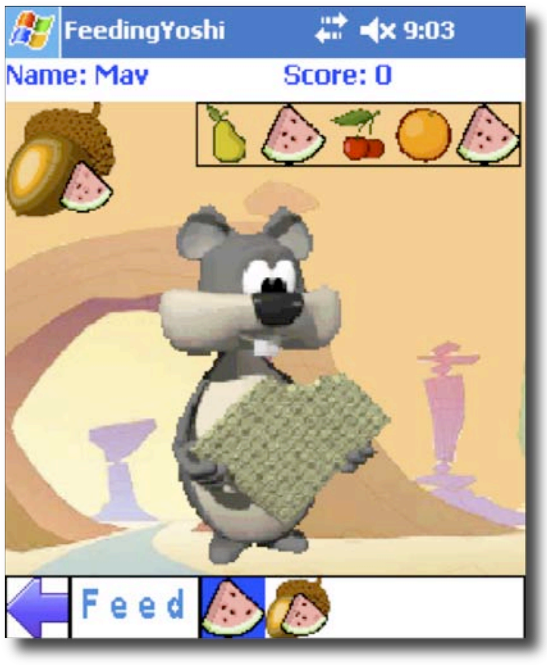


Figure 16: The Yoshi screen shows the Yoshi himself, as well as the five fruits he currently desires (top right) and a seed of his favourite fruit (top left). After selecting one or more of the fruit in the basket (bottom right), the Feed button is used to feed the Yoshi and gain points. The left arrow returns to the map.

As a player moves through the city, nearby plantations and Yoshis appear as names in a pull down menu and as icons on a map (Figure 15). An audio alert is also made when a plantation or Yoshi is detected so that the player does not have to continually visually attend to the PDA screen.

On first being detected, a Yoshi or plantation appears in the centre of the currently displayed area of the map, although a player can 'pin' a Yoshi or plantation icon in a better place. Once a few have been pinned then others organise themselves automatically using a spring model with the time taken to travel between access points as the weights. On the right hand side of the map are buttons for switching to a list view rather than the map, panning, zooming and selecting a Yoshi to be highlighted on the map as a 'favourite'. Along the bottom of this screen, and also shown in the other two screens in the game, is the player's 'basket' that provides space for a limited number of fruits and seeds to be carried. Clicking on a Yoshi brings up a screen showing the Yoshi, a seed for his favourite fruit, and the five fruit he currently wishes to eat (Figure 16). Similarly, clicking on a plantation leads to another screen with either a tree empty of fruit, i.e. an unseeded plantation, or a tree with fruit ready to be picked, i.e. a seeded plantation. Seeding is achieved by dragging a seed from the basket onto the plantation.

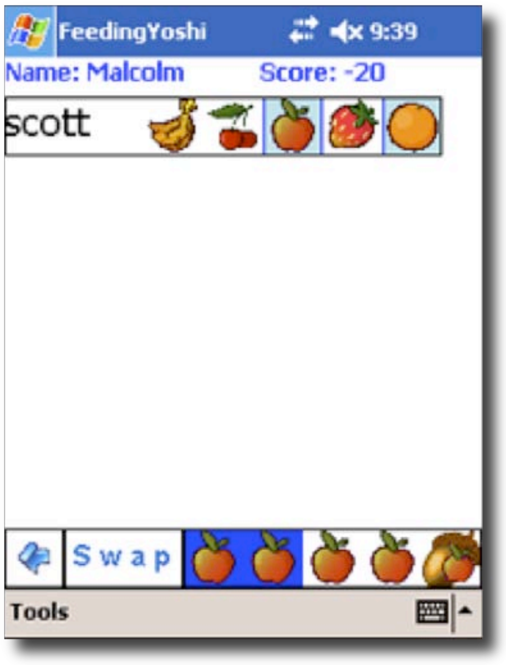


Figure 17: Fruit swapping screen. You can select items in your basket and items in someone else's basket and tap swap.

When two players approach one another, they see each other's icons on their maps. Selecting a nearby player's icon triggers an opportunity to swap fruit and seeds (Figure 17). This is useful if the Yoshis in the areas that a player knows want fruit that do not grow there. By swapping with teammates with access to other areas, they may gain more points. Swapping is also intended to encourage simultaneous play and to make it more fun to play together. Lastly, the game provides a webpage with a scoreboard showing each player's score so far, as well as the total score for each team. Players use this webpage to update their scores as described below.

The game runs on 802.11 equipped PDAs. For our trials we used a mixture of HP iPAQ 2750s and 4150s, which have built-in 802.11 and which, due to their small form factor, were relatively easy for users to carry with them throughout the week. Each PDA was additionally fitted with an SD card to allow us to store the substantial amount of log data we gathered as our users played the game.

The Yoshis and plantations that are detected while playing the game are actually wireless access points. As a player moves around in the city, their PDA continually scans for the presence of wireless networks. Secured wireless networks become Yoshis and open networks become plantations. While it would be an easy and in some ways a graceful solution to communicate with the Feeding Yoshi game server via the open access points that are discovered (e.g. to automatically upload scores), it is a matter of debate as to whether using open networks in this way is legal in some countries, including the US and UK (even though opening networks up to neighbours and passers-by may be a common and deliberate practice [30]). In order not to encourage our players to potentially break the law, Feeding Yoshi does not transmit any data over the open networks that it discovers. It only detects their existence and identity. Instead, players have to manually upload their scores at the game website using a 'score voucher code' that is generated by the PDA. This uses the PDA's MAC address as a unique key for this player to encrypt their current score and the current time in order to prevent cheating. When a code is entered on the website, decryption is attempted using the MAC address of every PDA in the game. The player's identity is made apparent, as only one MAC address is likely to provide a logical score. This workaround allowed us to keep the scores relatively up-to-date, which in turn helped to keep the game competitive between the different teams. Indeed, players reported that they often felt a strong urge to play immediately after checking the leader board and seeing their score was close to another team or player's score.

Swapping fruit between players is achieved through 802.11 peer-to-peer ad hoc networks set up between the PDAs. As mentioned in Section 2.2.4, *George Square* used Equip for discovery of peers and the services and data they had available. Whilst Equip ran adequately on the laptops and tablet PCs used in that system, it would have been too heavyweight, in terms of processing power and storage requirements, to run as a continuous process on a PDA or phone. Therefore, it was not suitable for general-purpose use on mobile devices, and instead Marek Bell developed a lightweight peer discovery system called Self Discovering Spaces (SDS) for future systems. SDS was put to first use in *Feeding Yoshi*, and operates as follows. Firstly, a modified wireless driver from *Treasure*, created by the author, is used to scan and connect to available ad hoc networks. Each mobile peer in the system will attempt to connect to the same ad hoc network, with the same SSID. Once connected to the network, the SDS running on each device continually broadcasts the service's existence, in this case the Feeding Yoshi game, using a UDP broadcast. Simultaneously, it will listen for broadcasts from others' PDAs on the same network. This is similar to the manner in which the ZeroConf service discovery[7] operates, but is achieved in a more lightweight fashion, which is more suitable for a device that may frequently be connecting to different ad hoc WiFi networks. When another Yoshi game is detected and one of the players wishes to initiate trading, SDS on that player's PDA stops scanning and sends a message requesting the other PDA to cease scanning too. This is vital as continual scanning is a relatively heavyweight network task for 802.11 on PDAs and has been found by the author to interrupt or slow the network traffic being transmitted between the devices. Finally, the exchange itself is done through traditional TCP socket connections.

Due to the situated nature of the game and our interest in observing how players responded to the contingencies of the technology and of the everyday world in which they were playing, we adopted an approach of ethnographic study. We collected data through interviews with each player, video clips of game play, a game diary that each player kept, and from system logs.

An initial pilot study was conducted with a game played over five days, between three teams of four people and with each team based in a different town or city. Our observations of these pilot games strongly suggested that the nature of the local environment (city, suburban or rural) had a great impact on the game play itself, since wireless access points varied not only between urban and suburban areas but

[7] http://www.zeroconf.org

also within different suburban areas. The study was used to fine-tune the technical issues of the game, and to make sure that it would be possible to play the game at these different locations. Finally, the game duration was extended to one week, so as to give players the opportunity of playing both at the weekend and during the working week, and in the different contexts associated with those times.

In the main trial, four teams played over a week in three different urban areas in the UK. There were two teams in Glasgow, and one in each of Derby and Nottingham. It was important for us to have differentiated milieu in order to investigate the impact of location on the experience. Glasgow is a densely populated large city, whereas Nottingham is a medium–sized city in which most of the game area was a mix between city and suburban, and Derby is a small city mostly of suburban character. In addition some participants travelled to other areas during the game, especially over the weekend. This meant that either they played in a completely different location or that they did not play these two days.

Generally the players found the basic mechanism of the game fun and engaging—that is, exploring the physical world to find Yoshis and plantations, sowing seeds, harvesting fruit and feeding Yoshis. A key factor in successfully playing Feeding Yoshi was the amount of time players invested in the game, and depending on the strategy they chose it affected their patterns of everyday life differently. The patterns of use and social interaction around the game were specific to the different players, locations and resources available for play. The quality of play, in terms of collaboration and scoring, depended strongly on the ability of team members to find ways to fit game play into their everyday lives. Play varied with, for example, people's work constraints, routes and modes of transport used in commuting, the character of home neighbourhoods, and choices and expectations as to who to spend leisure time with, and what to do where. These issues, and other trial findings are examined in detail in the paper presented at CHI [22].

There are two aspects of the Feeding Yoshi trial more relevant to the theme of this thesis. The first is the seamful design of the game, and the second is the spontaneous interactions between players using ad hoc connections between mobile devices.

Feeding Yoshi players learned to interpret urban environments in ways that would help them play the game, on the basis of their ongoing understanding of the game's technical characteristics, players' practices and the game's wider context. Inherent in the design process was an interest in using the existing ubicomp infrastructure as a resource for design and use, in a seamful way. The existing environment was part of the design context, and of the wider context of use. As Weiser put it, "the unit of design should be social people, in their environment, plus your device". Yoshi provides an example of how this can be approached, in that many of players' actions and strategies were specific to the characteristics of the wireless access points and PDAs' networking, used and interpreted on the basis of their experience and understanding of this wider context. Indirectly, players were learning about wireless networks' range, distribution and access control mechanisms, accommodating and appropriating from the perspective of the game rather than from formal education. Players also became aware of some technical features of the devices used. In one case, a player became aware—and angry about—the fact that his

PDA's 802.11 antenna had a significantly lower sensitivity than his team–mates', even though they were using the same model of PDA. Although this variation had been recognised during development, it was not expected that this variation would be the cause of such annoyance to players. The manufacturer may see smoothing this variation out as in their control and in their interests. In the future it may be interesting to consider new game designs that take advantage of this variation, introducing both advantages and disadvantages for devices that can 'see' further. For example, one disadvantage might be that they can also be seen from further away.

As mentioned, Feeding Yoshi utilised ad hoc peer-to-peer connections for players to exchange fruit between their devices. This feature was designed for players to play in groups and strategise the fruit and seeds they carry, as there is a limit of five per player. By chance, two members from opposing teams came across each other during play in a Glasgow. They did not know each other, but had both gone to the city centre to play since there were some excellent playing spots around a shopping centre. The woman from Glasgow1 who met a man from Glasgow2 describes the situation:

> *I was playing away and then this box popped up saying 'Norman would like to trade' and I thought 'I don't have a Norman on my team!' Then I saw this guy with a PDA and he was looking around, and then we caught up with each other and we thought 'hmmm… not the same team'. But he walked over and he said that he was from [the other Glasgow team] and could he trade? And well, I was in my prime playing spot so I had all the fruit I needed, so I just thought, okay I would trade with him.*

Technically, the fact that this happened this is a very successful achievement. As will be seen in the following chapter, these ad hoc connections between co-located users can be utilised for many interesting applications. However, implementing them is very challenging, requiring custom wireless drivers to overcome limitations of their design. The wireless driver built for Yoshi was an improvement of the one used in Treasure, modified to connect to ad hoc networks automatically and enabling the discovery of distributed applications using SDS. The trial proved the driver worked successfully in real spontaneous circumstances, and had great potential for being used as the transport mechanism in sharing information for adaptive systems. This driver forms the basis of the discovery systems developed in the following chapters, and its implementation is presented in detail in Section 5.3.3 in its final design iteration.

A public version of Feeding Yoshi was available at www.yoshigame.com and there was also a public version of the scoreboard. It received press coverage on some gadget blogs and there were 1000 downloads.

3.3 Conclusion

Seamful design goes back to the roots of ubicomp as a principle or ideal, but these systems show its first full application to ubicomp system design. The games were used as an initial vehicle to explore seamful design, and showed that ubicomp systems design could make infrastructure into a resource for users in ways that let them use or appropriate the system to suit their own contexts, for example, building up strategies and tactics, and to fitting them into their everyday lives. Histories of use were again found to be a useful resource in this task, in the WiFi maps of Treasure. To some extent these maps are examples of content being created by users, reflecting the benefits of user-generated content shown in George Square in Section 2.2. Again, these are principles to be taken forward into later work described in this thesis and help to form a solution to RQ1.

Chapter 4 Mobile ad hoc peer-to-peer systems

This chapter introduces mobile ad hoc peer-to-peer systems with a discussion on background work in the research area. It is shown that these kinds of systems have features that are applicable to the design of dynamic ubicomp systems. For example these systems can allow systems to share information opportunistically between the devices carried by people as they meet. Also, the information can lead to dynamic changes such as recommendations for content like information or media, places to go. Another type of information is to do with the awareness of other people in the vicinity, sharing their interests and behaviours. Following the presentation of background work in this field, the next systems created as part of this PhD are discussed. These include *FarCry*, a technical demonstration of how to achieve epidemic spreading of media between mobile devices using spontaneous ad hoc networks, and peer-to-peer Recer*,* a decentralised history-based recommendation architecture in which the history data is distributed amongst mobile devices over ad hoc networks.

4.1 Background

An interesting behaviour becoming increasingly common is the *beaming* of games, pictures and phone numbers between peoples' mobile phones. Both Infra Red (IR) and Bluetooth are technologies built into most phones that support this instant beaming technique that it is very quick and easy. Compared to IR, Bluetooth does not have the line-of-sight restriction and the range of approximately 10m (100m for class

1) covers well the immediate environment of people [31]. Using either of these technologies can simplify activities for the receiver. For example in the case of file exchange, it is not necessary to type URLs to visit file-sharing websites; he/she just receives the data directly from the other person's phone 'there and then'. Another main advantage is that it is free — commercial network operators cannot charge for sending data in this way since it does not traverse their networks. Although this has not been proven, it is possible that users actually prefer to share private data in this way. Since it is a direct transfer, there is no concern that anyone on the Internet might intercept the information, so there are fewer obligations to secure files by encryption before transfer. However it is still possible to intercept short-range wireless transmissions, so if deemed necessary the wireless communication can be secured. For example, the Bluetooth specification offers secure communications between two devices and the LoKey [32] system utilises the closed SMS network to exchange encryption keys to secure ad hoc WiFi connections. It would appear that some users think it is more personal to exchange things in this way and they are appropriating the technology built into the phone in a unique way. We note that sharing data between phones using ad hoc connections is the creation of mobile peer-to-peer systems.

This section first examines social proximity applications (SPA) that utilise the every day meetings between humans to drive the sharing of information between the mobile devices they carry. The second section discusses systems that exploit this novel sharing technique to distribute information epidemically.

4.1.1 Social proximity applications

Everyday meetings between humans involve many intricate protocols. Interactions occur at many different levels, ranging from negotiating how to pass someone on the street, acknowledging familiar strangers with a friendly hello nod, a one on one meeting with a colleague, to a presenting to a lecture theatre filled with hundreds of people. As people's mobile devices are becoming equipped with wireless networking capabilities, their devices now also have opportunities to meet. These meetings between devices involve similar complex protocols including peer discovery, negotiating an ad hoc connection, and efficient information transfer. This means people cannot only share their physical appearance and presence with others, but also exchange digital information that may include multimedia and recommendations. Meetings between people offer the insight into how best to implement the interactions between devices. Now that these benefits have been realised many researchers are developing applications to make use of this novel communications link.

The ability for wireless networking technologies to perform in ad hoc mode is well suited to supporting face-to-face interactions between mobile devices. An ad hoc network is "a transitory association of mobile nodes which do not depend upon any fixed support infrastructure. Connection and disconnection is controlled by the distance among nodes and by willingness to collaborate in the formation of cohesive, albeit transitory community." [33] Since the range of wireless devices tends to be rather limited, and it may be beneficial for your device to contact someone just outside your range, technologies are being built to create bridged ad hoc networks where information can travel between people to reach a destination. A mobile ad hoc network (MANET) is a self-configuring network of mobile routers (and associated hosts) connected by wireless links—the union of which form an arbitrary topology. The routers, in this case people, are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. MANETs tend to involve routing data over a live mesh network, however another approach could involve a store and forward type configuration where people can store messages, then pass them on later once they have travelled to a new place. Beaufour [34] propose a system to dissipate data across disconnected static nodes over a wide area by leveraging the movement of mobile individuals equipped with smart tags. Umbrella.net [35] is another such system but with the novelty that the hardware is integrated into an umbrella and the network of umbrella nodes is only formed when it is raining.

Soundpryer [36] is a wireless ad hoc mobile peer to peer car stereo application which allows people to tune into a music broadcast from someone else's car stereo which is near by. It allows people to take an aesthetic interest in surrounding traffic by sharing music, in a normally anonymous situation. A similar system, BlueTuna [37] is an application for personal audio devices allowing the 'tuning' into the music playing on other peoples devices at locations where people regularly congregate, for example bus stops.

Proem [38] another example, is an open computing platform targeted at mobile ad hoc information systems. Built by the University of Oregon, it provides support for developing and deploying collaborative peer-to-peer applications for mobile ad hoc networks and personal area networks (PAN). It is a collection of tools, API and runtime for developing and deploying applications and works on a really low level. A specific transport protocol has been specified. It uses XML for representation of messages and can then be implemented on top of TCP/IP, UDP or HTTP for instance.

Traditionally, the term 'peer-to-peer' describes direct communication between hosts on a large network topology, for example, the Internet. As long as any host is connected, no matter where it is geographically, it can send and receive data from any other host on the Internet (Firewall and NAT permitting). In contrast, the term 'mobile ad hoc peer-to-peer' means the network topology is created only for the peer-to-peer communication to take place, for example, between a minimum of two mobile devices that have short range wireless radios. Only the devices within wireless range of one another are able to communicate at any given time, however the overall number of devices which may come into range in the future is a much larger set. As the mobile devices move around, ad hoc connections are created spontaneously, creating a highly dynamic set of small independent network topologies.

We note that for reasons of brevity the term 'peer-to-peer' is often used in this thesis as a shorthand for 'mobile ad hoc peer-to-peer'. When the more traditional meaning is intended, this will be mentioned in the text.

The author is largely in agreement with Korteum et al. when they expressed strong opinions on the ad hoc peer-to-peer topology a mobile system should employ [38]:

In some sense, an ad hoc mobile information system is the ultimate peer-to-peer system. It is self-organizing, fully decentralized, and highly dynamic. However, current peer-to-peer systems are designed for stationary hosts connected to an Internet-like infrastructure.

Korteum et al. claim that mobile ad hoc information systems are, by definition, fully decentralized, negating the use of hybrid and centralised peer-to-peer topologies. This highlights the importance of that particular topology as a target in design. Korteum et al. list the advantages a peer-to-peer system inherits by utilising a mobile ad hoc network topology in a pure peer-to-peer environment [38]:

- ***Self-organizing****: as side effect of the movement of devices in physical space, the topology of a mobile peer-to-peer system constantly adjusts itself by discovering new communication links.*
- ***Decentralized****: each peer in a mobile peer-to-peer system is equally important and no central node exists.*
- ***Highly dynamic****: Since communication end-points can move frequently and independently of one another, mobile peer-to-peer systems are highly dynamic.*

The benefits of using a decentralized architecture are seen later in this thesis, firstly in *Samara,* a mobile ad hoc peer-to-peer recommendation system (Section 4.3.2), designed to overcomes the limitations of the centralised recommendation system design used in *George Square*.

In the previous year the potential for ad hoc networking applications has begun to be realised and consequently a few tools and applications have been developed which take advantage of this communications technique between co-located users. The kind of software that does exactly this is called a social proximity application (SPA). Persson classified SPAs into four distinct types [39]:

1. *Proximity messaging*
2. *Providing awareness*
3. *Intelligence in the exchange of data*
4. *Supporting identity expression*

The following literature reviews are of systems that fit into each of Persson's classifications.

The first type of SPA involves proximity messaging, the first example of this was not from a piece of purpose developed software, but actually from a novel appropriation of the use of the Bluetooth feature for transmitting your name and number to a nearby phone - essentially a business card exchange. People found that by changing the information in the business card from their name and number to a note or advert for example; they could use this as a medium for sending messages to strangers - a behaviour now known as ‘Bluejacking’. Mobiluck[8] is an application developed for this specific purpose that simplifies the detection of all nearby Bluetooth enabled phones and send messages and photos for free to friends or strangers with no need for their phone numbers.

One such system is the Hummingbird [11], a small device equipped with a short-range radio transceiver, through which it broadcasts its identity and receives information about other Hummingbirds in the vicinity. The devices are functionally self-contained, i.e. independent of surrounding infrastructure. The overall objective is to support awareness of “who’s around” within an established group of people. Whenever two or more Hummingbirds are close enough to communicate (maximum range 100m), the devices emit an audio notification and display the identity of the other device. In this way, it is possible for users to know which other Hummingbird users are in proximity. In studies of user experiences it was found that the Hummingbird is particularly useful in situations where a group of users are outside their normal environment, for example, when travelling [40] [41]. Although Hummingbirds lack the ability to mediate communication, the experiments showed it is be worthwhile to at least have the knowledge that other users are in close vicinity.

More recently, in research concerning awareness provision, the Familiar Stranger Project[9] at Intel Research suggested that trends in mobile phone usage were dividing co-located strangers within a community. For example, when in strange or uncomfortable situations, it is common for people to reach for their mobile phones as opposed to communicating with others around them, resulting in a decline in social interaction [12]. The *Familiar Stranger* was a social phenomenon first addressed by the psychologist Stanley Milgram in 1972 [42]. *Familiar Strangers* are individuals that we regularly observe but do not interact with. Paulos and Goodman at Intel investigated “digital scents” and “digital tagging” and consequently developed the system *Jabberwocky* designed to support awareness of unacquainted users [12]:

[8] www.mobiluck.com
[9] http://berkeley.intel-research.net/paulos/research/familiarstranger/index.htm

As two people approach one another, each person's individually carried Jabberwocky transparently detects and records the other's unique identity. Over time each Jabberwocky accumulates a log of unique entries of people that have been previously encountered. Similarly, a person is able to "digitally tag" a place (i.e. park, plaza, bus stop) or object (i.e. bench, bridge, parking meter) by attaching a fixed Jabberwocky to it. The combination of fixed and mobile Jabberwockies is the essence of the Familiar Stranger system.

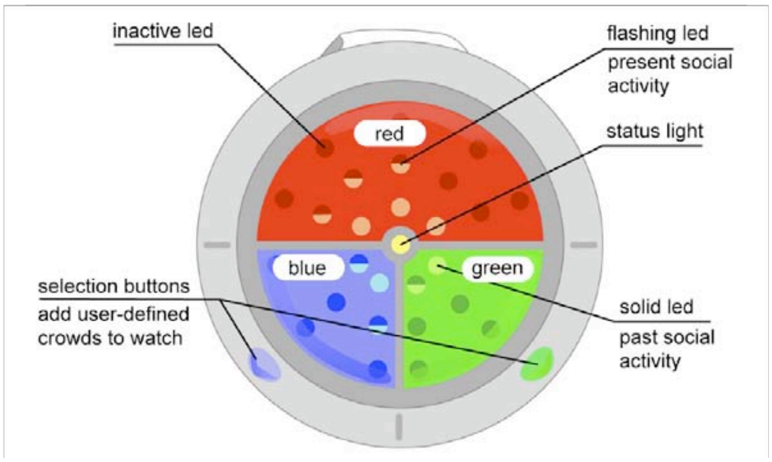


Figure 18: The interface for the iMote-based *Jabberwocky* device.

The system was implemented first on Bluetooth phones and then a custom hardware version followed, which was built using low powered embedded iMote[10] devices, and was small enough to be clipped on a belt or used as a key ring. The iMote interface (Figure 18) is split into three regions of coloured LEDs and has 2 buttons. The number of lit red LEDs represents the 'degree of familiarity' that corresponds to the overall number of Familiar Strangers who have been in the user's current area, whilst flashing red conveys the number currently nearby. Specific groupings of crowds, such as bus stops or neighbourhoods, can be tagged using the Blue and Green buttons and the presence of those groups is shown in the corresponding coloured LED areas. *Jabberwocky* was designed to extend the Familiar Stranger relationship while respecting the delicate, yet important, feelings between strangers in public places.

Road Rager [43] utilised proximity messaging in a WiFi peer-to-peer system to implement the communication mechanism in a multiplayer game. Part of the Backseat Playground Project[11], Road Rager is a game for child car passengers to play whilst travelling in the backseat. They carry PDAs disguised as wands and augmented with rows of LEDs. During traffic encounters, when kids in other cars are in range, the LEDs light up to alert the player that others are nearby and to give some basic directional information about where he or she may be located. Various types of attack can be launched against the peers by

[10] www.intel.com/research/exploratory/motes.htm
[11] http://www.tii.se/mobility/BSP/docs/project.htm

making gestures with the wand and typical interactions last only seven seconds but can consist of several data transfers. One second after a connection is made the player has already been notified and responded by making an attack gesture. After another six seconds, he sees the attack has succeeded and begins to discuss his next attack move with another traveller in the same vehicle. In many mobile peer-to-peer applications, such rapid interaction between peer devices is likely to be necessary, and it is clear that WiFi over MANETs performs extremely well in this scenario to support smooth interaction of this type. *Back Seat Gaming*[44] was a similar game as part of the same project but the targets to attack with the wand were geo-stationary objects at the side of the road, and the game was single-player so there were not such tough communication requirements.

In the third type of SPA, some sort of intelligence is embedded in the exchange of data between collocated users, with the purpose of giving recommendations or supporting collaborative work. Dating applications are the most simple and common examples of systems involving intelligent data exchange between co-located users. In these applications, the user completes a personal profile questionnaire, which is then compared with the personal profiles of other users in the vicinity. If there is a 'match' found between users, they are notified of the result and are often sent a picture of their counterpart (e.g. proxidating.com, dreamlove.it and bedd.com).

Similarly, WALID [45] is another system that makes use of this type of SPA enabling intelligent data exchange between co-located users. However, its purpose is completely different to that of the dating applications mentioned previously. WALID allows users to define various lists of tasks or errands that they personally would like to be completed. Whenever two WALID devices meet, their software compares the lists and suggests a trade of tasks between the users if there is a match.

As the dating applications and WALID illustrate, there can be many different purposes for this type of SPA. Social Net [46] is yet another example of the third type of SPA but this time with the purpose of providing information about a user's social network, more specifically, a new kind of network, a social network based on proximity. By collecting time logs of co-present users and then comparing these time log, common acquaintances of the newly acquainted users can be established.

The fourth type of SPA allows users to create a more or less sophisticated identity expression, to be broadcast to proximate users in order to facilitate socialising. NewsPilot [47] [11], developed as part of the MobiNews Project at the Viktoria Institute, allowed journalists in a broadcasting house to jot down the stories they were working on at the moment, which would then be shared with co-located fellow journalists, supporting collaboration. However, in an extension of this work, the system was changed to use ad hoc wireless connections between co-located PDAs.

An even simpler form of identity expression is to simply set the Bluetooth device name of the phone, which can be seen when other phones perform a scan. There is a significant culture of naming phones and other devices in the UK [48]. A study by Kindberg et al. [49] found that these naming practices reflect the social, physical and intentional context of the phone's owner.

Hocman [50] is a system designed to support identity expression and information sharing between motorcyclists when they are out driving. As they pass each other they get a brief glimpse at the other rider's bike and style, and in the same brief moment their PDAs exchange information including text and pictures, for example where the biker prefers to ride or equipment they own, to be viewed later. A sound in the rider's ear notifies them that an exchange occurred. This scenario presented implementation challenges for the communications between bikers' devices including exchanging information in a very short time frame. Since motorbikes are likely to be travelling at high speeds, this presented a technical challenge in this research revolving around the issue of enabling successful data transfer when the connection time available is extremely short. Systems like Hocman and Road Rager would not be possible without extremely fast and reliable peer discovery, and this is clearly one of the most important features the underlying communication technology must support. WiFi offers high-speed data rates and also works when travelling at high speeds, however it has some limitations such as unreliable connections, and complicated set up. These limitations have subsequently been addressed in the *Treasure, Yoshi* and *Far Cry* systems developed as part of this thesis, through implementation iterations of a sophisticated WiFi driver and peer discovery component, explained in each of the systems' sections.

DigiDress aka Nokia Sensor (Figure 19) is an application which allowed mobile phone users to create digital identity expressions, that could be seen by other phone users within Bluetooth range [39].

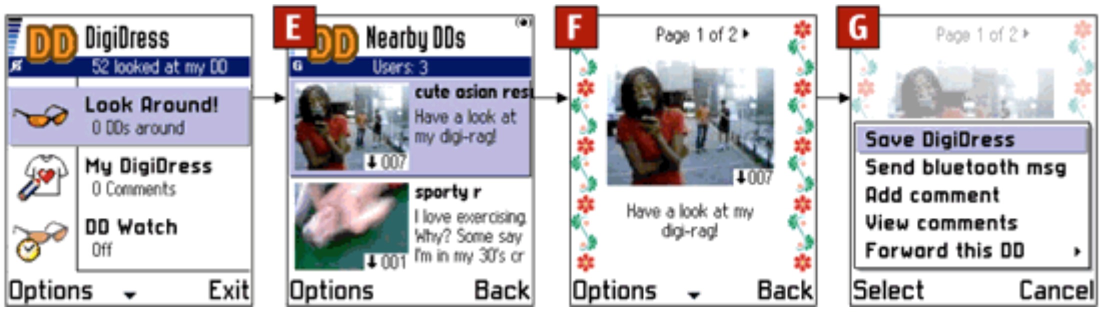


Figure 19: DigiDress user interface.

During the 89-day trial period, 618 users installed DigiDress on their phones, the majority of installations involving a phone-to-phone distribution style. DigiDress was one of the first applications to exemplify the term 'viral' distribution where, when an application is installed, it can then be duplicated to nearby devices immediately and very easily, taking advantage of the simplicity of ad hoc wireless technologies. The developers commented that this kind of distribution was critical to their trial, in that the simplicity of allowing people to share the application promoted its uptake. The average use span was 25 days. Some identity expressions created were serious, others playful, and some revealing more than others. Factors influencing the identity expression included strategies for personal impression management, privacy concerns, and social feedback. The application was used with both acquainted and unacquainted people, and viewing the identity expression of people nearby was one major motivation for continued use. Direct communication features such as Bluetooth messages were not commonly adopted. In several instances, DigiDress acted as a facilitator for social interaction, such as verbal chats, between previously unacquainted users.

4.1.2 Epidemic distribution using mobiles

Social ad hoc networks provide a novel platform for data distribution. For example as a person travels from home to work their device can carry data. Data could also be spread through connections between multiple people in a crowded street, with eventually everyone receiving it. This flooding-based approach, like diffusion, has a strong similarity with epidemic spreading of diseases.

The Cityware Project[12] presents a novel view of ‘The City as a System’. In the keynote of the 2006 Workshop on Software Engineering Challenges for Ubiquitous Computing, Kindberg suggested the London Underground subway network of trains could be thought of as a computer network with a huge amount of bandwidth. He suggested that since thousands of people travel by train and carry devices such as Apple iPods with gigabytes of storage, this ‘system’ could be regarded as a high bandwidth data network where the mobility of people is a metaphor for the transmission of data. As part of the Cityware Project, Bluetooth scanners were deployed across the city of Bath, UK to examine mobility patterns of people (by detecting their phones), and how they are affected by the way the city is structured [48], and an understanding of how people use space [51]. These models of people’s movement could possibly be exploited to enhance the epidemic transmission of information between people’s mobile devices.

Mobile ad hoc routing protocols allow devices to communicate without any pre-existing network infrastructure. However, Vahdat explains why existing protocols cannot be applied when social proximity is the target use [52]:

> *The majority of existing ad hoc routing protocols, while robust to rapidly changing network topology, assume the presence of a connected path from source to destination.*

This assumption is not valid given power limitations, the advent of short-range wireless networks, and the wide physical conditions over which ad hoc networks must be deployed; in some scenarios it is likely that this assumption is invalid. In this case, one develops techniques to deliver messages. Vahdat and Becker introduced the concept [52]:

> *Epidemic Routing, where random pair-wise exchanges of messages among mobile hosts ensure eventual message delivery… in the case where there is never a connected path from source to destination or when a network partition exists at the time a message is originated.*

[12] http://www.cityware.org.uk

The *FarCry* system presented later in this chapter offers a technique to spread media files epidemically between mobile devices, refined in later systems including Domino in Section 5.2.

4.1.3 Emerging commercial ad hoc peer-to-peer systems

As short-range wireless technologies such as WiFi and Bluetooth are becoming increasingly popular in all kinds of mobile devices, the commercial opportunities for proximity-based systems are beginning to be realised. Unsolicited Bluetooth messaging or 'Bluejacking'[13] was recognised as a potential marketing platform in 2003, where messages containing deals offered by nearby shops could be sent from a base station to people walking past [53]. Bluetooth is particularly useful in this scenario because nearly all phones can receive Bluetooth messages as part of their standard set up, and no additional software is required. Although WiFi offers larger range and is also becoming just as popular as Bluetooth, it requires configuration by the user, for example the device must be connected to a network, and software must be running on both devices for messages to be transmitted and received. However, marketing software to take advantage of people's mobility to spread promotions and discounts between users has been attempted [54]. Proximity based messaging is clearly a very powerful commercial platform, however its failure thus far to become more than a simple marketing technique could be because when digital content is being transferred outside of the control of a centralised system, there is the technical problem of how people can pay for or authorise the content they receive.

Melodeo, Inc is a company taking advantage of people's beaming behaviour to increase the sales of music. Melodeo Mobile Music Solution is a new peer-to-peer music-sharing platform where mobile phone users will be able to securely send full tracks that they have purchased, from one mobile phone to another mobile phone via Bluetooth. The receiver can play the track back in any player that supports the DRM used in the track for a limited time, however they purchase the full-length version of the track by downloading the Melodeo Mobile Music Solution. This software resides directly on the user's phone, allowing consumers to quickly and easily shop, preview, purchase/download over the air, and play and store full-length music tracks. Melodeo's software is compatible with many different phones and they have partnerships with many handset manufacturers, mobile operators, and major record labels.

The Microsoft Zune[14] is a portable MP3 music player that has built in WiFi (Figure 20). Zunes feature the ability to set up ad hoc (Zune to Zune) connections. Nearby Zune devices can be searched for. Then songs or albums can be exchanged however there is a limitation of 3-days or 3-plays, essentially it is a like a trial. Songs are transferred including their meta-data so album art persists between exchanges. After the 3 days or 3 plays are up, the song gets deleted from the Zune on the next sync, unless the song has been purchased. Photos can also be exchanged between Zunes and there is no limitation on viewing them.

[13] http://www.bluejackq.com
[14] http://www.zune.net

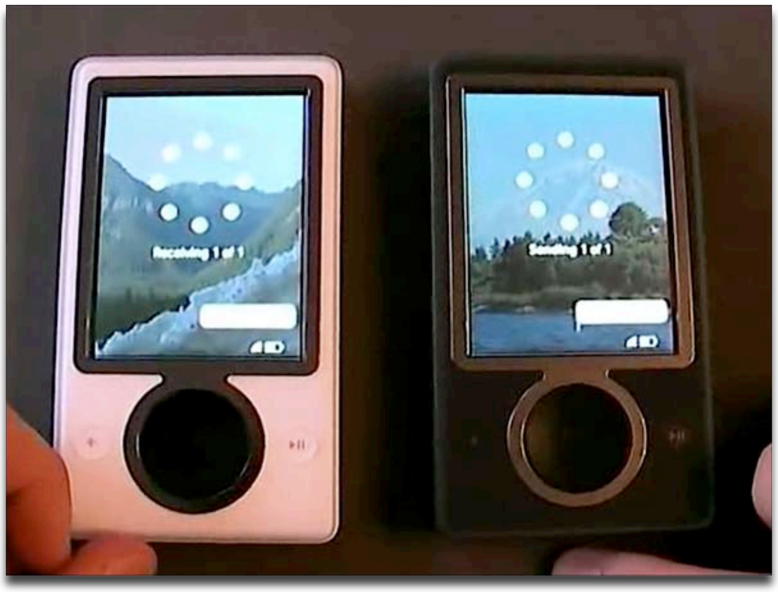


Figure 20: The Zune MP3 player exploits ad hoc WiFi connection to exchange music files.

One thing limiting the popularity of the Zune is the issue that enforcing DRM has proven to be quite restricting in ad hoc environment, since the licensing server might not be contactable. In October 2006, due to customer demand, major record labels agreed to sell DRM free music[15]. This certainly lifts one technical restriction of distributing by ad hoc networks, but the problem remains of how the music is eventually paid for by the receiver. This issue is relevant to the technique used for distributing software in this thesis, as some ad hoc distributed software might be required to be licensed and paid for. If an elegant solution to this problem can be developed it is possible that peer-to-peer beaming is going to become as popular as peer-to-peer file sharing on the Internet. One possible solution might be commission based, for example the more you distribute files, the more share of royalties you might receive.

Microsoft has included peer-to-peer sharing capabilities in the newest version of their operating system Windows Vista. The P2P Meeting Place[16] feature can automatically create ad hoc meeting networks and thus enable two or more users to connect directly with one another to share files, applications, nearby projectors, or an Internet connection, and it also supports instant messaging. Some example scenarios of use people watching a presentation can follow the slides on their own laptop instead of the projector. There is also the ability to work on documents together – you can edit the shared file and the modified version will be saved to the remote user's machine. These new automated meeting networks between laptop users will hopefully provide more opportunities for social proximity recommendation applications to exist.

[15] http://en.wikipedia.org/wiki/Digital_rights_management
[16] http://www.microsoft.com/windowsxp/p2p/default.mspx

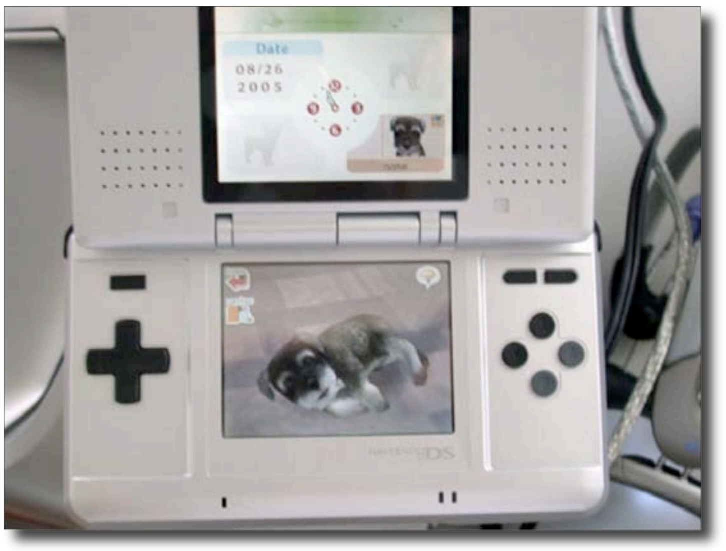


Figure 21: Nintendogs running on the portable Nintendo DS.

Nintendo released a game for their Nintendo DS portable games machine, called Nintendogs[17] (Figure 21). The DS features WiFi networking and although a few multiplayer games have been released, Nintendogs was the first to make unplanned interactions with strangers a feature of the game, which they named "passer-by mode". The player chooses between three versions of the game, each with different dogs, and trains the dogs using their voice. Once trained, the dogs will not respond to commands from anyone else. The player can walk around with the game set to passer-by mode and the game detects other DS systems also running the game. Players can send voice messages and presents can be exchanged automatically. It's interesting that in order to make progress in the game and discover new items, players actually have to go out and physically find others also playing the game. One player, whose dog is named Drinky, posted the following message in a gaming forum[18] conveying their fun experience with the game:

> *"I had another passer-by contact today, so I'm going to try this a lot more often from now on. I put the DS in passer-by mode before I left the house for work, and on my journey to work, Chiharu's Labrador Retriever, Inui, visited. He was a cute little fellow, wearing a pair of sunglasses. He brought me a present too! A pulling rope toy. I'll let Drinky play with it later. In return, I gave a Mushroom toy. Chiharu is a pro-trainer though, so she probably already has one... oh, and she left me a voice message "Please be nice to Inui."*

4.2 Far Cry

This section presents a framework for the pervasive sharing of data using peer-to-peer connections over wireless networks. *FarCry* uses the mobility of users to carry files between separated networks and groups of users. Through a mix of ad hoc and infrastructure–based wireless networking, files are

[17] www.nintendogs.com
[18] www.neogaf.com/forum/showthread.php?p=1287925

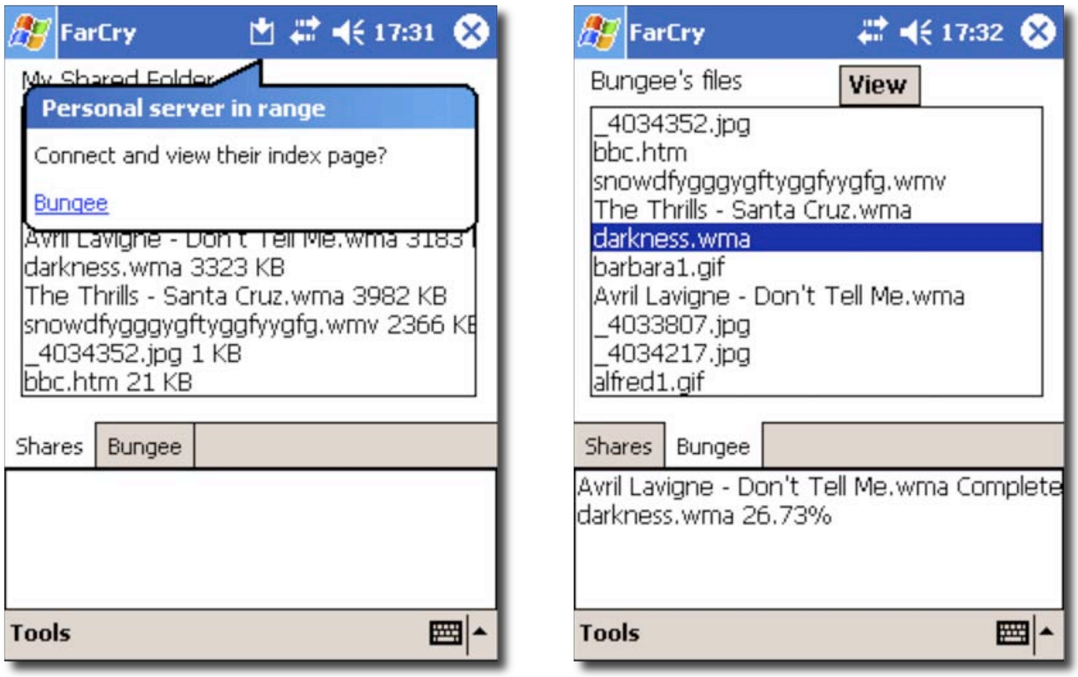
transferred between users without their direct involvement. As users move to different locations—meeting other individual users and communities of users—files are transmitted on to others, spreading and sharing information. Three applications using FarCry were developed. Each exploits the physically proximate nature of social gatherings, like the social proximity applications mentioned earlier. FarCry, and the applications built using it, leverage the fact that people with similar interests tend to spend time together. For example, as people group together in business meetings and cafés, this can be taken as an indication of similar interests, for example, they may work in the same field or enjoy the same food. The *MediaNet* application built using FarCry affords sharing of media files between strangers or friends. Another application, *MeetingNet*, shares business documents between work colleagues in meetings. The final application, *NewsNet*, shares RSS feeds between mobile users. NewsNet also develops the use of pre-emptive caching: collecting information from others not for oneself, but for the anticipated sharing with others at a later date.

FarCry is a .NET Compact Framework assembly—a library of code that can be easily by used by other .NET applications that run on mobile devices, or PCs. While it currently works over WiFi, its functionality is simple enough to be used over any TCP/IP network, such as a connection set up over Bluetooth or infrared. An important distinction from previous systems is that FarCry's functioning does not interfere or disrupt the normal use of communication hardware or network transmissions. That is, users are able to continue to use their devices for common network tasks—such as checking email or browsing the Internet—whilst FarCry operates simultaneously and transparently. This is possible because FarCry operates on infrastructure networks, such as a user's own wireless network, and will create its own network if necessary. Thus, whilst users are near infrastructure connections, such as a WiFi hotspot, the user can connect and use it as normal. However, when away from infrastructure nodes, FarCry will fall back to using ad hoc connections to communicate with peers. This allows FarCry to always be able to discover peers and supports the normal use of the communication hardware by the user.

FarCry allows applications to select files to be shared, and to specify which files can be received from other FarCry–enabled devices, and it provides an interface for dealing with the user-initiated spreading of files between devices. When running, FarCry spawns a background task that searches for other FarCry peers. If the device is currently on a wireless network, a search is made for other peers using a lightweight service discovery robust to changes at the physical network layer; the Self Discovering Spaces component implemented in *Feeding Yoshi* in Section 3.2.2. Using SDS, all FarCry clients broadcast their existence over UDP to the subnet; allowing listeners to become detect them. If no infrastructure networks are available, or the client is unable to obtain a connection, FarCry will create an ad hoc network to which other clients can connect. This ad hoc network has a fixed SSID of which all FarCry clients are aware. FarCry continues to scan for this ad hoc network even while connected to infrastructure networks, changing to the ad hoc network if necessary. When FarCry uses its own ad hoc network, IP addresses for devices are obtained using a similar approach to ZeroConf[19]. To switch between ad hoc and infrastructure mode, and to join WiFi networks, FarCry utilises the wireless driver created by the author. An early

[19] http://www.zeroconf.org

version of this driver was employed in Feeding Yoshi. However, the driver in FarCry is an improvement over that in Feeding Yoshi because the ability to switch between infrastructure and ad hoc networks in order to discover peers has been added; the driver in Yoshi was only capable of utilising ad hoc connections. Furthermore, when playing the game, the driver from Yoshi requires full access to the WiFi device. The improved driver from FarCry, however, only takes full control of the hardware when it detects the device is in an idle state. This means that when in operation the user's own network connections are not affected.


FarCry 17:31

My Shared Folder

Personal server in range

Connect and view their index page?

Bungee

Avril Lavigne - Don't Tell Me.wma 3183
darkness.wma 3323 KB
The Thrills - Santa Cruz.wma 3982 KB
snowdfygggygftyggfyygfg.wmv 2366 KB
_4034352.jpg 1 KB
bbc.htm 21 KB

Shares | Bungee

Tools

FarCry 17:32

Bungee's files | View

_4034352.jpg
bbc.htm
snowdfygggygftyggfyygfg.wmv
The Thrills - Santa Cruz.wma
darkness.wma
barbara1.gif
Avril Lavigne - Don't Tell Me.wma
_4033807.jpg
_4034217.jpg
alfred1.gif

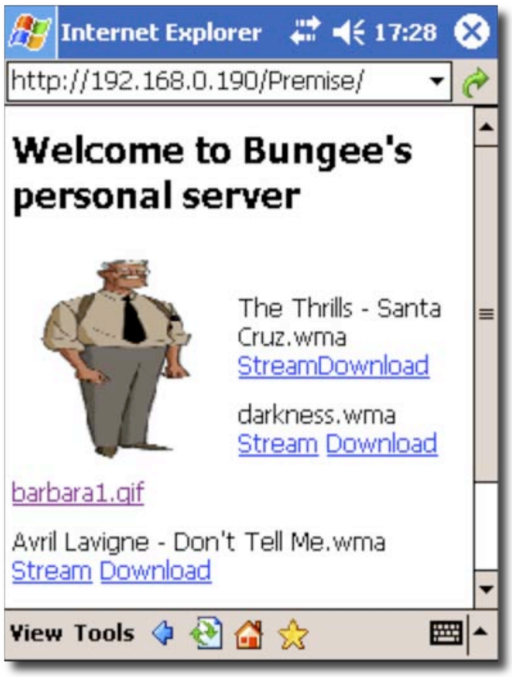
Shares | Bungee

Avril Lavigne - Don't Tell Me.wma Complete
darkness.wma 26.73%

Tools

Figure 22: The FarCry user interface. A notification appears when FarCry detects another user nearby, and then a list of that user's shared files can be viewed.

When FarCry devices discover each other and are connected to a sharing session, a file exchange system is set up using a simple personal web server running on both devices. Each downloader client automatically connects to the other's web server, downloads a list of all available files from other peers, and transfers files (subject to an application–defined filter). Files that are stored on both devices have their timestamps compared, with the newest file propagated in order to overwrite the older version. FarCry also supports an interactive mode where users are notified that a peer is nearby and can browse the shared files and select ones for download (Figure 22). Shared files can alternatively be viewed through a user customisable personal webpage, where a user name and avatar can be displayed (Figure 23) inspired by the identity expression SPA type. FarCry keeps a record of transferred files, so as to avoid duplicate transfers. As a protection against viruses, the FarCry assembly will only exchange documents or media, and will not exchange program code (identified by file extension). However, as will be discussed subsequently, distributing software modules in this way is a main theme of this thesis, presented in Chapter 5.


Internet Explorer 17:28

http://192.168.0.190/Premise/

Welcome to Bungee's personal server

The Thrills - Santa Cruz.wma
StreamDownload

darkness.wma
Stream Download

barbara1.gif

Avril Lavigne - Don't Tell Me.wma
Stream Download

View Tools

Figure 23: The personal web page of a FarCry user, allowing personalisation and supporting peer-to-peer audio streaming.

From an epidemic algorithm viewpoint, FarCry works on a device-by-device and file-by-file basis. Devices are ‘susceptible’ when they have not seen a particular file. When devices have files they become ‘infectious’, in that they can spread a file to other devices. Lastly, when a user or application deletes a file the device is ‘removed’ or immune—it can no longer be infected with that file.

As stated, one of the first applications built to demonstrate FarCry was MediaNet, a media sharing system designed to disseminate media files through a network of friends. Although controversial, media sharing is a popular pursuit. Some estimates put the amount of Internet traffic consumed by media sharing as high as 60% [55].

The FarCry framework depends on location, and thus facilitates sharing amongst people who meet regularly: generally friends, colleagues, and 'familiar strangers' [12]. MediaNet allows users to view, stream or download available files from any peers present nearby—excluding those files protected by digital rights management. Media can be automatically swapped, supporting the unobtrusive sharing of media between those in the same social grouping. This exploits the fact that social groupings of people are often co-located, and also often share tastes in music, film etc. Automatic swapping allows individuals to browse new acquisitions at leisure, but an alternative mode allows users to be alerted when other users come into view, and to browse their files while still connected.

Since these exchanges are triggered simply by network range, files are exchanged not only between friends, but also between strangers who are physically proximate. One proposed situation for this kind of sharing is for the cultural tourists studied with regard the *George Square* system: such tourists would tend to congregate in museums and related places, where sharing of media relevant to them as a social grouping could happen. The system also allows cross–pollination between distinct social groups, for example, people who go to the same social settings (such as the same bar) would exchange files more often than those who do not—although one might consider the bar visitors as being of a loosely tied social group. The nature of MediaNet might also allow the transfer of files between two individuals who have similar tastes but who do not know each other, and have not been in close proximate range where a direct transfer could occur. If a friend of both spends time with one on a certain day, and meets the other the next day, then the files that both may be interested in can become shared between these two strangers who have never met despite sharing the same friend or colleague.

In addition to supporting the complete transfer of a file to another device, MediaNet also allows streaming, meaning those using devices with lower storage capacities, such as flash–based PDAs, can also be supported. This part of the application is particularly suited to commuting, where users maintain proximity with strangers for long periods of time. MediaNet stores a history of streamed files, allowing the user to browse this later, locating and downloading them from the Internet when more time or storage is available.

The second example application demonstrates the use of FarCry in a workplace environment. MeetingNet coordinates a network connection between all the WiFi enabled devices present at a meeting. Within the meeting a folder on users' machines is dynamically set up and all files in that folder shared between participants. Users can then distribute electronic files to all attendees, such as to pass PowerPoint files around. Attendees simply place the files into the folder (either in advance) or during the meeting, and files will be exchanged between all the participants.

Technically, this system is very similar to MediaNet, differing only in details of its interface. One added complication over MediaNet comes from the private nature of the files likely to be exchanged. In our current application, MeetingNet is only practical for use with non–confidential files. Since files are automatically shared, it is possible that those who are within the vicinity but not actually at the meeting can share files. A planned extension to this system involved the use of a password to initiate sharing amongst meeting participants. All participants would all have to enter the same code to share the files.

The second area of functionality provided by the MeetingNet application is that of instant messaging. It is frequently useful to carry on interaction silently with one's peers during a meeting without speaking out and disturbing the flow of the meeting (so called 'backchannel' communication). However, colleagues may not all be subscribers to the same messaging service. Again, the geographical boundaries of FarCry's local networking are turned to positive use, by restricting the messaging system to those within range of the network, and running the MeetingNet application—in practice, those present at the meeting. Of course, both these features can be replicated using other tools, if a suitable network is set up. However, the automation of FarCry aims to make this process straightforward.

NewsNet leverages FarCry to make use of a recent popular technology—Really Simple Syndication (RSS). Unlike MediaNet and MeetingNet, NewsNet depends on the user having had a connection to the Internet at some point in the past. It examines a user's bookmarks for those based on RSS feeds, and then caches time-stamped versions of those pages. When the user meets or passes another NewsNet user, FarCry coordinates a network connection, and each device compares its list of feeds with the others. If any match is found then the older page is updated with the additional articles from the newer.

In this manner, a user is able to receive news updates without actually connecting to the Internet. Like MediaNet, NewsNet exploits social grouping, depending on the premise that friends share similar interests, thus have a better chance of matching feeds than complete strangers. In this way, NewsNet allows users who are outside Internet range to be updated with new feeds from others passing by who have recently accessed the Internet. NewsNet could be extended to predict which feeds other users may request and cache these, even if the current user is not currently subscribed to them. This is a form of 'pre-emptive caching' of news for other users. The highly ranked feeds vary with social interaction; for example, a group of technologists is likely to have several technology sites in their rankings, while a group of musicians may have more musically relevant sites stored. These kinds of recommendations are the topic of the peer-to-peer Recer system detailed in Section 4.3.1.

If a user wishes, he or she may browse the news feeds available on another device visible over a FarCry network. As with the practice of iTunes[20]–based 'musical voyeurism', NewsNet allows a 'news voyeurism', rather like reading a newspaper over someone's shoulder. In certain situations, such as

[20] http://www.apple.com/itunes – iTunes supports browsing music libraries on the local network.

commuter environments, this may prove an enlightening diversion, while its use among friends may lead to the same kinds of judgements described in [56].

Of the three systems described, NewsNet is the one most likely to benefit from the serendipitous meeting of strangers. While specialist news feeds are likely to be confined to those with similar interests, those of more general interest, such as local news, may have wider appeal. Additionally the size of the files used by NewsNet is likely to be considerably smaller than those of MediaNet, so downloading content from passing strangers is more plausible.

In our experiments with FarCry, it was found that files were reliably exchanged at a distance of around 100 metres. This may be too far to effectively support the sort of social situations that FarCry is designed for. Certainly for MeetingNet, the WiFi network extends far outside one meeting room to encompass adjacent rooms. Shorter-range wireless technologies could be used to define social groups, for example Bluetooth, and then the higher bandwidth WiFi network connection could be powered on and used for data transfers. Alternatively, activation of FarCry's transfers could be set to only occur when a high WiFi signal strength is detected, thus shortening the effective range by ignoring low signal strengths that correlate to large distances.

It is important to note that FarCry is still a potentially useful technology even if there is no clustering of interests, so that exchanges take place solely between strangers. In this case FarCry works at the level of a sharing 'zeitgeist' files—files that are popular in a particular area or organisation. This would mean that, for example, in MeetingNet, files from all meetings are exchanged and with MediaNet a geographical area's files are exchanged.

FarCry takes advantage of the mobile and social nature of individuals, providing a framework for mobile data sharing. Set-up overhead and bandwidth restrictions do not adversely affect the system, thanks to a dependency on social gathering as a means of determining compatibility. Sharing of information using FarCry exploits social proximity and face-to-face, rather than attempting to overcome distance, as most Internet–based applications do. In this way, FarCry has the potential to be integrated into existing social networks, such as groups of friends, in turn spurring further use.

4.2.1 StreetHawk

During development of FarCry it was discovered that it might be advantageous for peer-to-peer ad hoc systems to sometimes connect to infrastructure networks to gain Internet connectivity at times when no ad hoc peers are in range. To explore this, StreetHawk was developed to test opportunistically connecting to high-bandwidth infrastructure WiFi networks, with zero user interaction. The WiFi specification supports roaming between access points of the same network [50] but there is no seamless support when the network changes. Connecting to WiFi networks can often be troublesome for users as many issues such as interference, low signal, even hardware incompatibilities, can lead to much more user interaction and technical troubleshooting than should normally be necessary. WiFi is often preferred for connecting to the Internet when mobile because of high cost and low bandwidth of cellular technologies. The aim of StreetHawk was to investigate the use of systems that can opportunistically connect to the Internet, as users are mobile, in a city for example, without any required user interaction. Then if successful, mobile systems could be designed to exploit both infrastructure and ad hoc modes, for example systems that share data epidemically as best they can over any network medium available—like FarCry and Domino in this thesis.

In Manhattan, New York, the Times Square area has 3272 WiFi Access Points (Figure 24), and as many as 50 available to be connected to at any location. This is an extreme example, but a similar abundance WiFi networks can be found in most major cities in the developed world, and it is growing by the year.

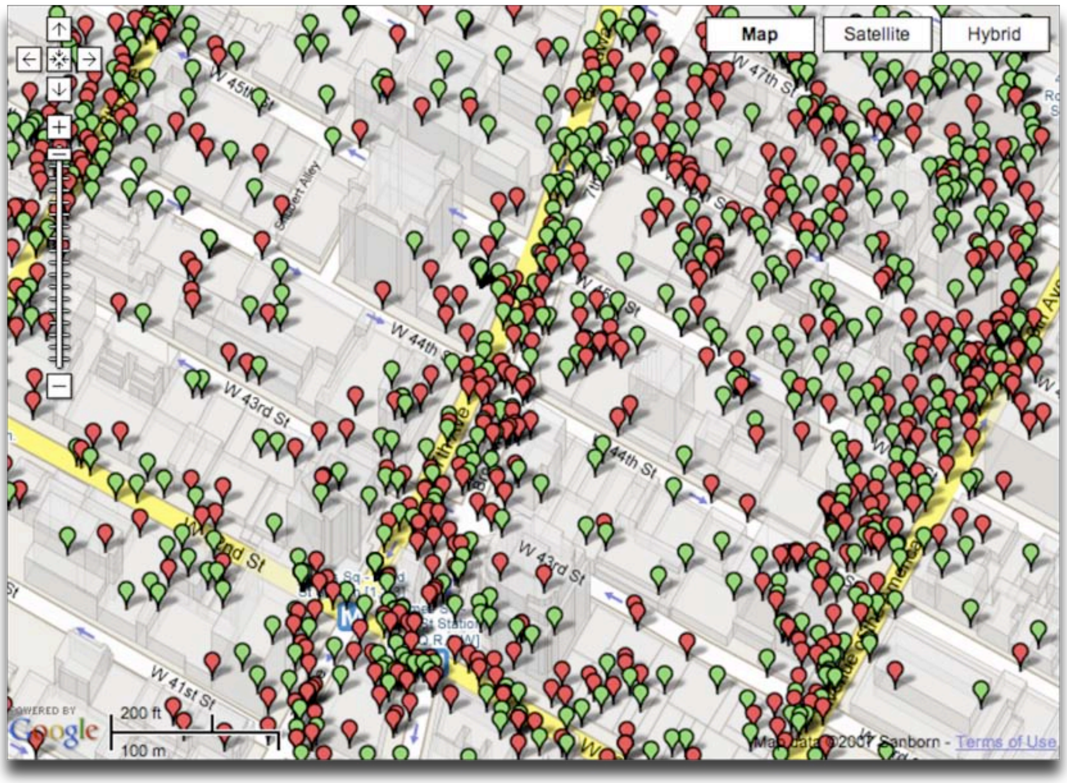
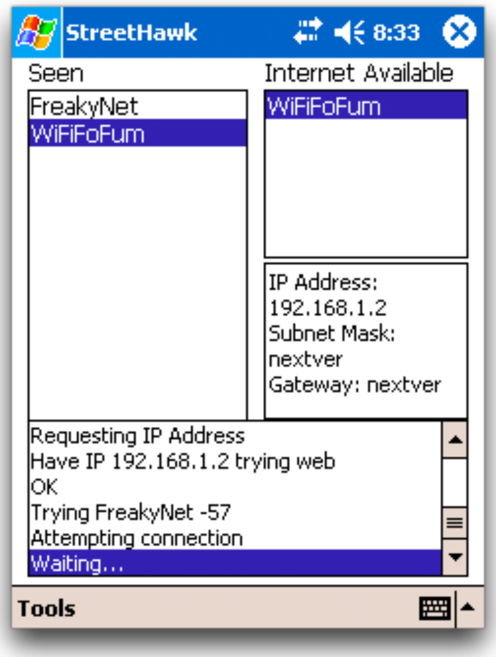


Figure 24: A map showing the Times Square area of New York with 3272 WiFi Access Points[21]. Red markers are secured networks, green are open.

StreetHawk will automatically step through available unsecured network names (SSID) and in turn, connect and check if there is web access by requesting a known web page (Figure 25). A known page is required because fact alone that a request has been fulfilled and a page has been returned is not conclusive, because the page request might be redirected to another, for example a hot spot log in page, so the page returned must be checked to be a match to a known one. StreetHawk expects a page with one

[21] Source: http://www.navizon.com

word as the contents. It only tries networks that are in infrastructure mode because ad hoc networks rarely have Internet connections. When a new SSID is detected it will try to associate (connect), and if it fails it will retry this SSID when the signal is stronger. If association is successful, it will then try to obtain an IP address from a DHCP server, usually the access point (or wireless router) itself. If an IP address is not allocated, it will try this SSID again the next time only when the signal is stronger, because in the authors tests, low signal is the main cause of this failing. Once it does obtain an IP address it will try to request the Google.com index page. If the request succeeds and Google responds, the SSID is added to the Internet list. Otherwise if an IP was obtained but it could not get the Google page, it will not try this SSID again.



FreakyNet
WiFiFoFum

WiFiFoFum

IP Address:
192.168.1.2
Subnet Mask:
nextver
Gateway: nextver

```
Requesting IP Address
Have IP 192.168.1.2 trying web
OK
Trying FreakyNet -57
Attempting connection
Waiting...
```

Figure 25: StreetHawk is an application to allow Windows Mobiles to automatically connect to WiFi networks with Internet access.

Thus there are 3 steps when connecting to a wireless network:

1. *Setting the SSID*
2. *Associate with an access point*
3. *Receive an IP address from a DHCP server*

Depending on signal strength, StreetHawk can connect to the Internet through an open WiFi network in approximately 20 seconds. Since implementation, some legal and ethical issues have come to light, about whether using Internet connections on open WiFi networks should be done, and this issue should be examined before utilising this approach.

The software was released to the public as an application for Windows Mobile Pocket PC devices[22] and has been very popular. Streethawk can seek out a WiFi Internet connection wherever it is available, however an issue remains of how to detect when the user is active, and would not appreciate the network settings being altered.

StreetHawk successfully demonstrates a novel 'WiFi network roaming' technique to automatically connect to unsecured WiFi networks that have available Internet access. This technique is relevant to many research areas that require automatic WiFi connectivity, for example handover between cellular and VOIP telephone calls, and with the inclusion of ad hoc networks. It can be used to discover peers to share data with on either network type—this is the discovery technique used by the adaptation architecture Domino, presented in Section 5.2 and StreetHawk's modifications to the wireless driver are utilised.

4.3 Efficient distribution of data in mobile, peer-to-peer environments

The brief investigations conducted with FarCry led to a realisation that an intelligent method for a system-controlled epidemic spreading technique and a method for both identifying and culling irrelevant or old data was required. Such culling of data is necessary to prevent devices filling up their storage whilst maintaining a healthy flow of data through a peer community. Culling old data is particularly important in a mobile environment since storage on mobiles is scarce compared to desktop PCs. Similarly, processing power on the average mobile device is also far lower than on desktop machines so, if the stored data is to be analysed or searched, then a large amount of redundant or irrelevant stored information can have a significant impact on an application's performance. Therefore, due to both processing power and storage capacity it is critical to ensure that only relevant data is maintained. This requirement led to an aim to investigate possible solutions to this problem and, following our experience in the area of mobile recommendations in George Square, it was decided that the source data for those recommendations should be used as the bulk data to be spread epidemically between devices.

One of the main limitations of the George Square system is that the recommendation system relied on a central server, through which all devices would submit event logs and request recommendations. There are many advantages of decentralising the recommendation system and allowing the system to run on multiple mobile devices with no reliance on a server. The first and most obvious reason is that the server may become unavailable on occasion, either because of a hardware or software failure, or because it has become unreachable due to intermediate network problems (which is a very likely scenario when using mobile devices, because of the high volatility of their wireless network connections). Additionally, as demonstrated by background work concerning Social Proximity Applications, it has been shown that there are many advantages to relying on the movement of, and proximity between people as the transfer mechanism through which data is spread from device to device. Furthermore, an ad hoc connection

[22] http://www.aspecto-software.com/Streethawk/index.htm

between devices is usually much higher bandwidth and at zero cost than infrastructure connections (WiFi hotspots or GPRS), allowing for a far greater amount of data to be transmitted. Thus, a peer-to-peer version of Recer was developed to investigate the control of epidemic distribution of history logs, used for producing recommendations with no reliance on a central server.

Following this implementation of Recer, Marek Bell developed a prototype application *Samara* (short for System for Ad hoc MetAdata RecommendAtions), a tourist guide involving multiple mobile devices, that recommends potentially interesting places to visit on a map, based on where others have been.

4.3.1 Peer-to-peer Recer

The Recer collaborative filtering algorithm, introduced in Section 2.2.3 was used in the George Square system to generate recommendations of places, web pages and photos. There was a substantial limitation that only one copy of the database, containing the logs used to generate recommendations, existed on a server. Thus, Recer was utilising a centralised architecture that proved too restrictive for use on mobile devices. As mobile clients moved in and out of network connectivity the server would switch between being available and unavailable, with recommendations either being delivered to the user or being absent respectively. By decentralising the database and distributing history logs efficiently, a peer-to-peer version of Recer was created. The initial distribution algorithm used to control the spread of the data in the distributed database environment is the design of Marek Bell. Both Bell and the author participated equally in all aspects of the implementation.

The history information Recer records is relatively small, as each individually logged action is typically under 100 bytes, which results in approximately 10,000 entries per megabyte of history data. Even though individual entries are compact, the entries are spread epidemically and the number of entries on any one device can multiply quickly, which leads to storage space becoming an issue. Thus, a level of control is required to ensure that devices do not crash or fail due to available storage being completely consumed by Recer. In a mobile peer-to-peer environment, the length of a peer encounter may not be predetermined. As users continually move around—meeting and parting from one another as they conduct their daily business—encounters may last only a short period of time, and often the wireless connections between their devices break off suddenly as the peers move out of range of one another. Therefore, it is critical that the information transferred in the short period available is that which is most likely to be of value to the recipient.

When a peer conducts its first exchange, it has no previous experience about which data may be of most relevance. Thus, in this initial case, the histories transferred between peers can be random or simply starting from the most recently collected data and working backwards through time. As mentioned, the wireless connection can be unreliable, and due to this the decision was made in Recer to transfer the history information in small segments of 100 entries at a time. This is a compromise between the performance gains of a bulk transfer and the subsequent database insert, whilst still preventing data loss of large amounts of data if a connection is broken. No more than 100 history entries will be lost if any single connection is broken, which is a relatively small number as an overall transfer could potentially contain tens of thousands of entries. To further assist in preventing data loss the transfer mechanism is implemented using separate threads to conduct the actual network transfer and to store and process that data. This often allows a secondary thread to extract any successfully transmitted histories regardless of where the transmission breaks. For example, if the connection is lost whilst entry 51 of a 100-entry segment is being transmitted then the secondary thread will be able to recover the 50 completely received entries. This robust protocol ensures a high successful transfer rate even in the extremely transient and unreliable peer-to-peer network environment.

As a consequence of peers receiving other peers' histories, they have the ability to not only share copies of their own histories but also those of others from which they have downloaded information, increasing the speed of epidemic spreading. For example, a client may download sections of history that belong to user A and sections that belong to B during a single encounter with device A. This can occur if device A had previously downloaded path data during an encounter with device B. Thus a single encounter with one peer can provide a wide variety of data from many users.

After history data has been received from an exchange event, and inserted into the database, it can be used when generating recommendations using Recer's usual technique described in Section 2.2.3. The recommendations generated might not be solely textual information, which is stored in the field within the logged event; they may be a reference (or URL) to a filename or some larger piece of data. Thus the recommendations might refer to the actual data that is to be shared within the peer community. For example, a music sharing peer-to-peer application may use the Recer algorithm in order to epidemically spread the Recer histories of MP3s song names played. However, this underlying metadata is shared only to allow for the identification of data that the peer is sharing which is most likely to be of interest.

Following the creation of a set of recommendations that point to files or data, these items can subsequently be actively searched for within the peer community if they are not immediately available from the device that initially caused the recommendation to be generated. For example, in the case of the music sharing application, if device *A* receives a history of songs played from peer *B*, which causes a recommendation for a particular song to be generated, then *A* may immediately request the song from *B*. If *B* has a copy of the song it will provide it for *A*. Note that, as devices carry history data from many peers rather than just their own history logs, it is not unusual for a device to cause a recommendation for a file it itself does not have to be generated. If this is the case and *B* does not have the song, then *A* will make a note of the recommendation and actively seek it out in encounters with other peer devices. When the item that has been recommended is discovered on a peer device it can either be automatically downloaded or may be presented to the user, who can then decide to accept or reject the recommendation for the file download to their device. Thus, this technique of recommending shared peer data not only filters the data to select that likely to be of most relevance but also supports both system-controlled and user-controlled epidemic spreading of data.

After a number of histories have been downloaded, the system periodically analyses them during times when the device and applications are idle. It attempts to determine which histories will generate the recommendations of the highest interest in the future. Each of the downloaded histories on a device are compared to the user's own history, and an average per-item similarity rating is calculated. The final rating gives the average similarity on a per-item basis rather than over entire histories, this is important because a history average would not give a meaningful ranking unless compared to a history of similar length. Only segments of history that are downloaded during peer encounters, thus it is unlikely that the two histories would have exactly the same number of entries. Therefore, a ranking calculated over an entire history would have a bias that simply favoured longer paths.

The similarity rating is subsequently employed in another feature of peer-to-peer Recer, its ability to prioritise the transfer of histories from which recommendations more relevant to the user are likely to come from. After similarity rankings have been calculated for histories, all subsequent history exchanges during peer encounters rely on them to give the highest ranked histories priority; that is, to prioritise those histories from which the most useful recommendations are most likely to be generated. Once histories have been compared, when a peer is subsequently encountered, it is queried for the list of users for which it carries history data. This is compared to the rankings list and if there are any matches then segments of history from the highest ranked user are requested first. If the length of the connection is long enough to allow the entirety of the history data stored on the peer device to be downloaded then the rankings list is again consulted and the next highest matching history requested. If there are no further matches then history data is again requested at random. Gathering data from these histories first helps ensure that the short encounters in a pure peer-to-peer environment are less likely to be irrelevant, the data that is likely to result in the most beneficial recommendations to the user is given highest priority.

The list of similarity rankings is also used to provide a method for culling irrelevant data in the interest of conserving storage space on the device. When storage space is full, or when it is approaching an allowed limit set by the user, the ranking list is consulted and the lowest ranked histories are deleted. This is regarded as relatively safe as this data is considered the least relevant to the user: least likely to lead to any recommendations in the immediate future. If, after deleting these histories, there is still not the sufficient level of required storage space, the oldest segments of history logs are removed, without removing the entire history. This produces a healthy routine of deleting old and irrelevant path data whilst renewing it with new data downloaded from peers. The overall outcome is a recommendation system that is kept up-to-date and continues to provide novel recommendations.

4.3.2 Samara

Samara is a prototype application, designed to test peer-to-peer Recer by recommending sight seeing locations to tourists exploring a city. The application uses GPS to log the user's movements, and recommendations for places to go appear as rectangles on a map (Figure 26). Marek Bell created samara, but this author implemented the WiFi discovery and peer-to-peer Recer components, and the outcomes of testing those are relevant to this thesis, as they used in later systems.

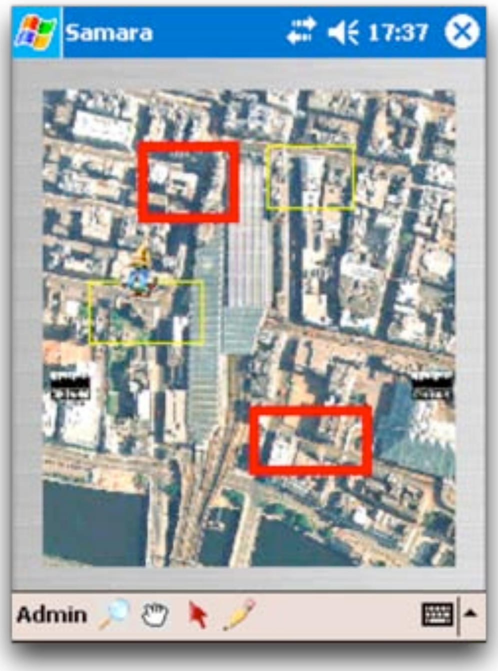


Figure 26: Samara is an application utilising a fully ad hoc peer-to-peer recommendation system to recommend places on a map to visit. Locations visited by the current user or other users are shown as yellow rectangles. Locations recommended to the user currently are highlighted in red.

The tourist carries a PDA equipped with GPS while walking throughout the city. Data from the GPS unit is delivered as a stream of text in the standard NMEA format, and the latitude and longitude delivered approximately once per second. A significant problem is working out how to interpret locations and areas people consider interesting, from this raw NMEA stream of coordinates.

Internet services exist that can provide a postcode for a supplied latitude and longitude. If a postcode were to be retrieved from a web service that provided this conversion, a business directory look-up service could then be used to supply a list of businesses at the current postcode. This method allows for intelligent guesses to be made about the name of a building the user had entered. However, since the aim is to produce an application for a pure peer-to-peer environment, it was required to remove all reliance on external third-party centralised services. Furthermore, this method relies on a static database of officially recorded buildings. It would, therefore, fail to recognise personal or locally known locations such as a common meeting place, or locations in the countryside. Such areas may be of great importance for someone who uses such a location to picnic, walk their pet or simply admire the view. Thus postcodes, addresses, and even listed place names sometimes are not useful to identify locations people visit.

To overcome the limitations outlined, a self-generating content structure for defining the places people visit was implemented. This worked by logging the GPS coordinates of a user, trying to identify locations of importance, and marking a surrounding region whenever the system detected the user was at a location that seemed to be important to them. Thus, it is the user's own actions that are relied upon to identify the locations that are of value to them; there is no reliance on officially recorded names for these locations. Identifying important locations is a significant research problem, and Samara utilised two published techniques that allow for both outdoor and indoor locations (at building granularity) to be detected [57].

The first technique is designed for indoor locations of interest and involves recording the time and locations where a GPS fix is lost due to loss of signal from satellites due to obscuring of line of sight to the sky. A GPS fix is a useful measurement because essentially it is a boolean value, providing a simplified starting point for a building identification algorithm. If the signal is not regained within a certain timeframe, for example five minutes, then it is suitable to assume the loss of signal is that the user entered a building. As in [58], a threshold of at least three instances of such maintained loss of signal is required around the same location within a period of two weeks before the location is accepted as one of interest, and marked as such by the system. This threshold aids in filtering out accidental losses of signal due to the user entering narrow street with tall buildings, heavy tree cover, accidental disconnection of the GPS unit from the PDA or other device failures. In this way, only entrances to buildings that the user frequently visits, and are therefore likely to be important to them, are recorded as important locations in the internal database.

The second technique is designed for outdoor locations of interest, and simply involves monitoring the period of time a user does not move over a certain distance threshold. If approximately the same location is maintained for a period of time, for example five minutes, then the location is recorded. As in the first technique, three detections are required within a two-week period for the location to be permanently entered as important in the location database. This technique allows, for example, the detection of locations such as a pleasant area on a beach or in a park.

When a location of interest is detected it is assigned a globally unique identifier (GUID) logged in the database on the user's PDA. If users desire, they can at any time review the locations and assign a more meaningful name. This can aid others in identifying what may be at the location if they later receive the entry on their PDAs. In addition to the two automated location detection techniques outlined, one may also manually enter locations that are of interest by panning the map to the desired location, and using an interface tool to draw a rectangle around the location. A tool for allowing the user to fake visits to locations by manually positioning themselves at any location is provided, along with a tool for deleting locations and visits, in the interest of privacy issues. Rectangles are used to identify locations because of the simplicity of storing and querying them in traditional databases. If irregular shapes are required, an improvement would be to use multiple smaller rectangles to make up larger shapes, or a spatial database such as PostGIS[23] could be utilised to provide efficient queries and storage of these location shapes.

[23] http://postgis.refractions.net

Once locations have been identified and recorded in the database, the system also monitors the user's GPS location to detect when the user visits them. The user is marked as having visited them whenever they are within the boundaries of that location for five minutes or more. The record of the visit is stored by entering the action of visiting the location into the user's Recer history log. This allows the application to identify any locations that are important to the user and creates a record of the user's visits to them.

Whenever a user's device encounters a peer, the previously discussed peer-to-peer version of Recer is responsible for exchanging the most relevant items for generating future recommendations. Thus, new locations and recommendations to visit these locations spread throughout the peer-to-peer community in a system-controlled epidemic manner. Recommendations for locations from a community of users supports learning about new locations, such as restaurants which have just opened, in the user's own area as well as providing a vital resource and guide to a tourist in a new city.

Samara demonstrates a novel use of a peer-to-peer epidemic distribution algorithm. It uses recommendations from a standard collaborative filtering system to drive the spread of data within the peer-to-peer community, with source data produced by the community, with no reliance on centralised servers.

4.4 Conclusion

This section has identified literature involving mobile ad hoc systems, and of particular importance are those systems that take advantage of people's everyday movement as the transport mechanism for information in a purely ad hoc mobile scenario - the social proximity application (SPA). To reiterate Persson [39] classification previously mentioned, SPAs are separated into four distinct types:

1. *Proximity messaging*
2. *Providing awareness*
3. *Intelligence in the exchange of data*
4. *Supporting identity expression*

The systems mentioned all fit into this classification. For example Persson's DigiDress was an example of an SPA that supported identity expression, and the others also fit into the classification. It appears there is a gap or potential advance on the current state of the art, specifically a new, fifth type of SPA:

- Sharing data to generate intelligent recommendations of software.

This potential advance in SPAs is the topic of work described later in this thesis and is also a concrete demonstration of an answer to RQ2.

This chapter also discussed several experimental systems that contribute to the systems described later in this thesis. FarCry demonstrated the epidemic spreading of information via peer-to-peer connections over wireless networks, aided by a custom wireless driver. StreetHawk pushed the limits of making use of dynamically discovered WiFi networks. The peer-to-peer version of Recer, built on top of the FarCry platform and tested in Samara, allowed history logs to be distributed and maintained efficiently and a recommendation service to be built that avoided George Square's limitations due of centralisation. Overall, the experience gained from implementing and testing these systems helped to answer RQ1 and meant we had reliable systems for peer discovery, communication and information sharing—identified in Section 4.1 as being key requirements of a successful mobile adaptive infrastructure, and used in the Domino infrastructure described in the next chapter.

Chapter 5 Dynamic software adaptation

In ubicomp, researchers agree there is a need for systems to become more adaptive and dynamic. First steps towards achieving this non-trivial ideal can be gained by examining the wider field of adaptation. Adaptation involving computers is vast area of research with many sub-topics, and this chapter examines only a subset of the technical aspects, for example flexible software architectures, of the user experience aspects, for example how people react to change. Following this, an adaptation architecture developed as part of the work in this thesis, *Domino* is presented along with its design and implementation as a possible solution to the requirements of adaptive dynamism in ubicomp systems. Domino's design encompasses the findings from the earlier chapters to do with context awareness, Seamful Design and mobile peer-to-peer systems (Chapters 2,3, and 4).

5.1 Background

The word *adaptive* can be quite ambiguous in the context of computer systems. The user interface can adapt to the device its being displayed on, the system can adapt the interface to be more efficient depending on what users regularly do (often called 'collaborative adaptation', i.e. the system and the user collaborating), systems can adapt to repair themselves when things go wrong or offer additional features over time, and content displayed can be adapted. The increasingly widespread use of the term makes it even more difficult to distinguish between meanings. This section examines the background research on different types of adaptation considered relevant to the design of adaptive ubicomp systems.

5.1.1 Displaying adaptive content

The simplest form of adaptation could be considered to be the presentation of customised content depending on the context of the user. Content can be regarded as any kind of non-computational media such as web pages, photos or video. Although the end result might appear quite simplistic, since the system does not feature any dynamic reconfiguration of functionality, the logging of context and content authoring topics can be interesting and are relevant to the work in this thesis.

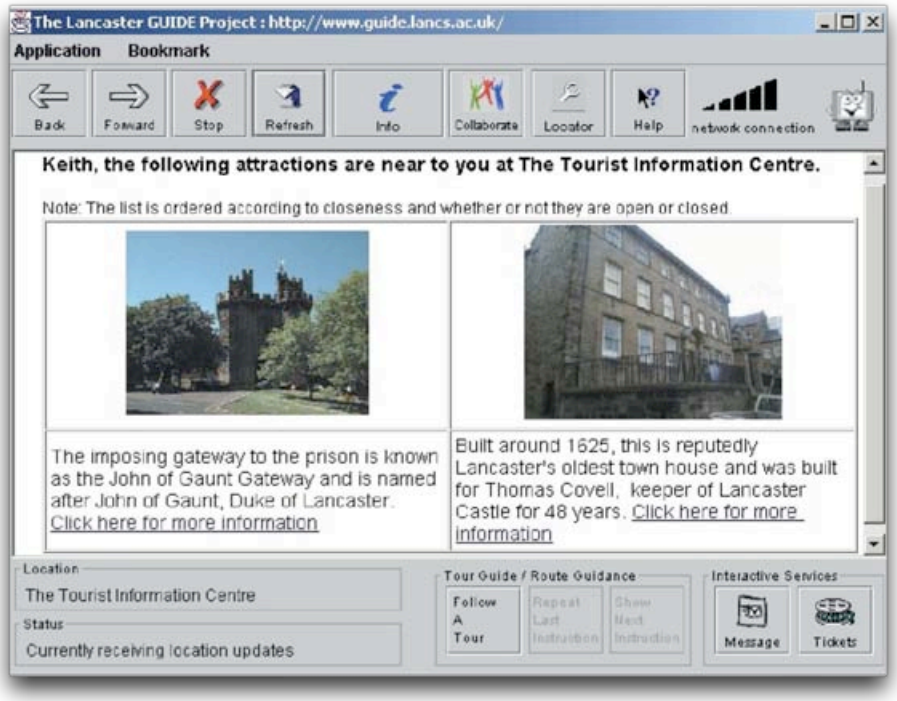
MovieLens Unplugged [59] attempts to learn what movie genres and actors a user enjoys and deliver appropriate recommendations when the user is actually in the store. One caveat of these systems is that they require a period where the user must spend time entering a profile of his or her own particular interests, which the system can draw on to generate custom or adapted content.

The requirement of a lengthy set-up process can negatively influence potential users. This negativity is increased for mobile devices if or when they are perceived simply as accessories to a main digital hub (desktop computer). Imagining such devices are capable of delivering only reduced, lower quality information and functionality to what they are used to elsewhere, many do not anticipate a large return for their time. Although it is likely that utilising the ideas previously discussed to implement more relevant and useful mobile applications may gradually alter users' perceptions of mobile devices, it is currently the case that many users will simply choose to disregard any mobile application that does require any time-consuming set-up period.

Another system, *Hippie*, attempts to customise information delivered to museum visitors, using location tracking as well as records of prior web browsing. When a museum visitor views a display, *Hippie* attempts to present information based on a record of what displays and related information the visitor had seen before, either in the museum or previously [60].

Schiele et al. describe another museum system [61] which users wear a mobile system with a video camera attached, and can associate recordings of tour guide descriptions and speeches with particular exhibits by clicking a button to activate the camera. Subsequent visitors are then delivered these video presentations through a head-mounted display when they are near the exhibits, and the system recognises, through the video camera, that the painting currently being viewed is the same as one for which a recording has previously been made.

Guide [62] [63] [64] was a tourist information system designed to overcome certain limitations of guidebooks, in that they are general-purpose references and consequently contain a significant amount of information of little relevance to the interests of a particular individual. Guide ran on a tablet PC and offered tailoring of information to the interests of the visitor by using context awareness and 'adaptive hypermedia', i.e. dynamic content. Driving the content adaptation was personal context information consisting of the visitor's current GPS location and a personal profile. Web pages and photos of nearby attractions deemed relevant to the user were displayed using a simple algorithm that compares nearby attractions with the list of previously visited ones (Figure 27).



Keith, the following attractions are near to you at The Tourist Information Centre.

Note: The list is ordered according to closeness and whether or not they are open or closed.

The imposing gateway to the prison is known as the John of Gaunt Gateway and is named after John of Gaunt, Duke of Lancaster. Click here for more information

Built around 1625, this is reputedly Lancaster's oldest town house and was built for Thomas Covell, keeper of Lancaster Castle for 48 years. Click here for more information

Location: The Tourist Information Centre

Status: Currently receiving location updates

Figure 27: The *Guide* user interface displaying a nearby location recommendation.

Guide had two limitations on its use. Similar to the MovieLens system, a user's profile needed to be filled in prior to the use of the system. This profile was required to contain personal details such as age and sex, interests, and a history of landmarks visited previously. The second issue is that all of the content presented by the system had to be entered by the designers of the system, severely limiting its scalability. This issue of pre-authored content also arose with regard to the *Lighthouse* in Section 2.2.1, and was addressed in the George Square tourist system by supporting user-created content such as web pages and photographs, woven into the system by analysing histories of user context and activity, presented in Section 2.2.

5.1.2 Displays and user interfaces

Going beyond the content displayed in an interface, one can consider adapting the interface itself. There are many issues concerning adapting the graphical user interface (GUI) of one application to work on many different devices, with different screen sizes and input methods [65]. For example, some work has responded to the way that GUI components designed for PCs with mice and keyboards are not optimal for use on finger-based interfaces such as kiosks or phones, where fingers obscure the user's view and impede the ability to make accurate selections [66]. When discussing adaptive user interfaces there are two application areas to examine. Firstly there are traditional applications that may run on a variety of devices, and there are web applications that may be displayed on a variety of web browsing platforms—ranging from a mobile phone's primitive WAP browser to a desktop with a fully featured browser. With web applications, things are slightly simpler since the web browser can report its screen dimensions and browser capabilities to the server when it makes the request, and the server can perform a translation such as XSLT to transform the page into a format suitable for the destination browser. However, working with traditional applications in a disconnected environment is much more complicated. With the advent of cross-platform software using virtual machines (VM) such as Java and Microsoft .NET, and these VMs being implemented on many different types of hardware including phones, this problem has become even more prevalent. One solution that avoids multiple static versions of the UI is to store abstract descriptions of the interface as meta-data, and then intelligently render the UI at run time by selecting components and lay outs specific to the current device. The interface descriptions are commonly defined in XML documents of which there are many specification languages, for example, User Interface Mark-up Language[24] (UIML), Extensible User-Interface Language[25] (XUL), Extensible Interface Mark-up Language[26] (XIML), and the Extensible Application Mark-up Language[27] (XAML). XUL is most popularly known for being the foundation of the Firefox[28] web browser and XAML was recently developed by Microsoft to be used for designing applications built using the Windows Presentation Foundation[29] in .NET Framework 3.

Crow [67] describes a 'malleable interface' in which collaboration between the system and the interface increases the system's knowledge about the user to aid future communication. Such systems should better support users' needs, and may even offer better efficiency in terms of the user's task:

> *"A malleable interface is a user interface whose behaviour is easily or transparently adapted by each individual user to support the tasks it can perform."*

Crow's implementation is DB_Habits. By observing the users' behaviour it discovers the tasks they regularly perform, and then makes the tasks available to the user as new commands. It works by monitoring the sequences of low level commands and abstracts them into higher level tasks, and adapts the interface to add GUI controls that provide the tasks as macro functions.

As new versions of software are released their user interfaces tend to become more cluttered, with more buttons and menus. Although new features normally provide benefits, users tend to only use a small subset of them [68]. Also, users can often be hindered by the complexity of an interface if frequently used

[24] http://www.uiml.org
[25] http://www.mozilla.org/projects/xul
[26] http://www.ximl.org
[27] http://msdn2.microsoft.com/en-us/library/ms752059.aspx
[28] http://www.mozilla.com/firefox
[29] http://msdn2.microsoft.com/en-us/netframework/aa663321.aspx

items are located amongst unused ones. This suggests a need for an interface that adapts and therefore can be personalised for each individual user.

The motivation for system adaptation is generally to improve the user experience. However unless it is implemented in a friendly or appropriate way, it is possible that a system constantly changing behaviour will cause frustration, and thus degrade the user experience. From a user's view, adaptation can be handled in many ways including allowing the user to choose whether to accept any new features, and allowing to reject them and roll back to a previous state. Also, if the user did prefer a more automated technique then the system should choose the time of adaptation well, so as to not interfere with what the user is doing. One might speculate that a training wheels [69] approach could be applied to an adaptive system. Slowly introducing application features and tools might allow for incremental understanding is more beneficial than being overloaded with choices and features at the start. If an adaptive system were constantly changing would there ever be a manual that could describe how to use it? Perhaps a minimal manual [70] would be effective in this situation, that only holds the basic information a user requires to get started, and through intuitive adaptations the system is learned through use. Another possibility would be to display samples of a new software module's past use to show the current person how it can be used. If the new module was a UI component a pop up example video might be a good approach.

Findlater & McGrenere [71] made a significant step towards understanding appropriate choices as to the autonomy and management of adaptation. They defined two main types of personalised interfaces: *adaptive* and *adaptable*. Adaptive interfaces are instigated by the system whereas adaptable interfaces are user-driven. In order to present an adaptive interface, the system logs which features are used most and using this information it dynamically changes the interface so they are more easily accessible. If there is a set of functions from which a higher-level task can be revealed, then a new interface component can be added to allow this macro-like task to be executed.

Findlater & McGrenere performed a controlled lab experiment comparing the efficiency of static, adaptive and adaptable menu techniques. The static menu was found to be significantly faster than the adaptive menu. The adaptable menu was found to be significantly faster than the adaptive menu when the users had seen and compared the menus, and understood the value of customisation. The majority of users preferred the adaptable menu overall. Even though more users preferred the adaptable menu to the adaptive menu, the users who preferred the adaptive expressed strong support for it. Findlater & McGrenere suggest combining the two in a 'mixed initiative' design may be the best way to satisfy a wide range of users.

Mixed initiative interfaces, where the system and the user both control some of the interaction, are designed to overcome some of the limitations of fully automated techniques. The Microsoft Paper Clip is an example of such an interface, and Horvitz reviews key challenges and opportunities for building mixed-initiative user interfaces that enable users and intelligent agents to collaborate efficiently [72]. Problems with automated interface techniques include "poor guessing about the goals and needs of users, inadequate consideration of the costs and benefits of automated action, poor timing of action, and inadequate attention to opportunities that allow a user to guide the invocation of automated services." Horvitz also considers decision-making under uncertainty and mentions a technique for examining probabilities and assessing costs and benefits of actions. Horvitz denotes 12 critical factors for integrating automated services with direct manipulation interfaces including considering uncertainty about user's goals, learning by observation and examining costs and benefits. Birnbaum [73] examines various techniques of incorporating artificial intelligence capabilities such as reasoning or learning into user interfaces.

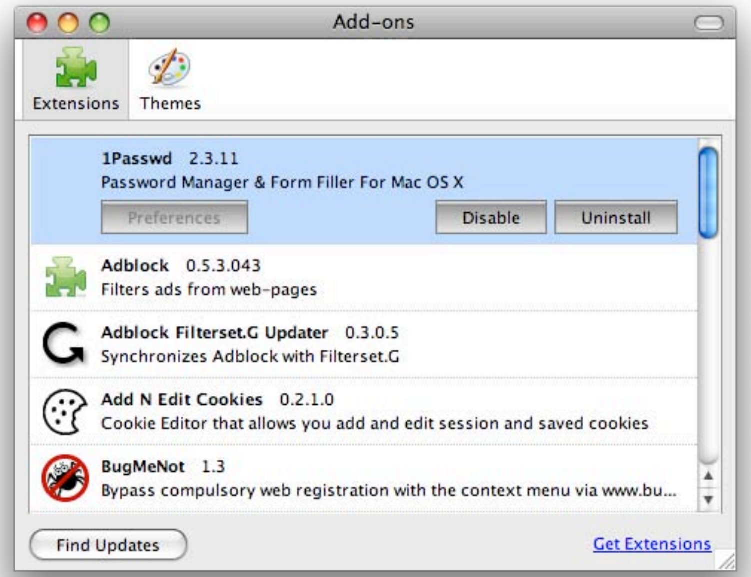
Eager [74] demonstrated a very simple form of adaptation, namely automatic macro generation. Usually macro recordings need to be started and stopped manually, thus users must decide themselves if a task is worthy of being recorded, and decide its extent. If the macro system is implemented badly, it may require more effort to learn and use the macro system than simply repeat the steps of the task manually a few times. Eager attempts to address this issue by continually monitoring the user's actions and if it detects a repetitive activity, the Eager notification, in the form of an avatar, pops up on the screen. Eager then anticipates what you are going to do: it uses green highlighting to mark what it thinks you are about to do. For example, Eager will turn a button green if it expects you to click on the button. So, as you perform your task, you can see whether Eager knows how to perform the task for you. Once you are confident that Eager knows what to do, you click on it, and it auto-generates a macro that completes the task automatically. Eager is a 'programming by demonstration' system — it is able to detect patterns in a user's actions, and to write a computer program that automates those actions. It can be considered a smart macro recorder because it does not have to be turned on it detects repetitive patterns automatically, and it is able to make generalisations.

Intelligent user interface research has shown it to be advantageous to consider helping users rapidly locate files in their folder hierarchies. *FolderPredictor* [75] applies a cost-sensitive prediction algorithm to the user's previous file access information to predict the next folder that will be accessed and reduces the time of locating a file by an average of 50%. FolderPredictor meshes with the existing interface for opening files on the Windows platform, limiting its obtrusiveness. This is an important feature as the effectiveness of an intelligent tool can be undermined if it distracts too much or requires too many resources, to keep the user working efficiently.

5.1.3 Individuals adapting software: Plug-ins and related systems

In the previous section it was mentioned that user interfaces can incorporate techniques allowing them to be adapted to different displays, and how interfaces can be made more efficient and streamlined to certain users' needs. In addition to users having these conveniences in the systems they use, they also often seek to extend the functionality of the software they use in a more general way. The extended functionality they seek can often be as subtle as an advert blocker for their web browser, but may be complex, for example, to build a development environment for an entirely different programming language than the one an IDE was designed for example, Coda[30] is an IDE for JavaScript and web design which is built on Eclipse, a Java IDE.

There are many software applications that allow individual users to manually add extra functionality through plug-ins. The Mozilla Firefox web browser supports extensions as shown in Figure 28. One popular extension for Firefox is an ad blocker that eliminates adverts from web pages. Microsoft's Visual Studio development environment similarly allows the use of 'add-ins'.



1Passwd 2.3.11
Password Manager & Form Filler For Mac OS X

Adblock 0.5.3.043
Filters ads from web-pages

Adblock Filterset.G Updater 0.3.0.5
Synchronizes Adblock with Filterset.G

Add N Edit Cookies 0.2.1.0
Cookie Editor that allows you add and edit session and saved cookies

BugMeNot 1.3
Bypass compulsory web registration with the context menu via www.bu...

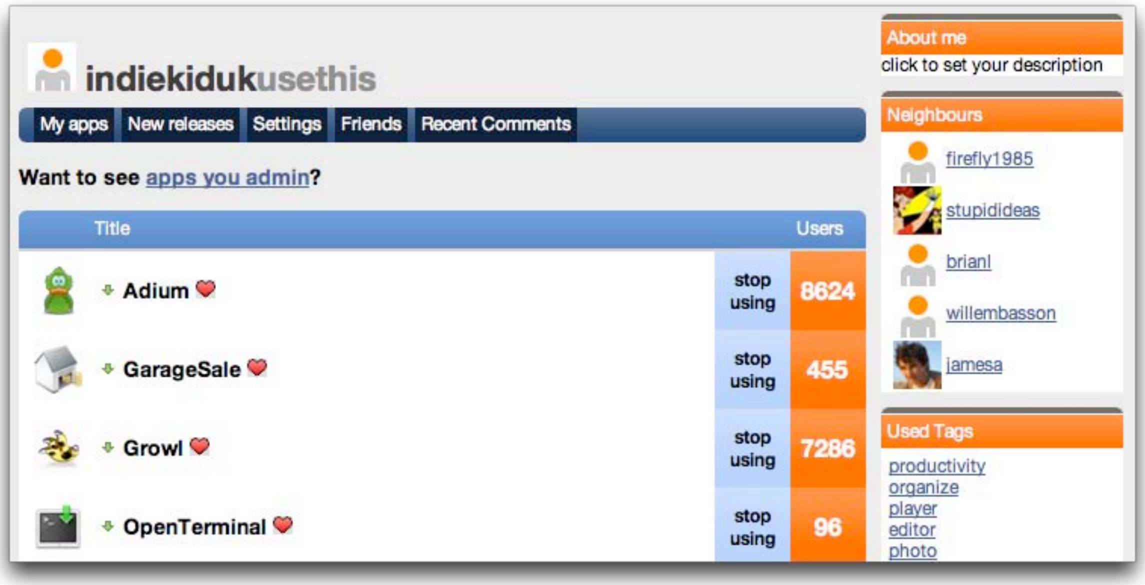
Figure 28: The Firefox web browser has the ability to add features using extensions.

Developers sometimes strongly feel the need to extend and adapt existing applications, as it can feel a waste of effort to re-implement things. The Application Enhancement (APE) toolkit for OS X was created to allow the creation of plug-ins for applications that were not originally designed to support plug-ins or have a plug-in architecture. It makes it easy to extend software that was not designed to be extendable.

It is worth noting that once again the choice as to the plug-ins to add or remove, and the workload of making such adaptations, is essentially left to the user. IUseThis and Wakoopa[31] shown in Figure 29 and Figure 30 are online communities where users can create a profile of the software they use – in Wakoopa tracking software automates this. Then, recommendations for new software the user might like are made using comparisons with other users in the community. Also, the entire logs of closely matched users can be viewed, which is useful for finding highly specific applications used by people of a certain profession. These services recommend whole applications, not plug-ins, and the process requires the manual installation of them. In fact, there are little or no tools that automatically support the process of finding,

[30] http://www.panic.com/coda
[31] http://osx.iusethis.com and http://wakoopa.com

installing and removing software or plug-ins. Also, there is little in the design of plug-in systems that takes advantage of the way that the process of adapting software may be influenced by social interaction among individuals, which we turn to in the next section.



Want to see apps you admin?

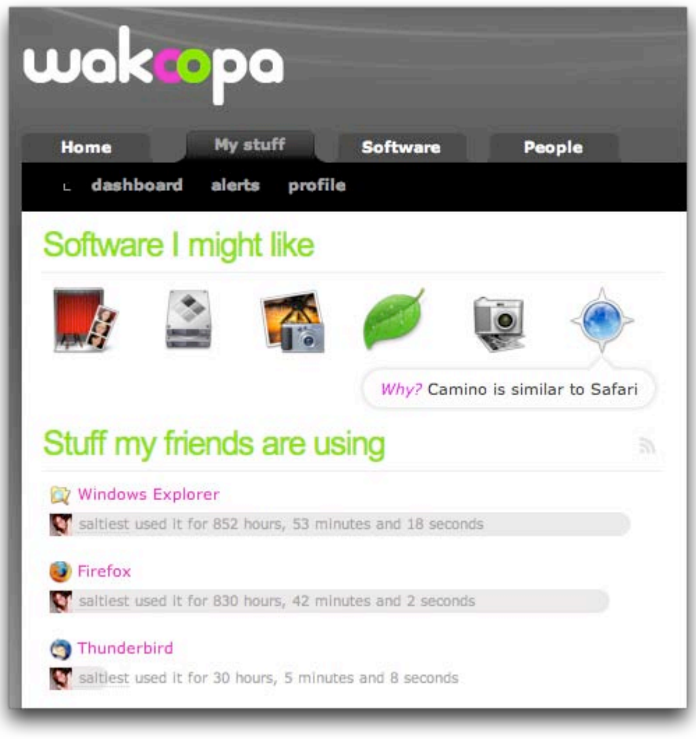
Title		Users
Adium	stop using	8624
GarageSale	stop using	455
Growl	stop using	7286
OpenTerminal	stop using	96

click to set your description

firefly1985
stupidideas
brianl
willembasson
jamesa

productivity
organize
player
editor
photo

Figure 29: IUseThis.com allows users to discover the software used by other people with similar software installed to theirs (neighbours).



Software I might like

Why? Camino is similar to Safari

Stuff my friends are using

Windows Explorer

saltiest used it for 852 hours, 53 minutes and 18 seconds

Firefox

saltiest used it for 830 hours, 42 minutes and 2 seconds

Thunderbird

saltiest used it for 30 hours, 5 minutes and 8 seconds

Figure 30: Wakoopa.com uses tracking software to log the use of applications, then submits this information to a web site to give recommendations for other applications based on comparisons with other users.

5.1.4 People's patterns of sharing software

In large organisations, software distribution is generally more centralised with regard to its source and the choice of what software to use. For example, such organisations tend to automate the installing of the same software on all machines, or have large servers with the software installed which thin clients can access. Personal machines are different and also even more businesses are becoming more flexible and decentralised nowadays, opening up new avenues for sharing software between people.

Mackay [76] performed a landmark early study of how people within an organisation share their modifications of customizable software, and revealed the social aspects of the apparently asocial practice of this simple form of software adaptation. An individual would normally customise software for reasons of personal taste or efficiency. However individuals with perhaps less time or desire to customise will look to others for customisation ideas and inspirations. This sharing process can have many benefits including individuals can experience how others work, perhaps learning new techniques or innovations, and time spent learning how to customise can be reduced so more can be spent getting projects and tasks done. Mackay discovered that by analysing the spread of UNIX customisation files within an organisation, the structure of the organisation's communication networks could be visualised. The highly skilled software engineers were usually the first to experiment with new customisations and were proactive in making their files available. Another group were less technically skilled but would modify those customisations to benefit the actual needs of the third and largest group: colleagues with the least technical skills. Mackay declares the following requirements concerning the design of customisable software:

> *1. The ability to browse through others' useful ideas,*
> *2. Better mechanisms for sharing customisations,*
> *3. Methods of finding out which customisations are used and effective, and*
> *4. Methods of identifying customisations that are ineffective.*

Mackay continues by outlining a design implication from these requirements:

> *"Reflective software should increase the user's awareness of how they actually use the software. Techniques used to instrument software for feedback to user interface researchers may be useful here."*

Note the use of 'reflective' here is different from computational reflection. The term reflective software here describes an information channel for allowing users to discover their own patterns of use, and as a beneficial consequence, it allows users to monitor their personal efficiencies. Furthermore, by making users aware of the features they use most, they can be far more explicit to others about the particular software they find most useful. To achieve this software would have to be instrumented, and as logging is likely to be fine-grained, then raw log data would need to be summarised and presented in an easily understandable format in the user interface. Presenting this underlying system information coherently is quite a challenging task, and one the seamful design technique presented earlier in Chapter 3 handles – that is, software designed to expose its underlying infrastructure in a meaningful way, to allow for better understanding of how works and how one might use it. This is also an issue in the Castles game, presented later in the thesis in Section 6.3.2, where information about software component usage logs was transferred between player's systems, and by displaying when, from who and how many logs were received, they could make better informed decisions about recommendations which were being generated from the logs.

The behaviour of sharing more efficient practices is not just useful for customisation files but is important in all aspects of humans' interaction with computers and with each other. For example, individuals may share experiences with using newer more powerful software, or using the best combination of software to fulfil a task. And when time pressure becomes a factor, people are often eager to ask for advice on the most efficient way to get something done. It may be useful to consider how computers could provide automated support for this kind of behaviour. For example, an application could monitor how useful people find it, and then it could be recommended to others who are in similar circumstances, i.e. who would probably also benefit from its use or (at least) from knowing about it. In fact, it is conceivable that software might actually be able to find the people whom it would most benefit.

One of the early landmarks in the study of collaboration in software adaptation centred on the Buttons system [77], in which modules were shared via email, and could be activated individually within the Xerox Lisp desktop environment. Users could make small changes to buttons, generally by setting parameters via pop-up menus, but deeper changes and integration of buttons were feasible only for experienced programmers. This 'tailoring culture' performed by the more experienced users mirrors MacKay's findings.

5.1.5 Recombinant computing

Recombinant computing is a design technique that allows communications between computational entities with limited prior knowledge of one another – to simplify configuration by end users [78] [79]. Speakeasy [80] is an example of such a system which is "designed to support ad hoc, end user configurations of hardware and software, and provides patterns for data exchange, user control, discovery of new services and devices, and contextual awareness." This relies on three key elements: a small set of fixed domain–independent interfaces that modules can use to initiate communication, mobile code that allows for dynamic extension of functionality to meet possibly unforeseen requirements, and "user–in–the–loop" interaction that accepts that users will be the ultimate arbiters who decide when and whether an interaction among compatible entities occurs. Speakeasy relies on contextual metadata, in the form of predefined name/value pairs, which is used in describing the semantics of each component to a potential user. Such descriptions also supported users' editing of task templates, changing or setting parameters. Speakeasy focuses on supporting users in handling a relatively small number of components associated with devices and related services in the local context, filtering on the basis of known locations, owners and other contextual features, but its "information filtering was only static—components did not update their contextual information, and the organization of components was not responsive to the user's current context". Edwards et al. [80] stated that "a more dynamic approach to information filtering, in which the organization presented to the user is tailored to the user's location, history, and tasks, could prove useful". A limitation of Speakeasy is the constraints on the fixed interfaces modules use to communicate.

Jigsaw [3] was another user configurable system similar to SpeakEasy. However, it differed in that it focused on allowing users to understand the arrangements of connected sensors, devices and services, for example a doorbell, SMS sender, camera and display. Jigsaw allows users to configure ubiquitous domestic environments using an editor, based on a jigsaw metaphor, to make connections between components more intuitive (Figure 31). Connecting jigsaw pieces together works by dragging a particular piece to a fitting target piece. The components communicate using the Equip shared database, also used in George Square explained (see 2.2.4). Whilst this approach is less constraining than the mechanism used in SpeakEasy, a large amount of effort is required to build the Equip wrapper classes responsible for submitting the events and parameter data objects into the dataspace, and retrieving them back out to be use in the destination component. Also, Jigsaw's end–user adaptation relied on a simple set of categories of transformations between physical effects and digital effects, designed a priori rather than adapting dynamically with use.

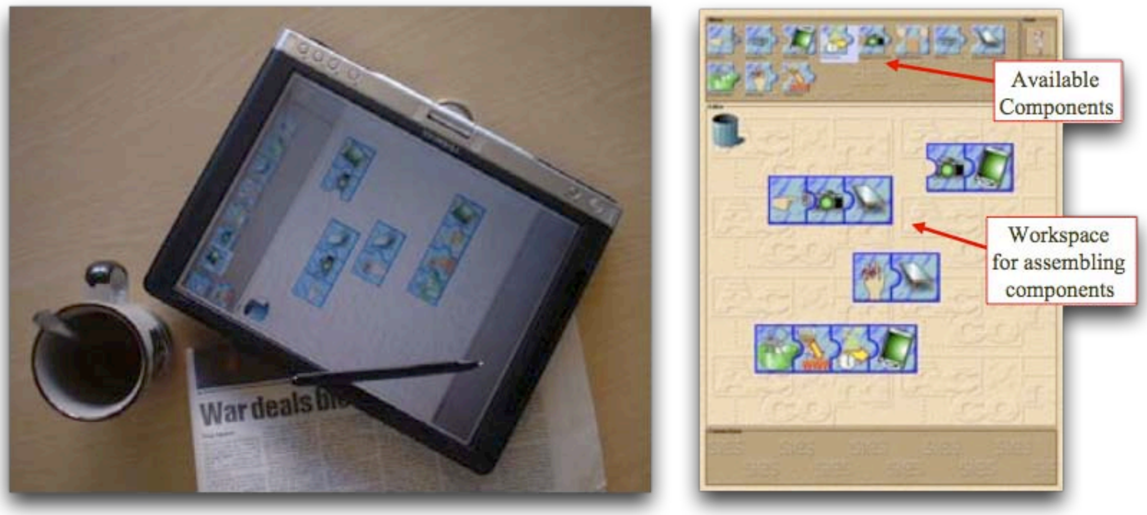


Figure 31: The Jigsaw user interface for connecting jigsaw pieces representing sensors, devices and services in ubiquitous domestic environments.

Kim et al. [81] designed an even more flexible configuration technique where if a component did not contain the correct interfaces to allow the main system to use it, it could dynamically build a wrapper which translates the calls from the system into the form expected by the new component. The motivation was that black box reuse is difficult to apply in some application developments, and if the component interfaces are different then component 'reusers' require a method for connecting components to support more seamless reuse. Conventionally, wrapping techniques result in increased size of components as they are continually wrapped to include new functions. Kim et al.'s technique uses adaptation pattern components so only one wrapping is necessary.

5.1.6 Self-adaptive and Self-healing Software

Self-adaptive software modifies its own behaviour in response to changes in its operating environment. The operating environment means anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation. Researchers working in this area, like those in ubicomp, consider that distributed systems need to evolve as human needs change, technology changes and the application environment changes—although they tend to focus on greater system autonomy.

Change can either be a modification of a currently available function, or the addition of new ones. These evolutionary changes can be difficult to accommodate, as they cannot be planned for at the system design stage. Consequently, we would like systems to be sufficiently flexible to permit arbitrary, incremental change. Kramer [82] believes "systems should be capable of supporting such change *dynamically* without interrupting the processing of those parts of the system which are not, directly affected." Kramer describes an approach where dynamic change can be specified, managed and controlled by distinguishing between functional and structural concerns.

Oriezy et al. [83] propose an Architecture Evolution Manager (AEM) that mediates operations that affect the architectural model. There is a change specification consisting of basic operations or more sophisticated change transactions composed of server basic ones. All transactions are atomic in that a "change transaction includes operations for forcing components into safe or halt states; adding, removing and replacing components and connectors; and changing the architectural topology." Restricting the set of states the system will operate in should allow it to operate more safely in such a dynamic circumstance.

Cheng et al. [84] developed a technique exploiting architecture style for self–repairing systems. They propose a general meta-model of architecture style using a complex system of types, rules and constraints. Exception handlers are described in a formal mathematical way, inappropriate for end users. "Systems may need to adapt, not just because underlying computation base changes, but because user needs change. This will require ways to link user expectations". While running, relevant system properties are logged by "gauges" in the monitoring mechanisms and updated in the Architectural Model. When the gauge value changes, the constraint evaluator re-evaluates the architectural constraints to check for violations. In the event of a violation, the constraint evaluator activates a repair handler that freezes the current state of the architectural model—to prevent subsequent violations from interfering with the present repair. The repair handler then begins running the repair script. During repair, the transaction handler listens for model API calls. If an abort signal is sent, the transaction handler rolls back any changes made to the architectural model, and the error is propagated to a human operator. If no abort occurs, the repair script completes with a commit operation, and all the Model API calls collected are passed to the translator for translation into Environment API calls. The translator utilises the Environment Manager to make the required changes to the Executing System. If there are any exceptions thrown during the Environment operations the transaction handler is informed and subsequently aborts the repair transaction. However, if the Environment manager returns a successful status, then the Transaction Handler commits the repair changes to the model. Whether the repair transaction commits or aborts, the Repair Handler signals to the Constraint Evaluator to resume system monitoring and resets the appropriate gauges. At this point, constraints are re-evaluated to determine whether any violations are now fixed, and the repair cycle completes. If a violation remains then a new violation is detected, the repair is triggered again and the process repeats.

Supporting the modification of a system's behaviour while it is running presents many more challenges than with traditional systems. Runtime extension facilities have become readily available in popular operating systems (e.g. dynamic link libraries in UNIX and Microsoft Windows) and component object models (e.g. dynamic object binding services in CORBA and COM). These facilities enable system evolution without recompilation by allowing new components to be located, loaded, and executed during runtime [85]. Cervantes and Hall [86] discuss their experience in developing a framework for constructing adaptive component-based applications. A limitation is that a very specific and inefficient protocol is required for communication between components, in the form of structured text messages. Modern programming platforms like .NET and Java can support communication between dynamically loaded components using reflection, so methods can be called passing parameters as normal.

5.1.7 Mobile Architectural Requirements

A mobile system running on a device requires communication between inter-system components, but often also, communication with other devices. Mobile systems generally achieve this through a middleware layer that bridges the gap between application program and platform dependency. Middleware is defined as follows by Linthicum [87]:

> *Middleware is an enabling layer of software that resides between the application program and the networked layer of heterogeneous platforms and protocols. It decouples applications from any dependencies on the plumbing layer that consists of heterogeneous operating systems, hardware platforms and communication protocols.*

Since many middleware systems have been created to simplify component based and distributed applications, it may be assumed that existing middleware can be applied to the architecture of particular concern to this thesis: mobile peer-to-peer adaptive systems. However, traditional middleware architectures, like CORBA and Java RMI, fail to provide the appropriate support for mobile applications because they assume applications will run in a static environment [88]. Gaddah & Kunz believe that middleware based on reflection techniques overcomes the limitations of the traditional technologies but there are also limitations of the mobile computing environment that also need to be considered.

Gaddah & Kunz identified three common limitations of mobile computing that affect the design of the middleware infrastructure required for that environment [88]:

1. *Mobile devices*
2. *Network connection*
3. *Physical host mobility*

Mobile devices currently have much more limited processing power than laptops. Also, devices vary from one to another in term of resource availability. Hence, middleware should be designed to achieve optimal resource utilization of whatever is available. For the design of the network connection, mobility is a serious consideration. Mobile devices can move to different areas with no coverage or high interference that will cause connections to disconnect or slow down. Physical host mobility is important because as people move around their device might disconnect from one wireless network and search for an alternative. Or, if no infrastructure networks are available, it might be appropriate to switch to an ad hoc configuration.

After identifying the limitations, Gaddah & Kunz continue to analyse the requirements for a mobile middleware design:

1. *Dynamic reconfiguration*
2. *Adaptivity*
3. *Asynchronous interaction*
4. *Context-awareness*
5. *Lightweight middleware*

Traditional middleware platforms like CORBA are too heavy to run on devices with limited resources. By default, they contain a wide range of optional features and all possible functionalities, many of which will be unused by most applications. In a component based architecture, if the features of the middleware are also components, then additional functionality maybe added to the lightweight architecture at run time. Dynamic adaptation is a crucial requirement for a system that should stay configured to fit a user's context. As a user changes task, it should be possible for the system to generate or acquire a recommendation that suggests components be added or replaced. A history–based recommendation system driving adaptation is investigated further in Section 5.2.

Since communications can break between devices, it is a requirement that they exchange information asynchronously. Similarly multiple devices may make requests to another device simultaneously, thus a multi-threaded asynchronous technique for devices to communicate is required, and a further advantage to this approach is scalability. Section 5.2, on the design of an adaptation architecture for ubicomp, also covers how context-awareness is an essential feature that any relevant middleware must support that at its core.

Gaddah & Kunz claim no existing middleware system support all the requirements identified and there is much room for further research on mobile middleware:

> *There is an urgent need for new solutions that support particular application requirements such as dynamic reconfiguration, context-awareness, and adaptation.*

This argument justifies the design of *Domino*, the novel architecture presented in the following section. *Domino* is reflection-based and satisfies Gaddah & Kunz's requirements and it was designed fully for use in a mobile peer-to-peer system. Although the applications that run on *Domino* are not necessarily fully distributed, the mechanism for distributing contextual history logs, and software components, could be considered middleware in the definition above.

5.2 Domino: An adaptation architecture for ubicomp

This section presents an architectural design directed towards dynamic adaptation so as to fit with users' constantly changing needs and environments, backed up by findings from prior work, systems and experiences. The idea, design and implementation of the system were entirely the work of the author. Domino's design was inspired from background research on other ubiquitous systems, and from personally developing the systems and experiences presented earlier in this thesis.

As mentioned previously, there are many definitions of adaptation with respect to computing. In terms of the architecture presented, the focus here is on the system adapting its structure to better fit with the dynamically evolving needs of the user. Functionality is added, removed or replaced when necessary, and can either be automatic or user driven; that is, the mechanisms here can be used in either adaptive or adaptable systems, to use the distinction of Findlater & McGrenere [71].

Domino's inspiration is drawn from MacKay's study described in 5.1.4, which demonstrated people's practices of sharing software customisations. People do want to improve their efficiency, learn new techniques and innovations. However, personal research into achieving this can be hampered by time constraints, and when this occurs people often look to colleagues or friends for inspiration. Domino could potentially augment this behaviour by automating the process of sharing recommendations between friends and colleagues; and improvements could even traverse the social group boundaries in organisations, as identified by MacKay.

In Chapter 2, successful adaptation was found to only be possible when context is appropriately considered and utilised. However, simply logging software use is not enough contextual information to make successful recommendations for adaptations. Dey and Abowd [8] drew attention to four core context types that they believe are necessary for fully identifying context: location, identity, activity and time. Chalmers' view was that overall contextual interpretation of logs of these information types changes with time because an essential and important part of the significance of current events is their relationship to past events. The *George Square* system successfully demonstrated these concepts*;* history-based contextual recommendations proved successful in adapting media content and removing the need for pre-authoring. Domino could use this technique to log the use of software and provide history-based contextual recommendations for the change in software structure and features. Outcomes of the George Square system included design guidelines for context aware mobile systems and thus will be abided by Domino: avoid reliance on content pre-authored content, information shared among and stored by multiple peers, avoid centralised peer-to-peer architectures.

In Chapter 3, *Treasure* demonstrated that, by exploiting *seamful design*, users could better understand highly technical aspects of infrastructure such as WiFi networks. Similarly, this might be applied to something as complex and dynamic as an adaptive system—which could be beneficial for user understanding and acceptance. For example home automation systems that dynamically control lighting as people enter rooms or sit on chairs have been suspected to cause frustration when they do not work as expected after the novelty factor has worn off [89]. Domino systems could expose aspects of their underlying infrastructure, taking advantage of past patterns of use i.e. finding practical significance in current events though their relationship to past events. For example, when history logs are being exchanged, applications using Domino could be designed seamfully, aiding user understanding by detailing what aspects of past use led to particular adaptations were recommended.

Chapter 4 on mobile peer-to-peer systems introduced the concept of Social Proximity Applications (SPAs) where Persson classified four types: proximity messaging, providing awareness, intelligence of data exchanges, and supporting identity expression. Domino may be viewed as a new, fifth type of SPA in which computational software modules are exchanged between users within social proximity:

- Sharing data to generate intelligent recommendations of software.

This type of adaptive SPA could receive new functionality in the form of software components between collocated users; that is, the exchange of components changes the users' systems when they meet. Thus one's social activity and movement determine how quickly one's system adapts, and collectively drive the mechanism offering changes in functionality.

Also in Chapter 4, the *FarCry* and *Samara* systems demonstrated that mobile peer-to-peer systems could exploit the social proximity of co-located users to drive the distribution of media and generate recommendations of places to go to tourists. Domino could use the same mechanism to distribute the data to drive recommendations for changes in software functionality, and to spread new software components. Furthermore, by utilising social proximity, the *location* of users becomes an inherent aspect of the context considered by the system, as Dey and Abowd articulated it should. This allows Domino to inherently achieve the automatic sharing of software improvements with the social groups who are likely to have the most similar interests, as MacKay's study found.

Combined, these ideas and experiences offer a potential novel approach to overcome the weaknesses of current mobile systems, in that they are mainly static, cut-down versions of desktop software, and offer an opportunity to build successful mobile adaptive systems, which blend with and support users in their dynamically changing activities and environments, thus fitting with the design ideals of ubicomp systems.

Domino's primary goal is providing support for the spread of functionality and content throughout a community of users, in order to improve system usability and content quality for the entire community. First, an example scenario of its use is shown, and then each part of the design is presented, then an overview diagram. Finally, a prototype implementation is presented in Section 5.3.

5.2.1 Example scenario

The following scenario represents what Domino aims to achieve:

James is walking down the street and has his mobile device switched on in his pocket. He enjoys dining out and going to the theatre, and he frequently travels into the city centre by bus to take part in these activities. On his device is a Domino–powered application consisting of a restaurant guide, a list of upcoming theatre shows and a map of bus routes. As James walks down the street, his device discovers another Domino system being carried by someone else nearby. The two systems connect and transfer data between each other. Later in the evening, as he begins to use his device, he notices that he has a recommendation for a module that displays bus time schedules. This module is clearly useful to him and complements his map of bus routes perfectly, and so he installs it and soon makes use of it to plan when to make his journey home. In summary, while James simply went about his day as normal, his device discovered another Domino system, shared data with it, generated module recommendations, loaded new modules, and presented them for James' approval. Most of this adaptation was done without requiring James' explicit interaction, as he only had to handle the choice of which recommendations, if any, to accept.

5.2.2 Communication

As mentioned, Domino's design was inspired by mobile peer-to-peer systems—in particular, social proximity applications. Domino has three aspects to its communication system: nearby user discovery, exchange of contextual histories including software use, and exchange of software functionality.

Domino runs on mobile devices that can be carried with users throughout their day. A typical device will be a mobile phone or PDA, and these devices will need to discover nearby devices whenever possible, using a wireless technology for example Bluetooth or WiFi. If infrastructure wireless networks are available, for example WiFi hotpots, then devices will connect to those networks so peers can be found. If there are no infrastructure networks then the devices will be required to communicate device-to-device, for example using WiFi ad hoc networking. When a Domino peer is discovered, contextual history logs will be exchanged, for example the logs of software use. Domino devices will carry the logs of multiple users and, when sharing logs, the logs of all users will be exchanged, i.e. not just those of the user the request was made to. This part of the design is similar to that of peer-to-peer Recer from 4.3.1. The Domino communication system will also be responsible for the exchange of software modules. Due to the inherent unreliability of ad hoc connections, it cannot be guaranteed that the Domino system that was the source of the recommendation will still be available to service a subsequent module transfer request. This is one of the reasons why Domino maintains a 'wanted module list'.

The transfer of history data and modules when Domino clients meet leads to controlled diffusion that is inspired by the epidemic algorithms of Demers et al. [90], and experimented with in the Far Cry system described in Section 4.2. Popular modules are quickly spread throughout the community, while modules that fulfil more specific needs spread more slowly but are likely eventually to locate a receptive audience because of history-based context matching and the use of 'wanted lists' to find required modules.

5.2.3 Recommendation

Domino’s recommendation system design is inspired by the contextual history-based technique in Chapter 2 and its successful demonstration in *George Square.* An advantage of Recer is that it is generic in its storage format and thus many types of contextual information can be stored. Thus if Domino is required to log other types of contextual information, for example GPS location or the number of peers in range, then it will be possible. The recommendation part of the system will be required to make requests to peers for history logs, via the communication system. When requesting a log, for efficiency, Domino will first check what time-stamp it has received up to already, and then send that as part of the request.

The main functionality of the recommendation system will be to generate recommendations for software modules. The result should be a ranked list of the software modules most likely to be useful at the current time. This will be achieved by selecting the most recent log entries representing the current context, for example, from the last 10 minutes to a day depending on the application, and using that as the context in the recommendation search.

The final requirement of the recommendation system is logging use of software and other contextual information to a database. This logging may vary based on implementation, but it must be flexible enough to support logging of user interface components that the user interacts with, and also non-user interface modules, such as drivers or background tasks.

5.2.4 Adaptation

Once a module recommendation has occurred, and the communication system has retrieved the module, control then passes to the adaptation system. A module’s invocation may either be automatic or the user’s permission will be required. Then it will attempt to dynamically load it into the running configuration.

First, the adaptation system uses reflection to obtain the module’s root class, which implements a simple interface, the Domino Module Interface (DMI). As well as basic start, stop and pause methods, the DMI contains methods for querying and modifying the module’s dependencies and dependants, and a method to expose what types of modules it can support. During development of a module, the programmer must specify the minimal set of modules it is dependent on for successful execution. Since dependencies are defined as type name strings, modules can support multiple dependencies according to the class or interface types its DMI-implementing class inherits from or implements. An example dependency is that a map layer is dependent on a map viewer to display it.

If dependencies cannot be fulfilled then the module is not started, and if there are any named dependencies these are added to a ‘wanted list’ in the communication system. If the dependencies are fulfilled then, subject to any user intervention, the module is started. The final step is that a call is made into the recommendation system to log the use of the module to the database.

5.2.5 Design Overview

In summary, Domino's design involves three sections: Communication, Recommendation and Adaptation. Figure 32 shows the three sections with the inner functions of each, and also the flow of events leading to an adaptation taking place, beginning with a peer being discovered to the system change having taken place, and the new state logged to the history database.

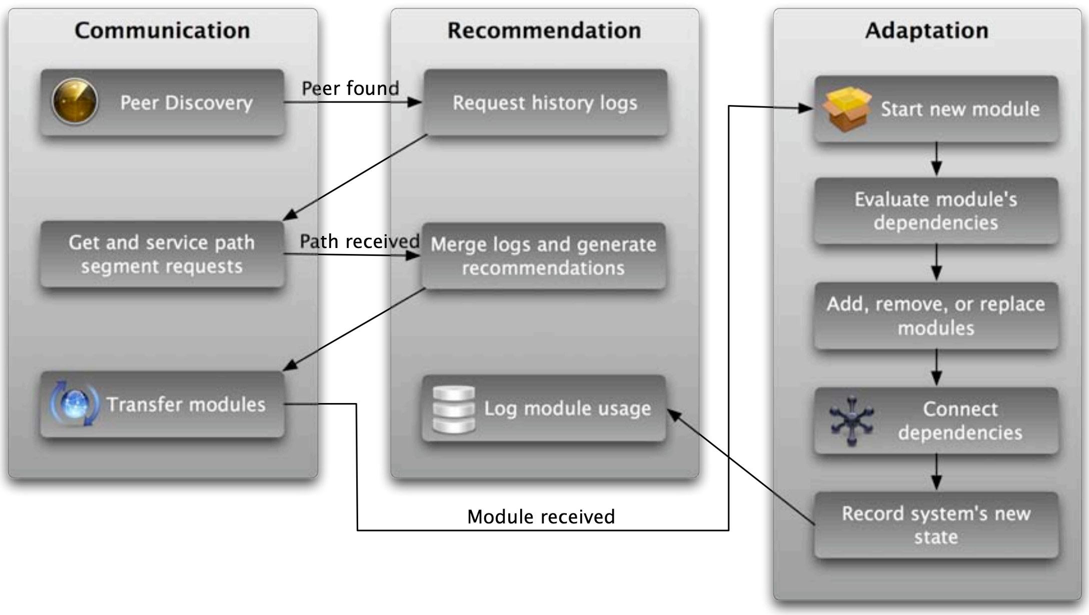


Figure 32: The order of events in the Domino architecture.

5.2.6 Scope

Domino is designed to be as general as possible, without sacrificing ease of implementation of modules by developers. Often adaptive architectures can have inefficient and cumbersome communication protocols between modules. For example in *SpeakEasy*, the modules communicated using text messages of a specific pre-defined protocol [80]. To overcome this weakness, Domino has been designed to support normal function calls across module boundaries allowing modules to be developed as if they were part of a normal static application. For example, a map layer could request the map viewer containing it to redraw by calling its Paint function. However this advantage restricts Domino's generality in that, when developing modules, developers will need to know the interfaces of the dependent modules—although the developer need not know about the modules' implementations. Ideally, the modules of a dynamic ubiquitous system should be able to communicate with no prior knowledge of one another, and through an identified need for combined use, the need for communication has developed, but this surpasses the scope of the current design. Fortunately, functionality to allow more flexible cross-module communications could be added by a future module, that other modules could utilise.

Due to the generic nature of the system model, when a module is received there is no predetermined place for it in the system. In the simplest case, the new module can query the Domino system's running modules to find ones that satisfy its dependencies, by analysing their classes and the interfaces they implement. However, a problem arises when multiple satisfactory modules are found. For example, if there are two map viewers running (i.e. two instances of the same map viewer class), each of which could support a new map layer module, which viewer should the new module be connected to? To resolve such ambiguities, a second use of the history data and the recommendation algorithm is made. By using the new recommended module as the 'context', we can obtain a ranked list of modules previously used in conjunction with it, to determine which is the most likely target. For example, imagine the case where a new 'pollution' layer module is to be added to a system that has two existing map viewers running, one with a traffic layer and the other with a restaurant layer. Through using this technique it becomes possible to determine that the traffic and pollution layers are used in conjunction more often than the pollution and restaurant layer. Thus, Domino would connect the new pollution layer to the viewer that has the traffic layer, where it is likely to be of most value. Alternatively, when starting up a new module, one or more of its dependencies may not be matched. If the required module is available on the system, then a new instance of it can be started up-generating a new check for dependencies and so forth. However, if the required module is not available on the system, the adaptation process for the new module is suspended, and the module is added to the wanted list. The user is informed, and can either drop the recommendation or wait until the wanted module is discovered.

5.3 Implementation

This section describes in detail one prototype implementation of the Domino adaptation framework that conforms to the design presented in the previous section. Of course, other implementations are feasible for other platforms and have more functionality in certain areas. The majority of the chapter is made up of detailed descriptions of how the individual parts of the Domino framework are implemented and how they operate. This covers choosing the implementation platform that includes the choice of device, programming platform, and wireless technology for communications. Following that is the implementation of each of the three sections of the *Domino* design. These are the discovery of peers, *Domino* modules and the adaptation mechanism, and the logging and recommendation system. How these parts operate together to support adaptation is then discussed.

Each instance of the implemented Domino system consists of three distinct parts: handling communication with peers; monitoring, logging and recommending module use; and dynamically installing, loading and executing new modules. We refer to the items that Domino exchanges with peers and dynamically loads and installs as modules. A module consists of a group of .NET classes that are stored in a Dynamic Link Library (DLL) that provides a convenient package for transporting the module from one system to another. Each Domino system continually monitors and logs what combination of modules it is running. When one Domino system discovers another, the two begin exchanging logs of usage history. This exchange allows each system to compare its history with those of others, in order to create recommendations about which new modules a user may be interested in. Recommended modules are then transferred in DLL format between the systems. Recommendations that the user accepts are dynamically installed, loaded and executed by Domino. This constant discovery and installation of new modules at runtime allows a Domino system to adapt and grow continually around a user's usage habits.

5.3.1 Selecting the device

The implementation of Domino was targeted at current popular mobile technology in view of the fact that the overall aim was to develop a fully useable solution on the smallest form factor possible, to be trialled with many users. The requirements for a suitable device to prototype Domino were:

1. *Small form factor*
2. *Wireless networking*
3. *Development tools*
4. *APIs for controlling 802.11*

Therefore the choice of devices readily available at the time consisted of Windows Mobiles and Nokia Smartphones.

There are two types of Windows Mobile device — Smartphone and Pocket PC. The simplest and most evident difference between the Windows platforms is that Pocket PC devices have a touch screen, where as Smartphones do not, however both types can have hardware keyboards. Both types of platform can be either GSM (Global System for Mobile Communications) or CDMA (Code Division Multiple Access) mobile phones with either GPRS (General Packet Radio Service) or 3G data connections. GSM is the mobile phone standard in Europe. It is becoming more popular in USA, but CDMA is the standard currently used by the majority of devices in USA. The operating system is the same, which is based on Windows CE 5, however the two platform variations differ in their user interface, since Smartphone is optimised for key input as opposed to touchscreen input. If designed well, for example if all functionality is exposed through keyboard presses, the same executable can run well on both platforms. Windows Mobiles typically have a 200Mhz-675Mhz CPU, 64Mb RAM, 64Mb ROM, WiFi and Bluetooth, and they offer memory expansion via SD memory cards up to a size of 8GB. More recent devices also have cameras and GPS built in.

Nokia Smartphones run Nokia's Symbian operating system. At the time of choosing a platform for Domino, Nokia phones did not have WiFi and only made use of Bluetooth and GPRS. Symbian is a frequently updated operating system. Usually with every new model of phone, a new version of Symbian is released; this can make supporting every device troublesome. Symbian phones can run both native and Java applications, however access to the phone's actual built in functionality can be quite restricting. For example any time an application requires the use of the GPRS connection, the operating system prompts the user and the dialog must be accepted before the application can continue. Recent Nokia devices are more powerful and have built in WiFi and some include GPS.

The quality of development tools is of high importance when choosing the mobile device platform. Because mobile devices have limited performance compared to desktop computers, it is not always viable to build software on a desktop and then run it on the device, as previously undetected run time errors might develop in the low performance situation. To support this, some platforms offer emulators that allow the software to run in a limited fashion, however the native wireless functionality of the device is not offered on emulators. So, for a platform like Domino that requires control of the WiFi feature, a lot of the development has to be done using the actual device. Therefore it is particularly important that development tools exist that allow developers to compile code, transfer it to the device, execute the code, and also support live debugging with break points, all on the real device, with activation from a single key stroke. This is why Microsoft are currently leading in this field as their Visual Studio IDE supports all of these requirements. Although Windows Mobile is a proprietary operating system, sufficient APIs and SDKs are available to give a very high level of control of the device. Furthermore, the wide availability of Windows Mobile has given rise to a very large development community.

Given that the development tools offered by the Windows Mobile platform met more of the criteria for this project, compared to the other operating system development tools mentioned, and very small powerful devices existed incorporating 802.11, which was controllable through software, Windows Mobile was selected for this implementation of Domino. The HP iPAQ 2750 was chosen as the first target device, which is a Windows Mobile Pocket PC, with 320x240 resolution touch screen, 675Mhz CPU, 128Mb RAM, WiFi, Bluetooth, Compact Flash and SD expansion slots, and running Pocket PC 2003. There have been two operating system upgrades since then, Windows Mobile 5, and Windows Mobile 6, however both were minor upgrades and software designed for earlier systems continues to run on the new devices.

Since the completion of the Domino implementation, four notable new mobile platforms have become available. The first is OpenMoko[32] that is a fully open source touch screen mobile phone and development platform. As an open source platform, OpenMoko provides developers with the ability to examine and control all of the phone functions, which is a huge advantage, however given that OpenMoko is still a relative newcomer in the field, the development tools are not of high quality yet.

The second new platform launched can be seen in Nokia's N770 and N800 WiFi tablet devices that run the open source Maemo Linux distribution. This is another open source operating system with a vibrant and growing development community, however as with OpenMoko, the development tools are lacking and have not yet reached the levels of quality of the IDE offered by Microsoft. Additionally, given that another requirement of Domino is small form factor, the N770 and N800 are not appropriate as they are quite large (twice the size of many other mobile devices). Lastly, the N770 and N800 do not include any

[32] http://www.openmoko.org

phone functionality, as their primary purpose is to provide users with Internet access. This is a major disadvantage, as users need to carry another device in order to make phone calls when mobile.

The third platform is the recently announced Android platform by Google[33]. The SDK and emulator are available to use, but no phone hardware has been released yet. Android is portrayed as an open source operating system, however not in the sense of Maemo and OpenMoko. Android's functions are exposed by an open source JVM and a Java SDK is supplied, but its native functions such as WiFi scanning and phone functions are private. Google's aim is to provide a platform where the same JVM can be implemented across many types of hardware meaning the same applications can run on different devices, however this might lead to frustration amongst developers. As previously stated, Windows Mobile devices come in two variants (Pocket PC and Smartphone), and that already provides a fairly complex development scenario, so if Google plan to support many more variants, the Android software may become simply too complex to be worthwhile developing for, and thus software may be simplified to only support one particular type of phone, which negates Google's stated benefit of the platform.

The fourth and final most interesting new mobile platform is Apple's iPhone[34]. The iPhone is the first mobile phone to run a powerful desktop class operating system, namely Apple's OS X. It is also the first fully finger-based (rather than stylus) multi-touch mobile device with WiFi, Bluetooth, GPRS, 8GB storage, 64Mb Ram, 400Mhz CPU. A unique feature is a dedicated graphics processor. High quality responsiveness is the main feature that makes using a touch screen a great experience, and the iPhone reacts instantly to touches, and can even detect two fingers at once, which allows for unique interactions such as pinching for zooming the interface. The current SDK offered by Apple is only for Internet browser based applications, however an SDK and IDE for native applications is due in February 2008. Because it runs the OS X operating system, existing libraries and utilities can be recompiled for the phone's ARM processor. The iPhone's architecture has an impressive memory bandwidth and in tests it can download over WiFi at 750K/s. In contrast, current Windows Mobile device architectures have a 'bottle neck' in their memory bandwidth and download at a significantly lower maximum of 100K/s. Because of the power of the iPhone and its features, and its potential for utilising existing libraries, it could become a very prominent platform for researchers in the future.

5.3.2 Selecting the programming platform

When implementing a reusable architecture, the choice of programming platform is crucial, and even more crucial when the target is a mobile device. On Windows Mobile there are three significant programming platform choices for implementing *Domino*. They are Microsoft .NET, Java and native C++. Each of them has advantages and disadvantages explained below, and the choice for Domino is then justified.

[33] http://code.google.com/android
[34] http://www.apple.com/iphone

Native C++ offers the highest performance and most straightforward access to native phone functions however it is a difficult language to develop in. Writing code can be quite error prone and development time can be significantly longer than in the other choices. Native C++ applications are developed in Visual Studio that includes a form designer and debugger, and one button to compile, deploy and execute the application on the mobile device or emulator. Domino's adaptation system requires components to be integrated dynamically, while the system is running. Although this is possible in native C++, it would require a significant amount of extra effort and therefore longer development time compared to the other choices.

Java development is possible on Windows Mobile, although the difficulty in choosing a Java Virtual Machine (JVM) is a significant issue. There is no longer an official JVM by Sun, instead there are several 3rd party JVMs. The best two are IBM's J9 JVM and NSICOM's CrEme JVM. There are several other JVMs on the market, but they all suffer from various problems. Sun's PersonalJava is no longer supported, both SuperWaba and EWE are not fully Java compatible, Esmertec's JBed is on sale to OEMs only, and Insignia Jeode is out of date[35]. One advantage of utilising Java is the wealth of libraries available designed for desktop computing, however porting these to the cut down JVMs for mobiles can be time consuming if even possible. From experience of building a mobile system on the CrEme platform [91] there were a few barriers to smooth development. The JVM was only implemented to Java specification 1.3 and thus was lacking several major features. Performance and memory management was poor and applications would tend to simply quit without any errors. Access to native phone functions is possible on the CreMe JVM however this is a very complicated process involving writing extra native code to bridge between Java and native code in the form of JNI (Java Native Interface) libraries. For these reasons, Java is a less popular choice for development on Windows Mobile nowadays and is perhaps not a good choice for building a reusable library like *Domino*.

Microsoft's .NET framework is available in a slimmed down version designed for mobile devices named the Compact Framework. MS .NET is similar to Java in that it is a high level object oriented platform that runs in a virtual machine—CLR (Common Language Runtime). MS .NET applications can be written in C#, J# (similar to Java), Visual Basic and C++. Most of the core .NET functionality is available in the Compact Framework and allows use of the majority of windows mobile UI components, access to databases, networking, simplified web services. So far the Compact Framework has been released in three versions. CF1 was the most basic and was quite limiting, however missing functionality was implemented by a community driven open source project called OpenNETCF[36], which for example allowed serial port access and more advanced drawing to screen. CF2 brought more functionality and the majority of OpenNETCF became redundant, and the main improvement was dynamic user interface layouts so applications displayed well on different screen resolutions. This was an important development as there is an ever-increasing list of windows mobile screen resolutions, currently: 176x226, 320x240, 240x240 and 640x480. Furthermore, most devices can switch from a portrait to landscape display, and CF2 allows for

[35] http://blog.vikdavid.com/2004/12/java_on_pocketp.html
[36] http://www.opennetcf.org

automatic re-layout when this happens. CF 3.5 brought compression and Windows Communication Foundation (WCF) which is a very simple to use remote procedure call technique for communications between devices and servers. A very powerful feature of CF is its ability to call native libraries directly without any bridging code; this is made possible by the Marshal classes in the .NET framework that are responsible for converting data from the managed CLR memory to unmanaged, and handle data type conversion automatically. This allows CF software to utilise native device features such as WiFi and Bluetooth scanning very easily. CF supports simplified access to SQL CE, a database for Windows Mobile devices, and powerful data binding classes for linking data structures and user interface components to data in a database. CF has reflection libraries that allow dynamically loading classes from DLLs at run time. Finally, .NET applications are developed in Visual Studio that includes a form designer and debugger, and one button to compile, deploy and execute the application on the mobile device or emulator. All of these features make .NET CF the best choice currently for mobile device development for research software.

In summary, small form factor is offered by most of the options discussed but high quality development tools and easy access to built-in functionality such as 802.11, whilst the ability to modify software structure dynamically at runtime is only realistically available through the Windows Mobile platform using the Microsoft .NET framework.

5.3.3 Communication

As mentioned in the design section, the communication segment of Domino is required to discover nearby peers wirelessly, exchange history logs, and transfer modules. The Windows Mobile devices chosen for the implementation offer communication through Bluetooth and WiFi, although other similar models also support 2G (GPRS and EDGE) and 3G (UMTS and HSDPA) communications. A comparison of these technologies is shown in Figure 34.

Technology	Frequency (GHz)	Cost	Power consumption (active) (mW)	Range	Theoretical Max (KB/s)	Laptop Max (KB/s)	Mobile Device Max (KB/s)
GPRS	0.85-1.9	High	1400	long	5	5	5
EDGE	0.85-1.9	High	1400	long	15	15	15
UMTS	2.1	High	1700	long	48	48	48
HSDPA	2.6	High	1700	long	460.8	204.8	128
Bluetooth 1	2.4	None	120	10m	54	53	53
802.11G	2.4	None	990-1600	300m	6912	3328	128

Figure 33: Comparison of current wireless technologies for mobile devices. Power statistics are from [92]. Speed tests are by this author.

It is interesting to see that mobile devices do not reach speeds close to the theoretical maximums of the faster technologies. Primarily, this is due to memory bus constraints on current phones. Furthermore, if the data downloaded is to be stored in flash memory then the speed required to write to such storage can reduce the speed even further.

Domino requires the use of local network connections, for example Bluetooth or WiFi, rather than long distance connections such as GPRS or UMTS. Firstly, local connections intrinsically offer relevance filtering by location. That is, those geographically proximate to a user are more likely to be carrying software components that are of relevance to that particular user. This is due to the fact that people tend to spend much of their time in close proximity with friends and work colleagues, who obviously have similar social or work interests. Additionally, local 802.11 and Bluetooth connections can work anywhere, even when a phone is out of cell coverage; for example, in 'dead zones' such as the subway [93]. Another significant advantage is that WiFi and Bluetooth are free to use for local ad hoc connections, whilst for mobile devices longer-range connections can be extremely expensive if large amounts of data are to be exchanged. Furthermore, local WiFi connections offer significant bandwidth and speed improvements over longer-range technologies. This is particularly important to the work in this thesis because Domino's design relies on spreading data epidemically, and so it often wishes to transfer large amounts of data. One final disadvantage of long-range cellular connections is that it is usual for a single, paid account per device with a SIM card. For example, a mobile phone or PCMCIA data card (for laptop use) both have a single SIM card, and the operator expects that there will only be a single user and that all their operator services will be delivered over this single device. If devices are to become more adaptive and do indeed begin sharing content between peers on a frequent basis, regardless of where they are, then operators could support usage models supporting multiple connection points per person or per household. An interesting new device is the Amazon Kindle e-book reader[37], which is the first device to have a free 3G data connection with no contractual obligation to an operator.

If connections such as 3G data services become cheaper and more flexible in the future, then it may become prudent to use them, as they would greatly increase the population of available peers from which information can be cultivated. However, if this does occur it would be prudent to ensure a privacy system offering a suitable degree of anonymity for users interacting with each other be created, as currently these connections may expose personal data, such as phone numbers.

Although Bluetooth consumes a relatively low amount power, it lacks the speed of WiFi. Furthermore, it suffers from a severe problem in that when a device is in discovery mode—that is, when it is attempting to seek for peer Bluetooth devices—it continually hops between radio frequencies. As a result, it is extremely unlikely that a device in discovery mode could ever be detected by another, as there is only a slim chance that two such devices would both be using the same frequency simultaneously. Indeed, as the chance of this occurring is so small, the majority of Bluetooth devices simply completely stop advertising their existence for the duration they are in discovery mode. As a result Bluetooth devices are a poor choice for a peer-to-peer technology if the system requires devices to be continually scanning for peers.

[37] http://www.amazon.com/gp/product/B000FI73MA

Due to the advantages and disadvantages listed, WiFi was chosen for communications for most of the research described in this thesis. By utilising the WiFi and the peer discovery implementations in *Treasure, Feeding Yoshi* and *StreetHawk* it could become the ideal peer discovery component for Domino.

To summarise the evolution of the WiFi discovery driver in this thesis, the initial design began with a driver to automatically lock a WiFi device onto a particular infrastructure network in Treasure to increase reliability. This was then improved in Feeding Yoshi to support ad hoc connections between mobile players out with the range of infrastructure networks. At this stage the Self Discovering Spaces (SDS) component was added to enable applications to discover each other and communicate once the devices were connected to WiFi. In Feeding Yoshi it was used to support the swapping of fruit between players. FarCry and Samara also utilised this driver and SDS, and an additional component was added to support reliable epidemic transfers of information. Finally, Streethawk added the ability to detect infrastructure networks that have Internet connections. Although the current design of Domino does not have a requirement for Internet access, it could potentially use this access in the future, for example to validate the authenticity of modules.

The final implementation of the WiFi peer discovery component in Domino, involves two parts. The connection to a physical network (ad hoc or infrastructure), and the discovery of Domino peers on the connected network. In wireless networks, the same physical network means that devices use the same SSID and network mode (ad hoc or infrastructure). This allows them to communicate using 802.11 Ethernet frames, which the application layer IP protocol runs on top of, and which Domino peers use.

Existing MANET systems provide no requirement for devices to auto configure their setup for ad hoc networking. In the majority of research, it is assumed the device has been preconfigured with a set SSID and IP Address before use by the end-user, and thus is ready for applications to communicate. Furthermore, it is often assumed all devices are in range of each other when experiments start. Obviously, this is rarely the case if users have their own devices and are fully mobile during their normal day. More commonly, devices would be set to a specific SSID when not in range of any others. As will be seen shortly, this can lead to a protocol failure that causes devices to be unable to communicate with one another when they are in range, even though they have the same ad hoc SSID set. This was the case in Hocman, in which devices frequently failed to communicate with one another when riders passed each other on the road. However, as will be seen, the systems in this thesis were designed to overcome this particular problem.

WiFi was designed as a replacement for Ethernet cables, and works in a very similar way that it robustly transmits Ethernet frames that encapsulate IP packets. Thus, in theory, applications can transmit data wirelessly using TCP or UDP as if they were using a wired connection without any specific WiFi code.

In a decentralised wireless ad hoc network, there is no DHCP server to control the distribution of IP address configurations, so devices automatically assign their own addresses. In other words, it is not possible to ensure the IP addresses are unique and on the same subnet, which applications depend on for successful communications. This issue is addressed by the Dynamic Configuration of IPv4 Link-Local Addresses protocol. This is implemented on Windows Mobile devices, and provides the mechanism for devices to self-allocate unique IP addresses to all peers on an ad hoc network. However, in practice it can take up to a minute for an address to be self-configured. Domino overcomes this by generating a static IP address from the device's serial number, and takes 3 seconds (because Windows Mobile require a power cycle of the WiFi device).

The Hocman [50] system enabled motorcyclists to exchange personal web pages of information whilst riding. Self-configured addressing at connection time was too slow for what Hocman required. So in the implementation of Hocman it was necessary to pre-configure each device to be constantly in ad hoc mode with a predefined static IP address in a specific subnet and constantly associated with a specific network SSID. The application can do this when it is run. Due to the problems explained in configuring IP addresses, some might assume that applications should exchange data at a lower level in the networking stack, and implement a custom protocol. However, this would greatly increase the complexity in the application, and would require the re-implementation of a great deal of the functionality offered by IP protocols; for example fault tolerance in TCP and broadcasting in UDP. The extra work required to implement and test this might outweigh any advantage gained. The authors of [94] point out that:

Although the technology for achieving MANET is well studied, applications of it are rare

Indeed, Hocman is one of the first widely known systems to successfully utilise 802.11 MANETs as the communication technology of a mobile, peer-to-peer application. The authors also state:

Operation without an infrastructure fits well with biking since traffic encounters can take place anywhere

However, this statement can be extended to most mobile systems, as the location of devices, and the likelihood of these devices being near an infrastructure node when they encounter one another, cannot easily be predicted.

In Hocman, whilst the MANET configuration worked well in the system, there was a problem with the MANET configuration in Hocman that was not discussed in published work but is known to exist from conversations and experiments with the authors. This was that the reliability of devices being able to successfully communicate with one another in the trials was far lower than expected. The most likely cause of this was due to constraints in the wireless device drivers that existed at the time Hocman was trialled. Devices at that time commonly had to be in range of one another when they were first set to ad hoc mode or they would not be able to meet. This constraint was caused by the fact that setting a device to ad hoc mode actually created a hidden BSSID that was used as a fake infrastructure node two peers in range could both address. As this BSSID was created randomly, if the two devices were not in range of one another when set in ad hoc mode they would be forced to generate separate IDs and thus not be able to communicate over the same ID when in range, as the protocol rejects any messages not received from the BSSID that matches that currently stored. The 802.11 specification defines that if ad hoc devices with the same SSID move in range of each other then all devices should associate with the same simulated BSSID. However, in practice, depending on the WiFi device and driver user, this is not always the case.

By using only local connections, rather than transmitting information over the Internet, no personal data about users is ever sent through a third party, and there are no logs that communications ever took place. However, whilst this threat is removed, others are introduced. For example, peers could be designed to disseminate malicious logs, resulting in the recommendation of components that would not typically be used, and which may have been designed to perform malicious attacks. That is, a malicious user could create fake logs in an attempt to get their own virus distributed amongst a peer community. Although complete security of Domino was not considered as a major feature of its design, and this implementation does not attempt to provide a fully secure system, its vulnerabilities are acknowledged and possible the issues and possible solutions are examined in detail in Section 6.6.

Awareness of peers is critical to Domino, and having a variety of up-to-date log data from these peers is key to the recommendation system's operation. When connected to a network, be it fixed or wireless, the SDS component within Domino repeatedly sends out packets containing an IP address and port number on which it can accept connections from other Domino systems. This allows any other Domino systems on the same network to discover, connect to, and request and receive history data and modules quickly from other peers. In order to maximise opportunities for encounter, peers continually attempt to meet on a certain network, and will consistently switch to one appropriate network. Domino systems running on devices with wireless connectivity actively seek out infrastructure mode networks and connect to them whenever possible. When no networks are available, Domino switches to its own ad hoc network. These features allow Domino systems to contact each other even when no 802.11 infrastructure mode networks are present, while still permitting users to use infrastructure access points, i.e. hotspots, to connect to the Internet as they normally would. In our trials the custom wireless driver code was extremely quick in carrying out the required switching between networks and network modes. Typical times involved with Domino's 802.11 connections are as follows:

- Switching between infrastructure mode and ad hoc mode: 1ms
- Associating with an infrastructure access point: 3s
- Time to acquire an IP address via DHCP for infrastructure: 5s
- Time to set IP address for ad hoc: 3s
- Discovering a peer after joining a network: 1s

The DHCP time for infrastructure is highly variable depending on the quality of signal to the access point and the number of users on the network. When in ad hoc mode we decided to assign static IP addresses as we found automatic private addressing to be slow and unreliable. It should be noted that the above times are taken from the moment a Domino system makes the decision to switch to another network. In our code we typically made this decision after trying but failing to reconnect to the previous network four times with a period of 250ms between each attempt. Thus, the effective total duration for switching from one infrastructure network to another typically was 9s ±5s.

This 'network discovery followed by service discovery' approach has advantages over IP-based discovery protocols such as ZeroConf[38], which only provide the latter service. Searching for other networks and discovering new clients continues even while connected and transmitting data over a network. This behaviour results in nearby Domino systems being able to locate each other in most situations. Indeed, unless the network card is required to be exclusively locked to another application, a Domino system is likely to locate another nearby in a matter of seconds.

The UDP packet that each Domino system broadcasts every second holds an IP address, a port number and a unique ID for the device. In order to protect a Domino user's privacy, his or her own username, actual device ID or MAC address are never used as identifiers in any of the data transmitted over the network. Instead, each user can choose one of two types of anonymous ID. Firstly, when Domino is initially run on a system, a random number can be generated and permanently stored on the device to be used as the ID in all subsequent Domino transactions. As the ID is randomly generated the user's anonymity is preserved. The main advantage of this technique is that if two or more sets of data are exchanged with a peer at different times then the receiver, although not able to identify the actual user, will be able to identify that the data comes from the same source and so will subsequently be able to determine more exact recommendation weightings for the entries. For example, if a new set of data is received and shows a moderate similarity to the current Domino user, the likelihood of it being recommended would high. However, if it was found that, in addition to this new set, previous data had been received from this user in the past, the chance of recommendation could be significantly higher.

The alternative ID that can be used is simply a random number generated for each transaction with a peer. Whilst this technique is the most efficient at protecting the originator's identity, it does result in it being impossible to determine if two different data sets came from the same source. However, the recommendation system mainly relies on finding similarities within short windows and, as these windows are commonly far smaller than any set of data transferred in a typical Domino transaction, this method actually has little impact on the quality of the overall recommendations.

When one Domino system receives the UDP packet broadcast by another, it can use the information contained therein to act as a client and create a TCP connection to the advertised IP address and port. Thus, the systems temporarily assume the traditional client/server roles. The most commonly used

[38] http://www.zeroconf.org

requests in our systems so far are to list the users for whom one has history data, to send the N most recent history entries for user X, and to send N history entries starting from the Mth most recent entry for user X. These three request types allow a client to identify which histories are available on the server, to begin obtaining the most recent history data and then to continue to gather more data as time allows. As connections can be lost at any time, a request generally consists of a single message, and we parse incoming streams so that we can make use of most of the data received up to the point when the connection was lost. As all connections are threaded and handled separately, each Domino system can act as a server and a client simultaneously. Indeed, this is the typical behaviour for Domino systems, as they will normally discover each other at approximately the same time. Each Domino system also contains a lightweight implementation of an FTP client and server for the exchange of modules.

When utilising existing communication technologies on the phone its important to be careful not to disrupt the activities users may carry out in the normal use of their device. The majority of mobile phones will not be running existing background services that utilise the WiFi connection, so an appropriate implementation would be for Domino to control WiFi when the device is not in use. This can be achieved on Windows Mobile devices by an API method that checks if the device is idle. This is defined as a state when keys have not been pressed for some time and the screen has gone off.

5.3.4 Recommendation

The recommendation of modules is implemented using the peer-to-peer version of Recer presented in Section 4.3.1 because of its simplicity and ability to log in a general format. This was previously proven successful in Samara, a tourist application for recommending places to go. The advantage of using the Recer collaborative filtering algorithm is it uses a generic logging format of a time stamp, plain text field, and another plain text field specifying the entry type. For peer-to-peer Recer to be used in Domino, the format of the log entry had to be chosen, a design for how module use would be logged was needed, and when recommendations should be generated needed to be decided.

Time	Id	User	Type	Symbol	ExtraData
13/11/2006 15:10	5106	AndyC	Domino	Barracks	barracks.dll
13/11/2006 15:13	5107	AndyC	Domino	Barracks	barracks.dll
13/11/2006 15:13	5108	AndyC	Domino	MaintainedForest	maintainedforest.dll
13/11/2006 15:13	5109	AndyC	Domino	MaintainedForest	maintainedforest.dll
13/11/2006 15:13	5110	AndyC	Domino	Sawmill	sawmill.dll
13/11/2006 15:17	5111	AndyC	Domino	Barracks	barracks.dll
13/11/2006 15:17	5112	AndyC	Domino	MaintainedForest	maintainedforest.dll
13/11/2006 15:17	5113	AndyC	Domino	MaintainedForest	maintainedforest.dll
13/11/2006 15:17	5114	AndyC	Domino	Sawmill	sawmill.dll

Figure 34: Sample data from the UserPath Recer database table in a Domino application.

The Time, Id, and User field are standard in any Recer application. The Time field is the date and time the entry was created. The Id is an identity field used to overcome the limitation that multiple entries may occur at exactly the same time, so an Id is used to distinguish them. The username is The Type field identifies the format of the logging used. Here, we use the type name 'Domino' to identify that this row's format of the final two fields are specified for a Domino application. The Symbol field contains the Domino module name, for example this could be GPS Driver, or if the modules were even more fine-grained it may be a .NET class name. Finally, the ExtraData field is utilised to specify the .NET DLL that contains the module named by the Symbol field, because their names may differ or there could potentially be multiple modules within a DLL. A sample of the Recer UserPath database table can be seen in Figure 34.

Whenever a module is activated or deactivated, the entire set of all running modules is logged to the history database. Also, within a Domino module there is the ability to log a usage event with a simple Log function call. Domino knows by reflection the context from where Log was called, for example the current user and current module, so it has enough information to send the new entry to Recer. Thus it is up to the designer of the module to decide when the suitable time is to log usage. In the case of user interface modules, for example a map layer of icons within a mapping application, it might be suitable to log whenever an icon is clicked. For non-UI modules, internal events could trigger logging, for example a GPS driver could log when a GPS fix is obtained. For modules that have no visual interactivity, and are not suitable for logging internal events, an alternative is to simply log a usage event at periodic time intervals. This logging technique is vulnerable to the log function being called too frequently within a module, either accidentally or maliciously, which could lead to the UserPath table being flooded. To combat this, measures to detect and prevent flooding could be implemented, however at present it is assumed the design guidelines have been followed.

By utilising peer-to-peer Recer, each Domino system can carry not only its own user's history but also the histories of many other users. The recommendation system periodically analyses the similarity between the owner's history and all other cached histories. It identifies the most similar histories in terms of overlap in module usage, and stores the IDs associated with their owners. As the more similar users are likely to provide the most relevant recommendations, similar users' histories are the last to be thrown out when storage space is low and the first to be requested from other devices when they meet. The similarity comparisons are carried out as an average of matches per history entry since a basic overlap would unfairly favour longer histories.

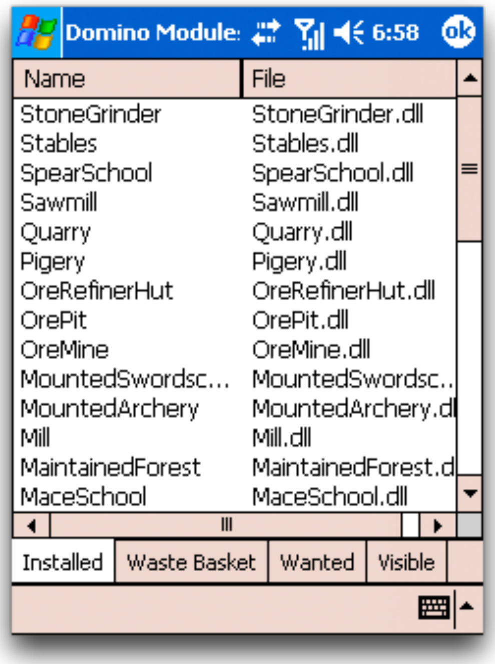
When similar, but not exactly matching, history sections are found, the modules not in the current context are tallied, ranked and delivered as recommendations. New recommendations are generated whenever a module is activated or deactivated, as these changes alter the current context of the user and so may alter the recommendation results—even if no new history data has been created in the interim. Recommendations are also generated when new history data is received from another Domino system, as this is likely to provide novel module recommendations. Finally, recommendations can be generated manually if required, and because the current context can change irrespective of the previous two events occurring, different recommendations will occur.

As shown, Domino has the ability to log all the information required to generate recommendations for new functionality in the form of modules, and trades usage data with peer recommenders on other devices. It is by scanning through this logged history of other users' data and searching for sections that are similar to the current user's current module configuration, i.e. the current context, that recommendations can be generated.

5.3.5 Adaptation

The adaptation component of Domino is responsible for handling new modules, installing modules into the system, determining module dependencies and finally executing modules. All of these tasks are achieved during runtime, allowing for dynamic and ongoing adaptation of *Domino* systems to support users with minimal disruption.

A module in Domino is defined as a bundle of functionality that can be transferred and installed, and each module runs under direct control of a Domino Manager. Managers are responsible for deciding whether to accept a new module, and for loading, installing and running modules. The Manager keeps track of all running modules in a hash table. A default Manager is provided in the *Domino* system, which is fully capable of performing all of these tasks. However, if a developer desires, this default Manager it can be extended in order to create specialised behaviour. For example, after receiving module recommendations, the default Manager automatically installs the new module. If such behaviour was not desired, the Manager class could be edited to bypass this automatic installation and could prompt the user for a final decision to determine whether the module should be installed. In general, Managers will be directly linked to the UI of an application as this allows decisions such as authorisation requests, module removals or when to pause modules to be directly controlled by the user. This allows users to easily remove adaptations and additions that turn out to be less advantageous than expected. In fact, Domino contains a default user interface form for managing modules (Figure 35).



Name	File
StoneGrinder	StoneGrinder.dll
Stables	Stables.dll
SpearSchool	SpearSchool.dll
Sawmill	Sawmill.dll
Quarry	Quarry.dll
Pigery	Pigery.dll
OreRefinerHut	OreRefinerHut.dll
OrePit	OrePit.dll
OreMine	OreMine.dll
MountedSwordsc...	MountedSwordsc..
MountedArchery	MountedArchery.dl
Mill	Mill.dll
MaintainedForest	MaintainedForest.d
MaceSchool	MaceSchool.dll

Figure 35: Default user interface for Domino module management displaying a list with module names and the .NET DLL container name.

The module management user interface displays the four lists of modules: installed, wastebasket, wanted and visible. By holding the stylus on a list item and selecting uninstall from the pop up menu, modules can be uninstalled. Then it enters the wastebasket, which allows the module to be dormant on the device, so that it can be transferred to other systems if requested. When modules are installed and the dependencies are resolved, it can be the case that a required dependency is not found; these modules appear in the wanted list. Finally the visible list displays the entire list of modules running on all devices currently in range and discovered, allowing for manual transfer and installation.

Domino modules themselves can contain many program classes and thus be of any size - from a simple map layer to a complex device driver. The only constraint on modules is that they must conform to the Domino Module Interface (DMI) to allow a manager to control integration with other modules already active on a system. Module developers can easily achieve the required conformity either by implementing the DMI directly or by extending a sample base class that already conforms to the interface. The interface itself is extremely simple and contains only seven method stubs: SetManager, GetDependencies, CanSupport, AddDependent, Start, Pause and Destroy.

When a new module arrives on a Domino system, the first method called is SetManager, which assigns the Manager to the module so the module can query the Manager at its own convenience. If the module has a dependency on another module, for example a handle to a particular instance of an existing object in the system is required, then the module should implement the GetDependencies method. For example, a map layer module may have a dependency on a map viewer. Such dependencies must be identified and coded into the module by its developer, basically ensuring any required fields are set, which is part of normal initialisation in any ordinary class. If any dependencies exist, the Manager tries to fulfil them, first checking if any active modules can support the new addition, which is achieved by calling the CanSupport methods all active modules in the hash table the manager maintains. If any of the modules that are already active can fulfil the dependency, the new module is connected to the one that is already running by a call to its AddDependent method. Checks to CanSupport are necessary, as in certain cases one module may be technically capable of supporting another, however, perhaps due to resource limits or other constraints, be currently unable to fulfil a dependency. For example, a map may be able to support a maximum of 5 map layers, and if it already has 5 connected layers then a call to CanSupport on behalf of yet another new map layer will return *false*. If no currently active modules can support the new module, but the Manager has access to a DLL that contains a module that could fulfil the dependency if active, it will attempt to start a new instance in order to fulfil the dependency. Finally, if all calls to CanSupport fail and the Manager does not have access to a required module DLL on the local device, the Manager will store the type of module that is required (usually an interface name) in a ‘wanted list’ and seek it in future peer encounters. If this occurs, the new module will not be started immediately, but in all future peer encounters the required modules will aggressively be sought and, when available, will be installed and the new module started at that point, if the user of the device still wishes.

By utilising the first four interface methods in this way, a *Domino* Manager can dynamically install a module in to any Domino system. The installation algorithm followed when a new module, *modx,* is received and started by the Manager, is described by the following pseudo code.

```
modx.SetManager(this);
dependencies[] = modx.GetDependencies();
stillRequired[] = new array();

foreach(dependency in dependencies) {
        fulfillers[] = new array();
        foreach (module in activeModules) {
                if (typeof(module) != typeof(dependency)) {
                        continue;
                }
                if (module.CanSupport(typeof(modx)) {
                        fulfillers.add(module);
                }
        }
        if (fulfillers.isEmpty()) { // no modules are currently active which could fulfil the dependency
                search module library on local device for dependency;
                if (dependency is in local library) {
                        newModule = start new instance of module;
                        dependency.fulfil(newModule);
                } else {
                        stillRequired.add(dependency);
                        continue;
                }
        } else if (fulfillers.length == 1) {
                dependency.fulfil(fulfillers[0]);
        } else { // there are multiple modules that can fulfil the dependency
                most_suitable = Run_recer_algorithm_to_find_most_suitable(fulfillers);
                dependency.fulfil(most_suitable);
        }
}

if (all dependencies fulfilled) {
        modx.Start();
} else {
        modx.Destroy();
        add missing dependencies to list of wanted modules;
}
```

The other three methods in the DMI are Start, Pause and Destroy. These simply allow an installed module to be executed and halted as desired. When the Manager first calls a module’s Start method the module should allocate any resources it requires to run and begin execution. Subsequently, if the Manager calls the Pause method, possibly at the behest of the user, the module should halt its execution but not release any resources it holds. Paused modules can be restarted by a call to their Start method that should not reinitialise the entire module if repeatedly called, as resources have already been allocated, but simply resume from the paused state. If the Manager calls the Destroy method, the module should fully stop execution and releases any system resources it holds.

Module removal is currently not supported when dependencies are involved. However a possible solution would involve a further use of GetDependencies to identify any other modules that rely on the module to be removed. These modules could then be listed and displayed to the user, who can decide if he or she still wishes to proceed with the removal. Relying on the user for every module removal may not be the most suitable course of action, as users may be involved in other tasks, and requesting they authorise module removals may be a distraction. Furthermore, users may not have the understanding required to identify how modules are connected, or even the functionality any one module provides. Therefore, an automated removal process may be preferable. Whilst automation of this type could be achieved through a developer creating a customised version of a *Domino* Manager, it may be more appropriate if this were the default behaviour of *Domino*, as requiring a user to understand modules seems in direct opposition with its most fundamental goals. Therefore, the automated removal of modules, and identification and handling of dependent modules, is an item for future work, and is discussed in the Future Work section of Chapter 6.

The adaptation manager, like the other components in *Domino*, is written entirely in C# for the .NET platform. Each module is a .NET class that is wrapped inside a DLL. DLLs are used as they provide a convenient package for transporting code between devices. When a new DLL arrives on a device the Manager uses the .NET reflection capabilities to dynamically instantiate the class at runtime.

5.3.5.1 Dependency resolution

As shown, Domino utilises a recommendation system to recommend potentially relevant and compatible modules to the set in current use. The simplest type of module recommended is one that is not functionally dependent on any other module. For example, calendar and address book applications are useful in combination to those who are keen organisers, however the applications are separate and are not functionally dependent. However when there are functional dependencies required to be fulfilled by recommended modules, there are two categories, and Manager implementations exist to handle both of them.

1. *Dependency on a specific module implementation*
2. *Dependency on a module interface*

The first type of module recommendation is one that has a functional dependence on another known module – ‘dependency on a specific module implementation’, for example a map layer dependent on a specific map viewer (Figure 36). In this case the Manager can query all running modules using the CanSupport method and find a suitable candidate.

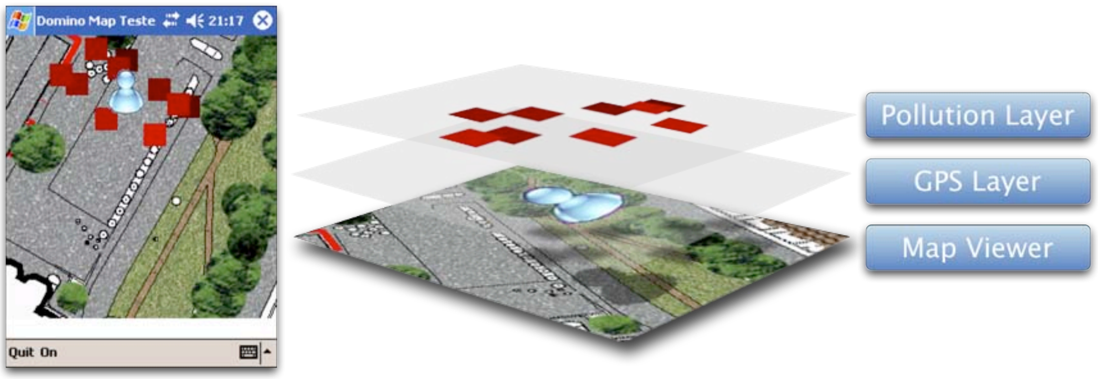


Figure 36: A map layer test application with three Domino modules.

The second type of module recommendation is when there are multiple implementations of a certain kind of module, where the function names are known, but the underlying implementations differ – 'dependency on a module interface', for example there exists multiple map viewer modules that all confirm to a known map viewer interface, which a map layer modules depend on.

A problem that can arise in both situations is when there are multiple dependency candidates in the running Domino system. For example, if there are two map viewers running in the application, and both could support the recommended map layer module, it may be only suitable to add the layer to a certain one. Fortunately Domino is in a unique situation to solve this problem through a novel use of the same algorithm that produced the original recommendation.

As previously mentioned, windows of context within the UserPath table in the Recer database produce the ranking of modules which when ordered by rank, produce a list of weighted recommendations. Domino can further utilise Recer to disambiguate between multiple dependency possibilities. By using the recommended module as the sole context in a recommendation search, a ranked list of dependency possibilities can be found. The result can be used to query the active dependencies of the original dependency candidates, and this allows a decision to be made, based on the history of past use. For example, consider an application with two map viewers, and one of those map viewers has a layer displaying the user's GPS location. Another application with one map viewer has been previously used with both the GPS layer and a pollution layer and Domino has logged this usage, thus has modelled the relationship in the form of the Recer database. When the pollution layer is recommended to the application with two map viewers, the dependency search is likely to result in the pollution layer being placed in the map viewer that contains the active GPS layer. This scenario has been tested successfully using the Domino Map Tester – the first implemented Domino application pictured in Figure 37 and Figure 38. A link to a video of this application can be found in Appendix A – Online materials.

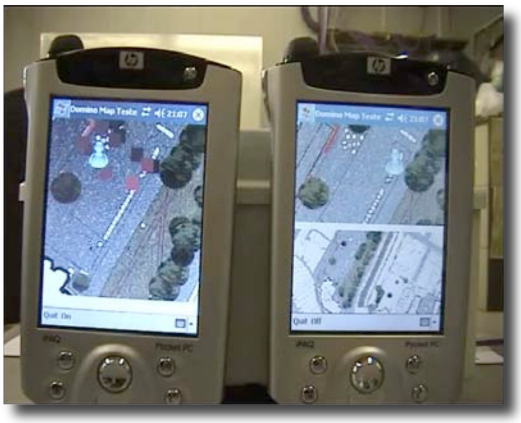


Figure 37: Domino map layer test application. One device has a map viewer with two layers: pollution and GPS. Another device has two viewers and at present only the GPS layer.

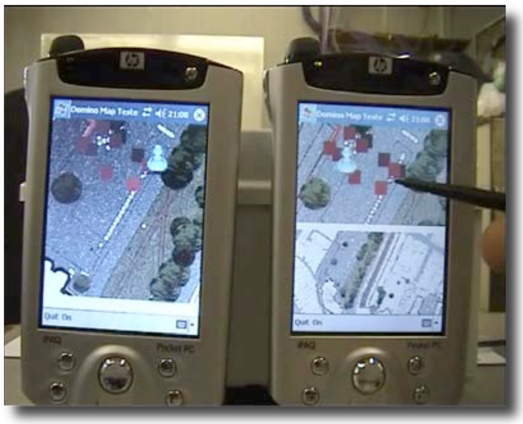


Figure 38: The pollution layer has been transferred from the PDA on the left to the PDA on the right, and is displaying in the map viewer containing the GPS layer, which it has been known to be used with.

5.4 Conclusion

In summary, this chapter began with a survey of relevant literature in the field of dynamic software adaptation. The term 'adaptation' is widely used in computing and can mean a variety of different things. Three different types of adaptation were examined in this chapter: adaptive content—the content of a web page, for example, is customised depending on the context of the user, adaptive displays and user interfaces—the actual interface itself is adapted as opposed to the content, and personal adaptation of software—users often seek to extend the functionality of the software they use in a more general way.

Most importantly in this chapter, the core system of this thesis was introduced—Domino. Domino's design was inspired both from background work on adaptation, and from developing the systems and experiences presented earlier in this thesis. Domino manages and supports adaptation of the set of software components within an ubicomp application. The Domino architecture actively supports incremental adaptation based on a user's activities, needs and interests. The word 'adaptation', in terms of the architecture presented here, refers to an adaptation of software structure to meet the dynamically evolving needs of the user. It takes utilises the infrastructure components described earlier in this thesis, for peer discovery, peer–to-peer recommendations, and epidemic spreading of components and usage histories between users. It tracks and logs the current system 'context' in terms of the set of components currently running in the application. This set is used to filter usage histories in the course of making recommendations of new components to install and run. Therefore, instead of relying solely on predefined templates or patterns of use, like Speakeasy for example, Domino also takes advantage of emergent patterns of use in recommending and integrating components. This is done in a way that hides from the user much of the technical details of discovery and integration of new software, but reveals enough to let him or her maintain control over the system. When a user accepts a recommendation, Domino checks whether other currently running components satisfy the component's dependencies—required interfaces declared *a priori* by the developer—in an effort to ensure that its execution is technically feasible before trying to install it. If not, it suspends installation until it finds such components, which then can be recommended, accepted by the user, and installed. Dependencies specify objective constraints on component combination, like the connections in Humble's Jigsaw editor and Speakeasy's templates, but Domino goes beyond other work in this area by also taking advantage of the evolving patterns of use that represent users' subjective preferences about component combination.

Domino could potentially work automatically and autonomously, but an expectation inherent in the design is that user intervention and control is part of the process of recommendation and adaptation, i.e. Domino is designed to be part of the sociotechnical process of adaptation, and can take advantage to patterns of use (and hence patterns of adaptation) not foreseen by module designers. In this way, Domino is an example of a more dynamic and adaptive system, in comparison to earlier systems, and is an illustration of a viable solution to RQ2, as discussed later in the conclusion of this thesis.

A link to the implementation source code of all parts of Domino can be found in Appendix A – Online materials.

Chapter 6 Investigation of a Domino application: Castles

To test the Domino architecture and the related user experience a novel mobile strategy game was developed. The design, implementation and evaluation of Castles explore the combination of many of the key features discussed in the preceding chapters. *Castles* is a context aware, peer-to-peer, seamful, adaptive system utilising the Domino implementation presented in the previous chapter. This chapter covers a design iteration involving a prototype, pilot trial, prototype redesign, a final trial, and a discussion of the findings. The pilot trial was primarily a technical one, conducted to ensure that the Domino system behaved as hoped within the game, with peers discovering one another, exchanging recommendation information and modules and offering useful adaptations in line with expectations. The final system overcame the usability and performance issues identified in the previous version, and then was trialled with 16 participants to investigate user experience, acceptance and privacy issues surrounding the use of socially proximate software recommendations and adaptations.

6.1 Why a game?

The study and design of games has added diversity to many areas of ubicomp research. Games have wide social and financial impact, and form an interesting application area in themselves [95], and introduce challenges in terms of designing enjoyable experiences and implementing distributed ubicomp systems. More particularly though, a game was chosen to test Domino because one can design a game to explore specific technical issues raised by wider research, and adapt it with ongoing findings relatively easily. Additionally, players find new ways to stretch one's designs, assumptions and concepts, and are often keen to participate in tests of one's systems. The Treasure [3] and Feeding Yoshi [22] mobile games presented in 2.2.5 were successfully used to investigate the exposure of system infrastructure in a 'seamful' way, so that users might appropriate variations in the infrastructure. *Can You See Me Now* [24] examined earlier in 2.2.5 on *Seamful Design*, led to generalisable results concerning positioning systems and the use of self–reporting.

Real Tournament [26], a simple 'shooter' game, explored IPv6 and issues such as host mobility, security, content delivery and wireless overlay networks:

> *A number of possible architectures exist which are suitable for supporting distributed multi-player gaming environments in fixed networks. We surveyed various groupware and online multiplayer gaming architectures in order to define a suitable model for sharing and communicating data between distributed sets of users. …we focus on the centralized server, peer-to-peer and mirrored server based architectures*

Multiplayer games require complex architectures, and have real-time games have requirements for high performance and reduced latency in network connectivity, both extremely well developed research areas. Games offer an interesting test-bed for real world experiments, complementing the simulation style work this network research often focuses on.

Another unique aspect of games is that people are often keen to play them, and can spend substantial periods of time experimenting with them. Good games are engrossing, to the extent that users are more likely to find ways to stretch designs. From experience of previous systems' trials, it is clear that trials based around a game result in a larger number of willing participants, and an increased and prolonged interest in using and reporting on the system. Players' engagement can lead to new patterns of use that reveal system strengths and weaknesses, in particular the system's ability to adapt to or be adapted to such changing uses and contexts. Games are an application area in which users are already often involved in radical re-engineering (i.e. adaptation) of systems. Many games now have extensive 'modding' communities (such as Half Life[39]) and so the idea of an adaptable or changing application is not one that is alien to game applications or to the people who use them.

6.2 Castles

Castles is similar, in theme, to other strategy games—such as the popular Age of Empires, Stronghold and Settlers games[40]—in which the player must create a building infrastructure, which in turn allows for

[39] http://half-life2.com/ (game information), http://www.planethalflife.com/features/motw/ (modding information)
[40] http://www.microsoft.com/games/empires http://www.stronghold-game.com http://www.settlers4.com

the construction of armies. The majority of the Castles game is played in a solo building mode, during which the player chooses buildings to construct and how many resources to use for each one. The goal of this stage is for the player to create a building infrastructure that efficiently constructs and maintains the player's army units. For example, a player may wish to have many 'Knight' units being produced. However, to achieve this, the player must first ensure that he or she has constructed suitable buildings to produce enough food, iron, stone and wood to build and continually supply a Knights' 'School', the unit that produces 'Knights'. There are two types of buildings players can build—*Shops* and *Producers*. Shops are buildings that, when built on the map, can be clicked on so that a new screen is shown and the player can build (or buy) army units. The shop screen also shows the required and available stock items to build the unit. Producers are buildings that, when placed on the map, begin producing items every game cycle (3 seconds). These buildings will produce their output if their inputs are satisfied. Some buildings, such as the House, have no inputs and produce peons regardless every cycle. Producers can be enabled or disabled by clicking them. Both types of buildings can be demolished if more space is required to build more efficient buildings on the map. When the game starts, there are a wide variety of buildings and army units available to the player, allowing for extremely varied combinations of buildings supporting distinct types of army. For example, one player may wish to have an army consisting mainly of mounted units whilst another may try a strategy of having a large number of ranged units such as archers. Different players begin the game with different sets of buildings.

Castles is played on a mobile device and, when two players' devices are within wireless range, one may choose to attack another. When a battle request is accepted, both players select from their army the troops to enter into battle, positioning their units in three possible locations (front, back and reserves). Players view the battle screen as the battle proceeds, and can view their army depleting and the waves of units moving forward. After the battle, players can talk about the game, or the buildings they have been using and either found useful or discarded. Also after the battle, players might see a notification about newly available buildings, and they return to 'solo building mode' during which they can rebuild or improve their armies using the new buildings.

The buildings in the game are, in fact, Domino modules. In order to mimic the way that plug-ins and components for many software systems continually appear over time, new buildings and units are introduced throughout the game, as upgrades and extensions that spread among players while they interact with each other. As shown in the previous chapter Domino is first required to log the use of Domino modules. When a building is interacted with, for example when a unit is created, an entry is created in the Recer database. During the battle logs of use are exchanged in the background over the ad hoc WiFi network. The recommendation algorithm is run after this data has been received, and the user is notified about new buildings they have 'discovered'.

With such a high number of buildings and units, there is significant variation in the types of society (module configurations) that a player may create. Selecting which buildings to construct next or where to apply building adapters can be a confusing or daunting task. However, Domino helps by finding out about new modules as they become available, recommending which modules to create next, and loading and integrating new modules that the player accepts.

The idea and design behind Castles originated mainly from the author, along with Marek Bell and Scott Sherwood and the actual system was implemented collaboratively.

6.3 Initial implementation

This section describes the first prototype implementation of Castles. The results of testing this prototype led to the modified implementation used in the user trial. As mentioned, Castles is built using the Domino adaptation architecture. In Castles, buildings are C# classes that implement the Domino Module Interface, and are compiled in separate DLLs. For example, the Barracks building in the game is a Domino module which is contained in its own individual DLL ready for transport over the network should any other Domino client request a copy. Castles ran on HP iPAQ hx2750 Windows Mobile Pocket PCs with 128Mb RAM, 624MHz CPU, WiFi and a 320x340 stylus touch screen.

First the basic components of the game had to be implemented. A map component with the ability to pan and zoom in was created, reusing the same map component developed by this author and used in *Treasure.* The stocks, units and buildings in the game were decided, and images found. The units have 4 strength attributes: *melee attack, melee defence, ranged attack and ranged defence*. A range of intuitive values were spread across all unit types, for example, the archer had a high ranged attack but a low melee attack, because its weapon was a bow and arrow. The buildings had *run costs* and *items produced* set, which were also intuitive with respect to their name, for example, the knight school's cost to produce a knight was a swordsman and iron. 36 building types and 11 unit types were created.

Buildings are built by choosing their name from a pop up list, by clicking *B* in Figure 39. Then they are positioned on the map using a building placement tool that ensured buildings were only placed in valid positions. There was a game timer, responsible for periodically looping every 'producer type' building and executing its build cycle. If the building's costs were satisfied these were removed from the global stock list, and the items produced were added.

The Domino implementation described in Section 5.3 was utilised in Castles, however the game had to be designed around the various parts of the Domino system. For Domino to be able to recommend buildings, their use by the players was required to be logged. The moments at which buildings were built on the map were deemed to be good times to trigger the logging of use to the database in the initial version. Also, use events were logged when units were produced using a building.

In this initial implementation Domino was left in fully automatic mode. When peers were found the usage histories were exchanged using the peer-to-peer Recer implementation explained in Section 4.3.1. At the end of an exchange Domino automatically started the recommendation algorithm. The result of the recommendation search is a paired list of *building name* and *rank number*. If a recommended building's module name was not in the list of Domino's installed modules, the system actively sought this module. Then, when found, Domino's adaptation component instantiates the building class, it reports a dependency requirement of a Building List module, and adds it to the list of buildings within the Castles game. Castles then presents a user interface dialogue mentioning a new building is installed, and available to be built and added to the map.

The ranked list of recommendations is cached, and the three buildings that the system most recommends the user construct next, are shown when the user clicks the *R* (recommendation) button (Figure 39). Thus, the user has quick access to guidance from the Domino system about how to proceed.

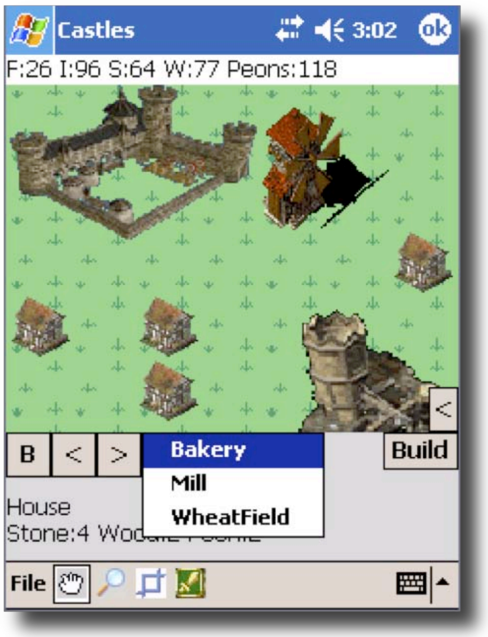
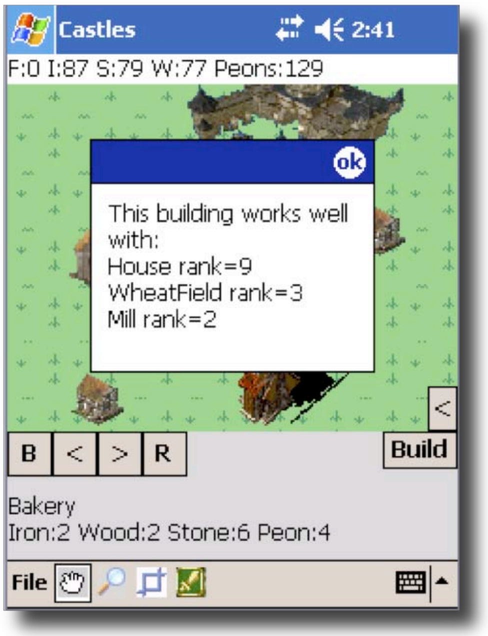


Figure 39: The recommendations show in a pop up when user clicks the *R* button (hidden under pop up).

If the user desires, he or she can get additional information about recommendations, such as its dependencies or the modules most frequently used in conjunction with it in the past in similar contexts. This information is a by-product of the internal work of the recommendation algorithm, and can be obtained in a pop-up dialog by clicking the recommendation information button in the build panel, can help the player understand more fully how the module might be used (Figure 40).



This building works well with:
House rank=9
WheatField rank=3
Mill rank=2

Bakery
Iron:2 Wood:2 Stone:6 Peon:4

Figure 40: Recommendation dialog window in the Castles game. These are the modules a module has been used in combination with previously.

A new module is smoothly integrated into the player's system without requiring substantial module management, or indeed any knowledge of the low-level transfer or installation process. Simply, the user sees the new options and recommendations, and can make use of that information without having to search manually for or install the new modules. It presents them in a way that lets the user see them as he or she plays, find out something of their past use, and show this information to others when meeting and talking with other players. Overall, Domino complements the conversation and discussion among players about new and interesting modules, and eases the introduction of new modules into each individual system and into the community.

6.3.1 Pilot trial

Pilot trials were run and offered some initial findings about the system's use. The game was set up so that four players sat in rooms distant from one another, and out of wireless network range. We periodically moved the players between rooms, so that they passed by each other, and met up in pairs. This meant that users spent most of the time alone but periodically met up to start battles and to talk about the game and its modules, much as they might if they were walking with their mobile phones during a normal day.

Each player started with the same base set of buildings and units available, as well as five extra buildings unique to him or her. For example, amongst the additional items given to one player was the catapult factory. As anticipated, when players met for battle, their Domino systems exchanged usage information and transferred modules between phones so as to be able to satisfy recommendations. Thus, the catapult factory and catapult unit began with one player, but were transferred, installed and run by two of the three other players during the game.

Several players who had been performing poorly because of, for instance, a combination of buildings that was not efficient for constructing large armies, felt more confident and seemed to improve their strategies after encountering other players. They started constructing more useful buildings by following the recommendations. In each of these cases, this did not appear to stem from players' conversation, but directly from the information provided by the system. After the first meeting with another player, the system had gathered its first new history to compare against, and thus it is the first time the player saw recommendations. When the player began to construct a new building, he or she always saw at least one recommendation for which building to construct next and followed it.

Each Domino system's interactions with others were mainly hidden from the users. When devices came into wireless range of one another they exchanged history data and modules, but this was not explicitly shown to the users. In testing it was revealed that this information could be of use to users and designing the next prototype involved making a decision about how to present the information in an useful way, and appropriate within the context of the game. A more seamful approach, as demonstrated in Chapter 3 would be utilised with regard to this particular aspect.

There were some experience and interface issues with the first prototype that were addressed in the final version used for the main trial. Specifically, the issues revolved around feedback and control, that is to say, allowing the user to control with whom history data would be shared, and feedback on how much data had been received. Also, there were issues with the players not having enough information about buildings, for example a building had to be built before the information about what it produced and required was shown. Lastly, players had trouble understanding the rank numbering. The rank number is relative to the number of entries in the database so different ranges of numbers can appear, for example the highest ranking might be seen as 10 or 10,000.

In the trial, the *Domino* technology performed as had been hoped. When battles occurred history logs and modules were exchanged and in following periods of isolated play recommendations were generated and newly available *Domino* modules used. During the trial an average of 2140.8 history entries were exchanged between peers during their encounters, and it took an average of 10.1ms to transfer each individual history entry. This time includes the time for the requests to occur, the database to be queried, entries to be sent over the network, and entries to be inserted in the receiver's database. This suggests that an average total of 99.4 *Domino* history entries can be transferred per second. History transfers occurred in the background during battles, so users did not experience any delay whilst waiting to exchange this data with peer devices. In all cases, history exchanges between devices occurred simultaneously. It is likely that the transfer time would be lower if data were exchanged only in one direction, but the scenario of data being simultaneously exchanged is more typical of a pure peer-to-peer environment, in which all peers are equal and behave similarly. An average of 1.33 modules were transferred during each encounter in the game, and the average size of the modules transferred was 7871 bytes. At the end of the game trials, after playing three other participants, the average size of the history database on each device was 1452KB.

This initial trial of *Domino* in the Castles game was primarily a technical one, ensuring that the technology behaved as expected, and practically demonstrating that high levels of adaptation, based on users' context and a recommendation system, are possible in mobile systems. Whilst the trial did fulfil this primary goal, and revealed some user interface design issues, it was small and it left open many questions—particularly regarding the user experience with the adaptation a *Domino* system provides. For this reason, a second larger trial was carried out, involving a greater number of participants and an updated version of Castles. The goals of this trial were to examine the user experience, discovering if an adaptive system is acceptable to users, whether they value the recommendations for software modules that Domino provides, and showing how they used and interpreted the system's recommendations and adaptations.

6.4 Final System

The first version of the user interface relied on the native look and feel of the Windows Mobile UI widgets and users it affected the game's intuitiveness. Unfortunately these widgets are not very aesthetically pleasing, and the default pop-up list used for buildings proved to constraining for the information users requested, for example the inputs and outputs. To make the software look more like a fun game, it was decided to redesign the interface using a full screen approach with large iconic buttons that clearly represented the game features. Furthermore, the interface chosen was inspired by a popular Windows Mobile car navigation system called Route 66[41], which has an excellent graphics and clear navigation between various forms of information. The map user interface with larger icons and basic stock level information can be seen in Figure 41.

[41] http://www.route66.com

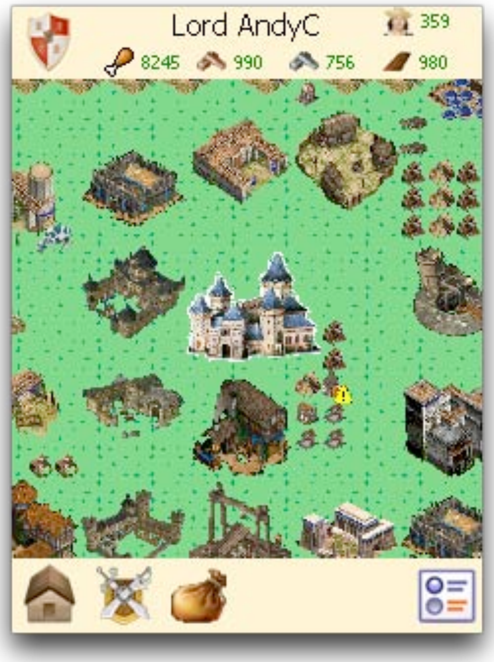
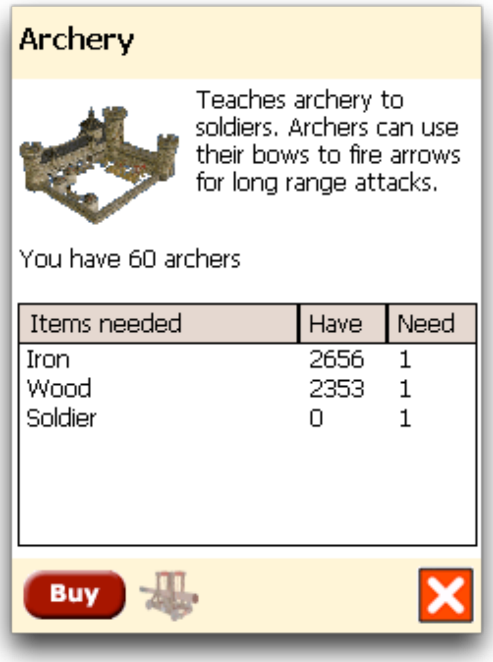


Figure 41: The updated Castles map screen with large intuitive icons and basic stock levels.


Archery

Teaches archery to soldiers. Archers can use their bows to fire arrows for long range attacks.

You have 60 archers

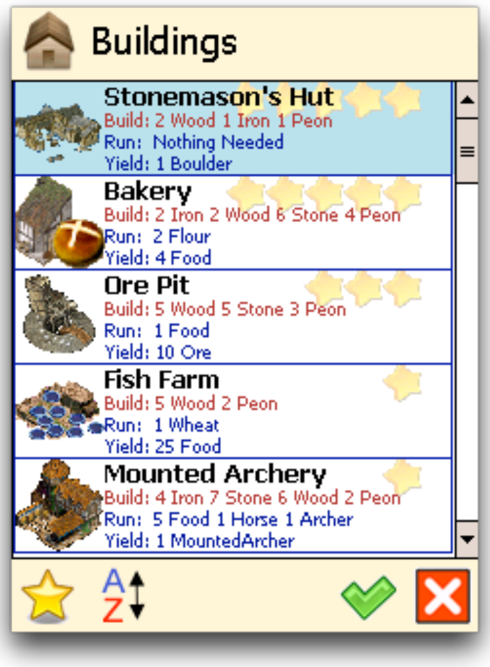
Items needed	Have	Need
Iron	2656	1
Wood	2353	1
Soldier	0	1

Figure 42: This screen is shown when the player clicks on the Archery building. From here, archers can be created or the building can be demolished.

In the pilot, users felt a lack of understanding for why new buildings were suddenly available and it was apparent that more of this process should be made visible to the player. Players felt they would like to see more information about, and have more control over the adaptation process. To address issues of feedback and control, a seamful design was applied utilising the outcomes from Chapter 3, and a similar approach to *Treasure* was used to present underlying technical aspects of the system infrastructure in game terms, for easy understanding that did not detract from the experience of sharing and discovering the software modules. As mentioned in 5.2 Domino was designed and implemented to expose the events within its infrastructure, and this was utilised by Castles, in particular more feedback and control about the recommendation process.

A more seamful design was achieved in two ways. A more informative building screen was created and allowed users to request recommendations at any time. History log exchanges were restricted to only occur during battles and at the end of a battle a message written in ‘game terms’ was shown mentioning that an exchange had taken place.

A building list screen was created to replace the old pop-up list. It displayed each currently installed building and the stock required to build it, and what it requires to produce, and what it yields, along with a star ratings for recommendations (Figure 43). By presenting the detailed building statistics, the aim was that users would feel more confident in making decisions about what to build, and verify they did follow a recommendation they had the stock to make use of it.



Buildings

Stonemason's Hut
Build: 2 Wood 1 Iron 1 Peon
Run: Nothing Needed
Yield: 1 Boulder

Bakery
Build: 2 Iron 2 Wood 6 Stone 4 Peon
Run: 2 Flour
Yield: 4 Food

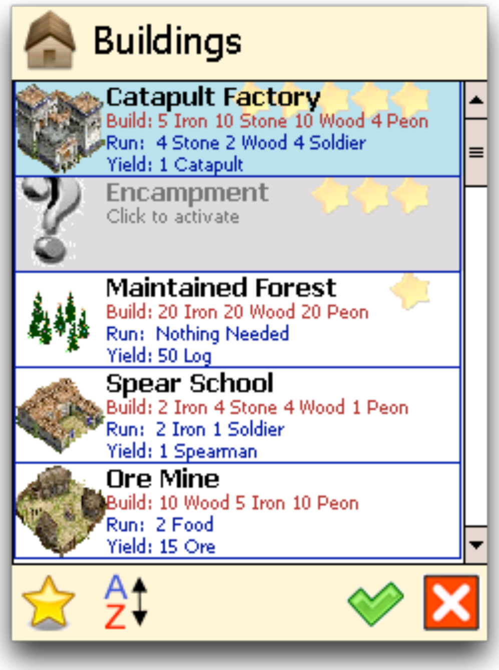
Ore Pit
Build: 5 Wood 5 Stone 3 Peon
Run: 1 Food
Yield: 10 Ore

Fish Farm
Build: 5 Wood 2 Peon
Run: 1 Wheat
Yield: 25 Food

Mounted Archery
Build: 4 Iron 7 Stone 6 Wood 2 Peon
Run: 5 Food 1 Horse 1 Archer
Yield: 1 MountedArcher

Figure 43: The *Castles* building list screen showing star ratings for recommendations of what to build. The star button runs the algorithm and sorts the list in order of rank.

The player can invoke the recommendation algorithm at any time using the star button on the building screen. After a few seconds, a star rating appears on the buildings—which are also ordered by rating. The wide ranges of rank values displayed in the old user interface were scaled to six-point scale: from zero stars to five stars. If a building is recommended that is not one of the installed Domino modules, it appears as a greyed out item although it still has a star rating. If the module has been transferred to the device and is ready to be activated a "click to activate" message is displayed on the list item (Figure 44).


Buildings

Catapult Factory
Build: 5 Iron 10 Stone 10 Wood 4 Peon
Run: 4 Stone 2 Wood 4 Soldier
Yield: 1 Catapult

Encampment
Click to activate

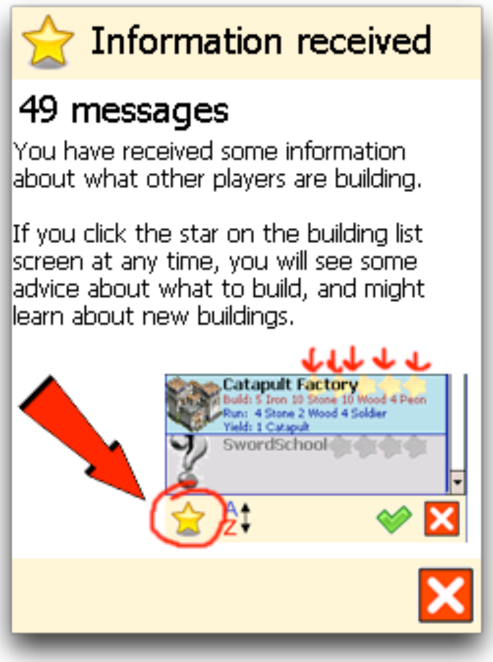
Maintained Forest
Build: 20 Iron 20 Wood 20 Peon
Run: Nothing Needed
Yield: 50 Log

Spear School
Build: 2 Iron 4 Stone 4 Wood 1 Peon
Run: 2 Iron 1 Soldier
Yield: 1 Spearman

Ore Mine
Build: 10 Wood 5 Iron 10 Peon
Run: 2 Food
Yield: 15 Ore

Figure 44: The castles building screen, displaying a greyed out building, which is a recommended Domino module ready to be installed.

After a battle ends, and Domino history information has been received, another new screen is displayed. This screen displays the message "Information received – you have received information about what others are building, use the star to learn about new buildings" and a message count is displayed (Figure 45). This wording was chosen as it notifies the user that information is received during battles, and it directly relates to the star ratings of buildings.


Information received

49 messages

You have received some information about what other players are building.

If you click the star on the building list screen at any time, you will see some advice about what to build, and might learn about new buildings.

Catapult Factory
Build: 5 Iron 10 Stone 10 Wood 4 Peon
Run: 4 Stone 2 Wood 4 Soldier
Yield: 1 Catapult
SwordSchool
A
Z

Figure 45: This *Castles* screen appears after a battle and represents the size of the history path received.

With the new screens, Castle's design was more intuitive. All features were available from the main map screen. The structure of the screens can be seen in Figure 46.

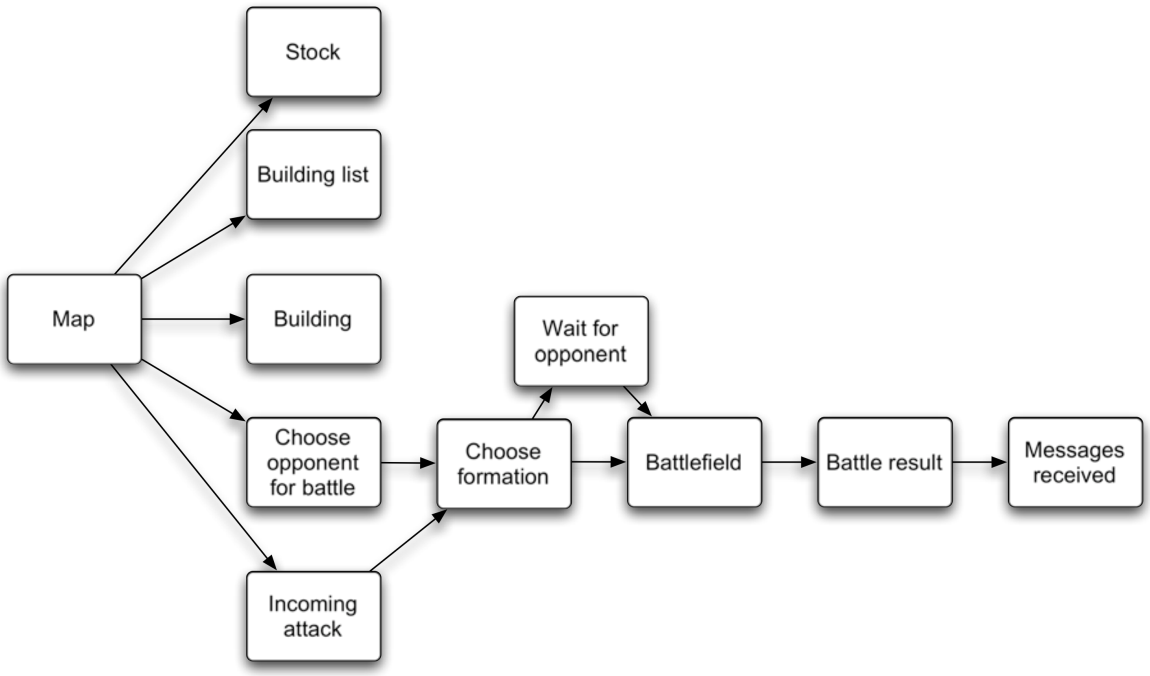


Figure 46: Hierarchy of the screens in *Castles.*

The system logged all game information in an XML format. The log contains both system events, such as when recommendations are requested and modules installed, and state information, for example the stock level that continually changes. Enough information was logged so that games could be replayed in a simulator to examine the adaptations that occurred. Figure 47 shows the XML logged when the recommendation algorithm completes. The domino DLL name is not logged because it can be inferred from the building name.

```
<record type="event">
  <timestamp>632990293040000000</timestamp>
  <Recommendations>
    <HunterHut>1209</HunterHut>
    <StoneMasonHut>580</StoneMasonHut>
    <CatapultFactory>201</CatapultFactory>
    <SpearSchool>145</SpearSchool>
    <Encampment>136</Encampment>
    <MaintainedForest>66</MaintainedForest>
    <Sawmill>33</Sawmill>
    <Pigery>19</Pigery>
  </Recommendations>
 </record>
```

Figure 47: Example from Castle's game log of a recommendation event occurring. The Hunter Hut has a rank of 1209 – the best recommendation for this player.

This version of Castles was a vast improvement of the last. With better game screen structure, more intuitive graphics, and most importantly including feedback and control over the adaptation process presented in a user-friendly fashion — a seamful design.

6.5 Final evaluation

The author primarily conducted the evaluation, however Barry Brown, Louise Barkhuus and Scott Sherwood assisted with the interviews. The trial consisted of 6 game sessions, with 4 participants in each. At the start of the trial, a short tutorial on how the game is played was presented, and participants were then given an opportunity to familiarise themselves with the game and its controls. The trial lasted approximately one hour, with three 10 minute solo building rounds during which participants create buildings and army units, and three battle rounds against each other participant in the trial, each of which lasted until there was a winner. As in the pilot trial, each player started with a different set of 19-21 buildings. The experimental setup can be seen in Figure 48.

Participants were encouraged to openly discuss the game at any time, and they were filmed with two video cameras. After the rounds were complete, they were asked about their experience in an interview in groups of two. The aim of this user trial was primarily to examine people's reactions to history based software recommendations and system adaptation, to see if the changes made due to pilot findings had improved the user understanding of the system.

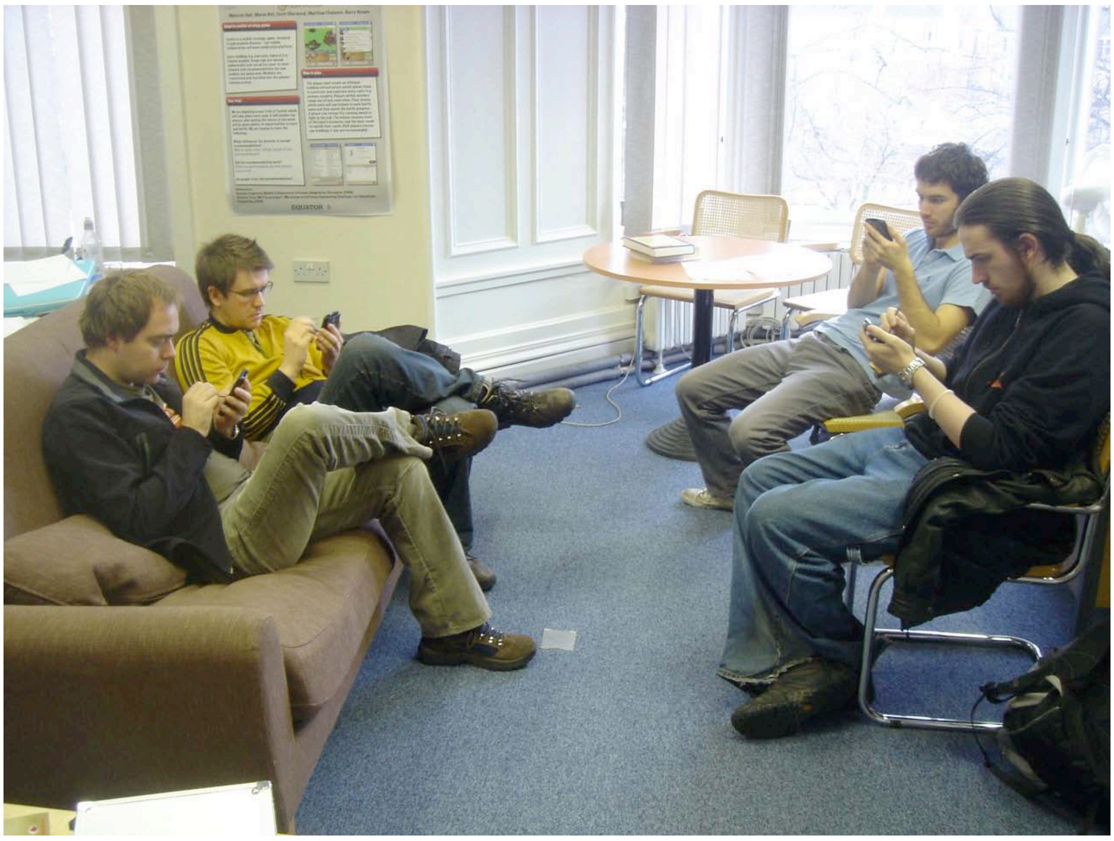


Figure 48: After solo play, the four players battling in *Castles*.

6.6 Results

In very broad terms, the basic mechanisms of Castles—building up resources, meeting other players to battle, and discovering new buildings—were appreciated as being fun and engaging. Players particularly enjoyed the open-ended aspect of the game, and reported that they would have liked to continue to play longer after the trial ended. The players' understanding of the game's features developed to differing levels over time, but all players understood the buildings were being transferred from other players devices.

Generally, from a technical viewpoint, the game worked well and only a few crashes occurred. This was primarily due to bugs in the battle calculation code, not in Domino, and these have since been fixed. As the game state is saved, and Domino saves the list of installed modules, the game could be restarted immediately with no loss of information when crashes did occur, meaning that users could simply restart and continue playing where they left off.

By examining videos of users and logs of system data, we obtained insights into the emergence and success of different strategies, and how features of the system and the setting were used in players' interaction with each other. For example, as their understanding of the game grew and players began to comprehend that their activities were the source of recommendations, they adjusted their use of game strategy accordingly. For example, they began creating other types of units they had seen successfully used by their opponents, especially the cannons, which seemed to generate particular interest because of their dominance on the battle screen. These changes in their game play did not always result in winning battles, but they did generally lead to more excitement and engagement in the game.

Castles mirrored the findings described earlier in *Treasure* in 3.2.1 in that players' development of game play did not stem solely from the space the game was played in, the system design alone, or the space and the system together. Discovering and learning to use new buildings was a combination of the system affording use of them and the conversation with others of how to use them. Players' use was a mix of old and new media developed through a historical and social process. Over time, people affected and were affected by each other, and system and space served as resources, as well as constraints, on interaction.

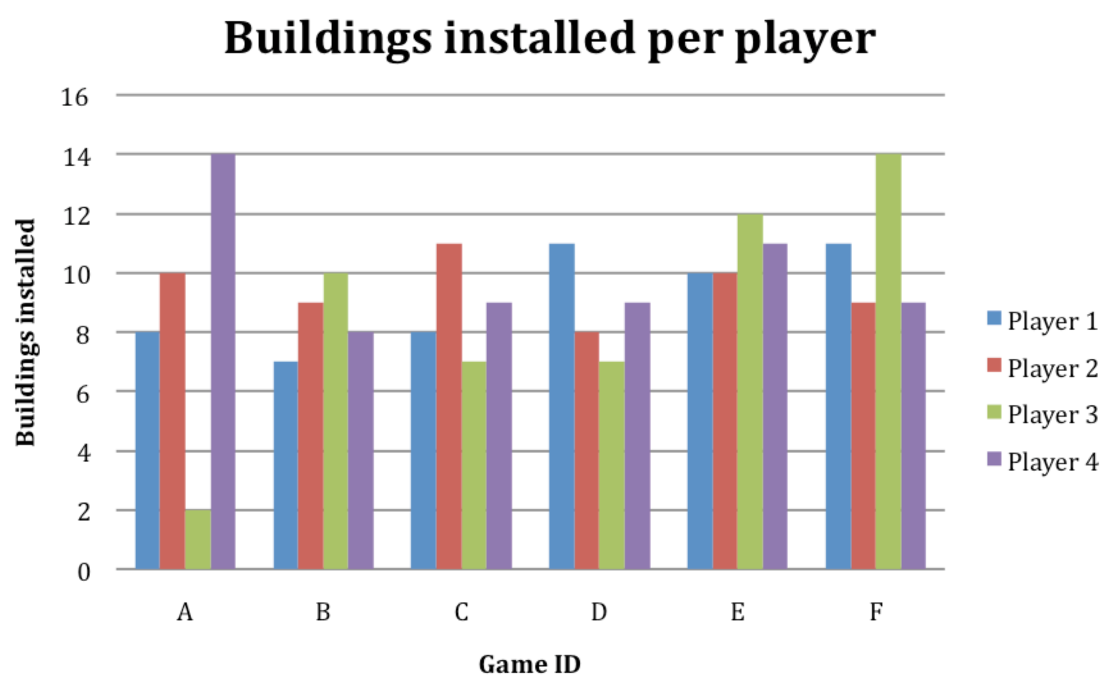


Figure 49: Graph showing the number of buildings installed per player in the trial of Castles.

Figure 49 conveys that all participants successfully adapted their systems by installing many recommended buildings. On average, 9 new buildings were installed per player, bringing the average total buildings after play to 29 (each participant started the trial with an average 20). Strategies around players utilising recommendations and installing new buildings were evident from two findings. People's opinions on how well the recommendation system worked were greatly influenced by early experience.

Secondly, as players' confidence in their own gaming strategy developed, they relied on recommendations less. However, they re-visited the feature in combination with revelations from discussions with other players, especially when stuck in 'resource traps'. Players were 'gaming' the recommendation system to use it to their advantage as much as possible.

As seen in Figure 49, participant A3 only installed two buildings, the lowest of the entire game. This participant explained that the first recommendations received were unsuitable, and then disregarded every future recommendation. This participant had a notably poor performance, winning no games. After the trial games the player reported that if playing again he would definitely reconsider using them, because in discussions the other users expressed the recommendations' positive effects on their game and the broadening of their tactics. It is clear that even though the system may perform well overall, poor impressions are possible and can sour people on using or relying recommendations. If there was more transparency in the recommendation system, then when bad recommendations happen users could understand the reasons behind them. This supports the argument that recommendation systems should find a compromise between giving enough information to allow people to make more informed choices and bombarding them with too much information.

However, the majority of other players had positive early experiences, infusing confidence in the system overall. For example:

> *The most significant thing is that initially when I was learning the game it (recommendations) helped me a lot at the beginning cause I didn't know what I was doing and I had this semi-useful strategy of building the raw materials but after that I didn't know what I was doing and I just clicked the recommendations and it helped me get along.*

Another player specifically mentions that the recommendations allowed him to positively overcome the mental pressure in evaluating the dependencies between buildings:

> *The building part was a bit more involving because you had to think about the dependencies and what you need to build so that you don't get shortages and stuff, however if I may go on to the recommendation system that's where it came in handy at the beginning because I didn't know exactly what I was doing so I would just click on it, ok 5 star and I would just build it with the confidence that it was somewhat the right thing to do. After the first 15 minutes or so when I realised what the game was about and how each thing related to one another I could just, I didn't use the recommendation system at all but before I had gotten to that I guess you could call it level of expertise or whatever - familiarity with the game.*

Managing a balance of resources is a key part of the game. For example, an efficient Castle configuration is one where the outputs of buildings are fully utilised as inputs to other ones. The player quoted felt that the recommendations were only useful in the beginning. However, there were situations in which they became useful again: recommendations were also found to be useful when players got ‘stuck’—that is, their production had come to a standstill, and they did not have enough stock to build new buildings. Although the recommendations were not guaranteed to fix their situation, sometimes trying a recommendation for a building they had not used previously gave players fresh inspiration, perhaps encouraging them to continue in a slightly different direction involving other types of stock.

It became apparent from interviews that all players were able to distinguish between the star rating of buildings for the recommendation for what to build, and the newly discovered buildings that appear with the ‘click to activate’ message. In fact, both of these features are the result of the recommendation system. New buildings were installed continually, but the use of the star ratings were most used early on, and at moments of getting stuck, as mentioned above.

Although users understood that modules came from other players, they did not really understand that histories were replicated. So if A played B, and then B subsequently played C, C received the histories of both A and B; so recommendations for new modules could have arisen from either A’s log, B’s log or both. This failure to understand is most likely due to a deficiency in the messages presented on–screen, and players agreed more information would have been useful. Castles could improve comprehension of players by presenting even more underlying infrastructure as features in the game, by allowing users to ‘drill down’ to gain insights into deeper aspects of the system. For example, by displaying what user’s log the recommended module appeared in, or an information screen displaying statistics about a player’s performance, for example battles won.

Another interesting example from play is that during a battle B1 noticed B3’s use of cannons in the back row of his formation. After the battle, in which B1 lost, B1 was recommended the cannon factory. Then B1 decided to build cannons and use them in the same formation as used by B3. So B1 took the recommendation for the cannon factory and also the extra information of how to use the cannon units successfully in a battle. This is a significant activity, overcoming a weakness of the recommendation system: it only recommends that something should be used, not how to use it, and if the use is not experienced, for example if player B1 had not noticed the cannons, the benefit might have been missed.

To support this feature, Castles could be improved to show screenshots, video clips, or demos of use to show more of the context of how others had used buildings. Dialogues of “do you want to install it?” as well as “do you want to see how to use it?” could be shown, a bit like the “see what’s new” dialogue in some software applications. However, in the case of a Domino application, this information is not something added by the developer, because one cannot predict the way every feature will be used. This should be something either created from log data automatically, or added by users of the component who might make a tutorial or demo for new users or friends they’d like to help—for reasons that might vary from altruism, wishing to be part of the game community, to more selfish demonstration of expertise—i.e. showing off. This example serves to show just one way in which system designers could do more to support the development of the tactics and strategies of users of an adaptive system. Such 'playbacks', and demos based on recorded data on system use, would be another use of historical data within the user experience—extending the use of history beyond that of prior systems, including George Square and Treasure.

Richer display of the use or potential of a building would also address another issue found in the trials. A central part of the users' experience of such dynamic systems is based on their experience of the recommendations—the quality of which depends on the recommendation algorithm, but also on how users play and whom they battle first. Some players dismissed the use of recommendations after an early bad experience, and then regretted it subsequently. Some, perhaps naively, blithely assumed all recommendations were of high quality after initial good experience with them. In actuality, to use a building well, one needs to know what the best balance is with regard to others; for example, what buildings feed into it, and what it can supply. Players have to balance the costs and benefits of taking time to try out recommendations and learn about new buildings and strategies, and so it is likely that it would be beneficial to have more ways to explore past play when beginning a new game; that is, from games prior to the current one—rather as new visitors to George Square were able to draw upon the experiences of prior visitors that they might never have met.

Unsurprisingly, all the participants reported that they felt more confident in selecting which buildings to create after they had their first encounter with a peer, and hence were able to generate recommendations. However, none of the players realised that the modules transferred could contain potentially harmful code. Since participants were likely to have felt secure in the controlled trial environment, and would likely have assumed that the trial organisers wrote all the code, they did not perceive the risk from harmful modules. This raised both security and privacy concerns on our part, as participants were clearly unaware that logs were being transmitted or that the new building modules that were received could have contained any code—including code that could be harmful, or access logs for malicious reasons. These concerns led to a slight redesign of the peer-to-peer Recer recommendation system to allow for greater anonymity and to considerations of possible solutions to increase security. Some of the anonymity improvements have since been implemented, whilst possible security solutions are still being considered.

As with the George Square and Treasure trials, the trials of Castles revealed that the system's use was dependent on the people using it as much as the objective functionality the system offered. Module recommendation and associated system adaptation were generally found to be useful, and yet they were also subjects of individual interpretation and social discussion. In other words, recommendations were not followed blindly and individually, but tactically (e.g. to work out what other people were doing) and socially, as a means to learn about the system's possibilities, as a mechanism to be understood and manipulated to one's own advantage (i.e. to be 'gamed'), and so forth.

These findings tend to confirm the view of system adaptation as a sociotechnical process rather than a purely objective technical one. This backs up the design decision not to make adaptation happen fully automatically. The trials suggest that the system should not only support individual user awareness of current choices, but should show—in comprehensible terms appropriate to the application and the users—what has been done by users in the past and how they might behave in the future.

Lastly, the game was successful as a means to display that variations in another ubicomp infrastructure—software infrastructure, in this case—were presented and used in ways that were crucial to user activity and enjoyment. Castles, like Feeding Yoshi and Treasure, adds to the evidence that ubicomp technical infrastructure can be considered by designers as a *seamful* resource for users' interaction—interaction with the system and with each other.

A link to the trial system logs and interview transcripts can be found in Appendix A – Online materials.

6.7 Security considerations

Security is a serious problem for any system that uses mobile code that is exchanged between different devices, and it has been an important focus of this authors and others' research, for example, [96] [97] [98] [99]. A particular threat is so called 'sleeper viruses' that act as valid and useful modules for a period of time, become accepted in a community, and then after an incubation period 'turn bad' and start to act as damaging viruses.

Signing is a popular technique for deciding which applications to trust, in which a trusted authority analyses each possible application or module and decides whether it is harmful or not. Those that are determined to be non-harmful are signed with a secure key that end-clients know they can trust. In theory this can inhibit harmful applications from spreading to many machines, however most implementations permit a user to decide to force an unsigned module or application to run, allowing dangerous code to spread regardless of its lack of authorisation.

Whilst employing signing for Domino would provide an almost complete solution to security concerns, there are severe disadvantages that have, so far, stopped us from implementing it. Firstly, one of Domino's main strengths is that it allows for an extremely open community where anyone can contribute a new module or amend an existing one. In an environment where each module had to be signed a large number of users would decline to create new modules, as those modules would then have to go through the signing process. As this would be likely to involve some cost (in terms of money or time for developers) this would further deter potential developers from contributing to the community. Furthermore, forcing each module to go through a central location where it was signed would negate the strength of the epidemic spreading Domino supports. There would be little or no reason to provide epidemic spreading if one source had access to every possible module in the community and could therefore, in theory, simply distribute them all from one central location.

A second possible solution is to create a sandbox environment for both the entire Domino environment running on a device and for each individual module within that environment. Indeed, as Domino is implemented using the .NET Framework that it already runs through a virtual machine — specifically the CLR (Common Language Runtime). Code Access Security is a feature of .NET to restrict any application from having access to a part of or the entirety of the rest of the operating system depending on set permissions. However, if a Domino module wanted to access a file on the local device and didn't have the necessary permission, may have to ask permission from the user who could deny, accept once or accept forever the module's request. Whilst many languages that run on virtual machines employ this method, it would be likely to be too intrusive to users in a Domino environment. Previously, this method has usually been used where the number of new modules or applications is relatively low, and so the user is required to intervene on an infrequent basis. In a typical Domino system there can be an extremely large number of modules running at any one time, and requiring the user to intervene for each one could prove too time-consuming. Furthermore, as one of the advantages of Domino is that it allows users to quickly obtain expert tools, it is unlikely that the user would have the required in-depth knowledge of each particular module to make the correct decisions about when to trust them. Methods of automating the process of determining which applications should be permitted to run or have access to a particular part of the operating system may aid the user in this process. For example, *Deeds* [100] attempts to analyse code and roughly categorise it before comparing it to the access levels given to code that previously fell into the same category. Such a technique could make permissions a viable option in the Domino architecture, by removing many of the constant interruptions that might otherwise be presented to the user.

A third potential solution relies on the same epidemic algorithms as the spread of the modules themselves, spreading information about malicious modules during any contact with peers. For example, if one user found a malicious module they could, after removing it, add it to a list of known bad modules. From then on, the list would be transmitted to any Domino peers that were encountered. A Domino client which had received this information could then refuse to accept the module if it ever encountered that module. Similarly, a client that was running the module and received information that it was malicious could quickly remove the module even if it had not yet done any damage. As the information about malicious modules would be constantly spread rather than having to be recommended, and as clients would be able to remove the module before it did any damage, the spread of the information that the module was malicious would probably be faster than the spread of the module itself. In this way, viral outbreaks of malicious modules might generally be prevented. However, this solution has a weakness in that although it would stop a large viral outbreak in the community, it would not stop damage to a particular client who received the module before receiving the information that it was malicious. More advanced implementations could make use of the Internet to broadcast information about malicious modules, 'overtaking' their spread through peer-to-peer contact. In so–called 'honey pot' implementations [101], this has been shown to be particularly effective at stopping the spread of conventional computer viruses.

Apart from these technical approaches to countering viruses, it is possible for a user to view a module's history of use: on which device it originated, on which other devices it was used prior to its arrival, and in what contexts it was used along the way with regard to other modules. This helps users to decide for themselves whether the history is typical of a trustable module. Alternatively this history information could be fed into an algorithm such as that in [102] or [103], to give a calculated level of trust. Although this technique may not be sufficient in itself, its use is advocated as an additional protection method to be used in conjunction with other measures.

As stated, security is a serious issue and, whilst this is being researched along with possible solutions, a single robust solution has not been found yet which is completely trustworthy. For this reason, the creation of 'mission critical' applications based on the Domino architecture have, so far, been avoided and have instead, for the time being, concentrated implementing Domino into game systems. While this does not avoid problems of viruses and malware (since 'bad' modules could destroy a user's game, or be used as a way of cheating) it does provide an environment for experimenting with module recommendation and the broader security issues, limiting the potential damage to users' devices. It is one of the topics of future work planned for Domino, as mentioned in the next subsection.

6.8 Future work

Although a substantial volume of work was completed during the course of this thesis, there remain problems and limitations that the author hopes to overcome in the future. Domino is an advanced infrastructure, designed to be easy to integrate into mobile systems in order to increase their adaptability and flexibility. Work continues on improving Domino, and one aspect is to show the benefits of removing a running module from a system, rather than only adding new ones. Users can manually remove modules, to reduce the system becoming bloated or confusing, but at the moment Domino does not assist users in this process. Analysing logs of user activity can help with these issues, if the details of modules' use and removal are recorded. Normal, continuing use could involve periodically recording a small positive weight for each module in the current configuration. However, if users consistently install one module and then manually remove another soon after, this may indicate that the former is an upgraded version of the latter or otherwise replaces the latter's functionality. This recorded pattern of use might then be interpreted by the system so as to record a substantial negative weight for the removed module in the history database, to help lower it in the rankings of modules while the new module builds up its use. If a user does not have the apparently older or superseded module, then he or she will be less likely to receive recommendations for it. If a user does have the module, the system may be able to recommend the new module as well as the removal of the old one.

Unfortunately, as its design and implementation of Domino was complex and consumed a substantial amount of time, there has not been an opportunity to make Domino generally available to other developers yet. Although the author and the Equator team at Glasgow were able to use Domino in a few of their research systems, completion of a version of Domino that was well documented and robust was not achieved early enough to allow for its dissemination to the wider mobile developer community. Therefore, an important item to address in the near future is the public release of the Domino infrastructure in order to allow a greater number of new mobile applications that make use of its unique abilities to be constructed. To achieve this a Domino SDK, developer documentation and example code will be released through CodePlex.com in the coming months. Due to demand, a public release of Domino and Castles, without documentation has been made available at the Equator project web site[42] and an updated version with documentation will be available at the link in Appendix A – Online materials.

The Castles trials proved extremely useful, and successfully demonstrated that the Domino infrastructure proved to be of great value to end users when integrated into the Castles game. However, it is realised that the trial was relatively constrained, both in terms of its setting (primarily it was conducted in one building), and length. It is felt that longer trials over a period of weeks, and covering a greater geographical area, would allow for a more in-depth study of how users react to software adaptation and recommendations in a more natural setting as they go about their normal daily tasks. Indeed, if participants carried a PDA continually over a period of weeks, the area covered would likely be large, and the chances and situations for encountering peers would be more varied. Therefore, a larger and longer trial of Domino in a new application aimed at supporting module developers is planned to take place in 2008.

A different area of work relates to the way that the concepts and techniques behind Domino have application to less mobile settings—in particular, applications in software development and plug-ins for IDEs (integrated development environments) and in web browsers such as Firefox. As pointed out in [104], many such systems are large and yet rather chaotic, and a Domino-like system might assist users in finding new code to work on as well as new code to work with.

In IDEs, mail tools and in mobile systems, Findlater and McGrenere' comments about involving the user should be borne in mind. There may well be applications that would demand or involve automatic changes to an interactive system without a user's permission, but it is difficult to come up with too many examples of them. Instead, the techniques explored in Domino are a means to combine adaptable and adaptive elements, so that the system and the user both control some of the interaction. Domino goes beyond most existing systems in providing evidence that collective records and patterns of use can be a productive resource for individuals adapting their adaptive systems.

As shown in background work on context aware systems, location information is a very useful resource. Expansion of Domino's model of context and its recommendations is at the core of a forthcoming EPSRC-funded project due to commence in Spring 2008, *Contextual Software*. Quoting from its work plan, "At present, Domino's model of context is univariate: the modelled dependencies and associations are only between software components. Building on *George Square*, which used patterns of association in

[42] http://www.equator.ac.uk/index.php/articles/c117

usage histories to combine locations and URLs, Domino's information on context, operation and use will be enriched. New Visual Studio tools will allow the programmer to specify and analyse dependencies and patterns of association between components, locations and other logged contextual features. The programmer will then use the project's new communications infrastructure to disseminate new components among user trial participants, along with any specified usage histories needed to 'bootstrap' the recommendation process. For example, a programmer might specify that a new component offering an audio tour of a museum can only be started up when the user's device is in that museum. In a trial, he could then analyse the sets of components running when the new one was installed and used, where in the museum this occurred, and video recordings of user discussion at those times."

Similarly, security is a topic of future work in the new project that will take advantage of the low-level middleware of the EU *Haggle* project[43]: "The distribution and integration of Domino components is, at present, relatively inefficient and insecure. By porting Domino to run on top of the .NET implementation of Haggle, secure opportunistic forwarding via both fixed and mobile ad hoc networks can be used, using multiple forms of connectivity, for example, WiFi, SMS, GPRS and Bluetooth. Domino will keep its facilities for recommendation and adaptation, but will use Haggle for device discovery, transmission of usage histories and components, resource management and security. Thus, collection of usage data from mobile devices, and distribution of new components and usage data from the IDE to those devices—including new Domino components that offer users choice over Haggle's facilities for communications, resource management and security will be supported. A PhD student (RS1) will focus on usable security and privacy built on Haggle's facilities and techniques such as audit trails, virtual machine sandboxing and OS virtualisation."

6.9 Conclusion of investigation

The *Castles* game demonstrates one application of *Domino* in a mobile, peer-to-peer environment. As mentioned at the start of this chapter, Castles is a context aware, peer-to-peer, seamful, adaptive system. The treatment of context used is very limited, being explicitly restricted to the set of Domino modules in use but implicitly also employing the proximity of other users so as to narrow or bias the sharing and discovery of modules. Castles used the infrastructure developed in the systems discussed in earlier chapters, so as to gain the advantages of peer-to-peer mobile communication such as mobility, cheapness and high bandwidth. Castles is a seamful design in that it selectively exposes software structure to users, so that they can be aware of software modules and appropriate them for their own contextually relevant patterns of use. Lastly, the trials offered a technical demonstration that Castles is adaptive around a user's activity, and that functionality can be discovered, shared and integrated in the mobile environment whilst applications are actively running. This therefore demonstrated a degree and style of system adaptation not previously shown in ubicomp, but it also demonstrated the value of future work on topics such as removal as well as addition of modules, greater transparency (or seamfulness) in the interface, addressing of security and privacy concerns, and extension of the set of contextual features used in recommendations.

[43] http://www.haggleproject.org

Rather like the findings of the George Square trials, the trials of Castles revealed that the system’s use was dependent on the people using it as much as the objective functionality the system offered. Module recommendation and associated system adaptation were generally found to be useful, and yet were also subjects of individual interpretation and social discussion. In other words, recommendations were not followed blindly and individually, but tactically (e.g. to work out what other people were doing), as a means to learn about the system’s possibilities, as a mechanism to be understood and manipulated to one’s own advantage (i.e. to be ‘gamed’), and so forth.

These findings tend to confirm the view of system adaptation as a sociotechnical process rather than a purely objective technical one. This backs up the design decision not to make adaptation happen fully automatically. The trials suggest that the system should not only support individual user awareness of current choices, but should show—in comprehensible terms appropriate to the application and the users—what has been done by users in the past and might they might do in the future.

Chapter 7 Conclusions

In this chapter a summary of the results and contributions of the thesis are presented.

7.1 Summary of thesis

The introductory chapter expressed that there is a growing need to employ adaptation in mobile ubiquitous systems to alleviate problems in design caused by difficulties in predicting context changes in the mobile environment, users' needs over long periods of time and the various unique situations mobile applications are expected to be available in. It was observed that, to date, the majority of mobile applications are simply reduced versions of standard desktop applications. They are rigid and inflexible, requiring many compromises from users to provide any useful functionality; they are essentially blind to the vast volume of context information potentially available to a mobile device.

Chapter 2 covered research on context awareness relevant to the work in this thesis. The features that make up what a program might model as 'context' were discussed. In particular, it is clear that context awareness is of critical importance to the type of dynamic mobile systems proposed throughout this thesis, and two aspects of context might were highlighted as being potentially useful with regard to adaptation: the configuration of the system itself, and the history of its use.

Chapter 3 presented the concept of seamful design, and returned to the roots of ubicomp as a principle or ideal. The systems described in the chapter, Treasure and Feeding Yoshi, demonstrate seamfulness' first full application to ubicomp system design. The games were used as an initial vehicle to explore seamful design, and showed that ubicomp system design could make infrastructure into a resource for users in ways that let them use or appropriate the system and its infrastructure to suit their own contexts, for example, building up strategies and tactics with regard to WiFi, and fitting such as system into their everyday lives.

Chapter 4 identified literature involving mobile peer-to-peer systems, and it was discovered that of particular importance are those systems that take advantage of people's everyday movement as the mechanism for transport and filtering information in a purely ad hoc mobile scenario - the social proximity application (SPA). Several systems were presented that utilised social proximity to spread data epidemically within a community of users.

Chapter 5 examined existing literature in the field of adaptation. An architecture for a mobile software recommendation and adaptation was presented, Domino, involving a design based on the findings of the earlier chapters based on literature and systems, in particular mobile ad hoc networks, peer discovery, epidemic sharing of software modules as well usage histories, and a peer-to-peer version of the Recer recommender subsystem.

Chapter 6 discussed the Castles system, which demonstrated the deployment and user experience of a Domino-based system. Module recommendation and associated system adaptation were generally found to be useful, and yet were also subjects of individual interpretation and social discussion. Trial findings confirmed the view of system adaptation as a sociotechnical process rather than a purely objective technical one, added to the evidence for the viability of the seamful approach, and gave pointers towards future work on further ways to reveal history and recommendation information, and on handling privacy and security.

7.2 Contributions

In conclusion, this thesis has addressed the three original research questions introduced at the outset of the thesis:

RQ1 *What aspects of existing research and systems can be applied to the design of an adaptive infrastructure for ubicomp?*

RQ2 *How can ubicomp researchers design more dynamic and adaptive systems?*

RQ3 *How do users react to adaptive and dynamic ubicomp systems?*

RQ1 was addressed through a literature review, detailed analysis of systems and user studies in the areas of context awareness, seamful design, mobile peer-to-peer applications and adaptive systems (Chapters

2–5). RQ2 was addressed through the design and implementation of Domino, an adaptive infrastructure for ubicomp presented in the second half of Chapter 5, designed specifically based on the research and systems investigated to address RQ1 in the earlier chapters. RQ3 was addressed through the implementation and user trial of a Domino based adaptive dynamic ubicomp system, Castles, where users found recommendations and associated system adaptation generally useful, and yet were also subjects of individual interpretation and social discussion.

Chapter 2 addressed RQ1 by conveying that making a system have a broad model of context—spanning the external environment around it, such as peers and other devices, internal context such as files or software configuration on the device itself, and historical aspects of context such as usage histories—may greatly assist towards functionality, usability and behaviour appropriate to an adaptive ubicomp system. The George Square system proved an excellent test bed to examine how an advanced context aware system may operate, and offered findings highly relevant to the design of adaptive systems. George Square demonstrated how the past could be used as a resource to generate relevant information from recommendations, based on a comparison of the current user's context with the histories of past activity and context generated by previous users, and indicated that the same mechanism could potentially drive software adaptation. George Square also offered design guidelines that should be adhered to when creating mobile, context-aware adaptive systems. The architecture of the George Square software infrastructure, although modular and reconfigurable, was limited by its reliance on central servers, suggesting that dynamic architectures would be more flexible if relieved of this restriction. George Square featured content and associations that adapt dynamically with use, rather than being completely pre-authored. George Square highlights the importance of the peer-to-peer community in mobile systems and suggests that only by sharing information between a community of users can many mobile systems gain a critical mass of data that makes them useful and interesting.

Chapter 3 addressed RQ1, demonstrating that the application of seamful design to a complex and dynamic adaptive system could be beneficial to user understanding and acceptance. If adaptive systems were to expose aspects of their underlying infrastructure, users could take advantage of past patterns of use; for example, finding practical significance in current events though their relationship to past events. Treasure and Feeding Yoshi showed that ubicomp system design could make infrastructure into a resource for users in ways that let them use or appropriate the system to suit their own contexts. Histories of use were again found to be a useful resource in this task, for example, in the WiFi maps of Treasure. To some extent these maps are examples of content being created by users, reflecting the benefits of user-generated content shown in George Square.

Chapter 4 addressed RQ1 by examining literature involving mobile peer-to-peer systems, and of particular importance are those systems that take advantage of people's mobility as the transport mechanism for information in a purely ad hoc mobile scenario—the social proximity application (SPA). This thesis contributed a novel type of SPA, one that shares data to generate intelligent recommendations of software. This type of adaptive SPA can receive new functionality in the form of software components between collocated users, utilising social activity and movement to drive the mechanism offering changes in functionality. As such, the systems described in his chapter also serve as concrete demonstrations of responses to RQ2. *FarCry* demonstrated the epidemic spreading of information via peer-to-peer connections over wireless networks, aided by a custom wireless driver. StreetHawk pushed the limits of using dynamically discovered WiFi networks by supporting both infrastructure and ad hoc modes. The peer-to-peer version of Recer, built on top of the FarCry platform and tested in Samara, allowed history logs to be distributed and maintained efficiently and a recommendation service to be built that avoided George Square's limitations due of centralisation. Overall, the experience gained from implementing and testing these systems helped towards answering both RQ1 and RQ2, and meant that we had reliable systems for peer discovery, communication and information sharing—identified as being key requirements of a successful mobile adaptive infrastructure.

Chapter 5 addressed RQ1 by examining existing literature on dynamic software adaptation and its relevance to ubicomp systems. Three different types of adaptation were examined in this chapter: adaptive content—the content of a web page, for example, is customised depending on the context of the user, adaptive displays and user interfaces—the actual interface itself is adapted as opposed to the content, and personal adaptation of software—users extending the functionality of the software they use in a more general way.

The ideas and experiences explored in seeking to answer RQ1 offered a potential novel approach to overcome the weaknesses of current mobile systems, in that they are mainly static, cut-down versions of desktop software, and offered an opportunity to build successful mobile adaptive systems, which blend with and support users in their dynamically changing activities and environments, thus fitting with the design ideals of ubicomp systems and addressing RQ2. The Domino system was designed to specifically address the requirement that mobile systems should be highly adaptable to the user and his/her context. By relying on the peer-to-peer Recer system, Domino continually monitors a user's context, records his/her actions and actively adapts, at runtime, to what the user is involved in. By distributing both the history logs and functionality in the form of modules, the Domino system is able to provide an extremely high degree of adaptation within a mobile, peer-to-peer environment. The level of adaptation Domino provides is undoubtedly novel to the mobile environment, and yet also goes beyond the dynamic adaptation that has been achieved in most desktop systems.

To address RQ3, the *Castles* game was built to demonstrate one application of *Domino* in a mobile, peer-to-peer environment and was the subject of a user trial to uncover users' reactions. Castles is a context aware, peer-to-peer, seamful, adaptive system. Castles used the infrastructure developed in the systems discussed in earlier chapters, so as to gain the advantages of peer-to-peer mobile communication such as mobility, cheapness and high bandwidth. Castles is seamful in that it selectively exposes software structure to users, so that they can be aware of software modules and appropriate them for their own contextually relevant patterns of use. The user trial offered a technical demonstration that Castles is adaptive and adaptable around a user's activity, and that functionality can be discovered, shared and integrated in the mobile environment whilst applications are actively running. Rather like the findings of the George Square trials, the trials of Castles revealed that the system's use was dependent on the people using it as much as the objective functionality the system offered. Recommendations were not followed blindly and individually, but tactically and as indirect expressions of players' activity and skill. The Castles trial findings and surrounding discussion therefore document users' reactions to an adaptive and dynamic ubicomp system, and therefore are an answer to RQ3—thus completing this thesis' response to the three original research questions set out at the beginning of this thesis.

7.3 Conclusion

This thesis set out to address a longstanding issue of how to design software that can continue to be helpful and appropriate whilst operating in the highly dynamic environment of people's everyday lives. Researchers agree that system adaptation is a key issue in the field of ubicomp because it can be hard to design for all situations how and where software should be used and how devices should behave under different circumstances. The fact that an overwhelming number of researchers paraphrase the same issue confirms that this is indeed a problem in the field.

This thesis presented the Domino architecture, and its approach to dynamic adaptation to support users' needs, interests and activities. Domino identifies relationships between code modules beyond those specified in code by programmers themselves prior to system deployment, such as classes, interfaces and dependencies. It uses those relationships, but it also takes advantage of code modules' patterns of use and combination after they have been released into a user community. The Castles game demonstrated Domino's components and mechanisms, exemplifying its means of peer-to-peer communication, recommendation based on patterns of module use, and adaptation based on both module dependencies and history data. The openness and dynamism of Domino's system architecture is applicable to a variety of systems, but is especially appropriate for mobile systems because of their variety and the unpredictability of patterns of use, their frequent disconnection from fixed networks, and their relatively limited amount of memory. As people visit new places, obtain new information and interact with peers, they are likely to be interested in new software that can optimise their tasks, and novel methods of interacting with and combining modules. These requirements justify the need for dynamic adaptive systems in ubicomp. This thesis has successfully presented a viable infrastructure for the development of such systems, and therefore delivered a basis for solving a long-standing problem in ubicomp.

Appendix A – Online materials

Online materials that accompany this thesis are briefly listed below. The up-to-date index is available at http://www.dcs.gla.ac.uk/˜hall/thesis/. These materials include demos (suitable for Windows Mobiles with touch screens), videos, and pictures of the various implementation demonstrations in action. The electronic version of this thesis is also available.

Domino **Section 5.3**
Source code and documentation.

Domino map sharing application **Section 5.3.5.1**
Video of a map layer being transferred between PDAs.
Source code.

Castles **Chapter 6**
Source code and documentation.
Trial system logs, videos and interview transcripts.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American,* vol. 265, pp. 94-104, 1991.

[2] L. Barkhuus and P. Dourish, "Everyday Encounters with Context-Aware Computing in a Campus Environment," in *Ubicomp*, Nottingham, UK, 2004, pp. 232-249.

[3] J. Humble, A. Crabtree, T. Hemmings, K.-P. Åkesson, B. Koleva, T. Rodden, and P. Hansson, "“Playing with the Bits” User-configuration of Ubiquitous Domestic Environments," in *UbiComp*, Seattle, Washington, USA, 2003.

[4] T. Rodden and S. Benford, "The evolution of buildings and implications for the design of ubiquitous domestic environments," in *CHI 2003*, pp. 9-16.

[5] W. K. Edwards and R. Grinter, "At Home with Ubiquitous Computing: Seven Chal-lenges," in *Ubicomp*, 2001, pp. 256-272.

[6] O. S. Kaya, O. D. Incel, S. Dulman, R. Gemesi, P. Jansen, and P. Havinga, "Using TinyOS Components for the Design of an Adaptive Ubiquitous System," in *International Workshop on Wireless Ad-hoc Networks*, 2005.

[7] A. K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," in *Human Computer Interaction*, 2001, pp. 97-166.

[8] A. K. Dey and G. Abowd, "Towards a Better Understanding of Context and Context-Awareness," in *1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, Karlsruhe, Germany, 1999, pp. 304-307.

[9] B. N. Schilit, N. I. Adams, and R. Want, "Context-Aware Computing Applications," in *Workshop on Mobile Computing Systems and Applications* Santa Cruz, CA, USA: IEEE Computer Society, 1994, pp. 85-90.

[10] R. Hull, P. Neaves, and J. Bedford-Roberts, "Towards Situated Computing," in *The First International Symposium on Wearable Computers*, Cambridge, MA, USA, 1997, pp. 146-153.

[11] J. Redstrom, P. Dahlberg, P. Ljungstrand, and L. E. Holmquist, "Designing for Local Interaction," in *Managing Interactions in Smart Environments*, 1999.

[12] E. Paulos and E. Goodman, "The familiar stranger: anxiety, comfort, and play in public places," in *Proceedings of the SIGCHI conference on Human factors in computing systems* Vienna, Austria: ACM, 2004, pp. 223-230.

[13] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in *2nd IEEE International Symposium on Wearable Computers*: IEEE Computer Society, 1998, pp. 92-99.

[14] M. Chalmers, "A Historical View of Context," *Comput. Supported Coop. Work,* vol. 13, pp. 223-247, 2004.

[15] W. Newman and P. Wellner, "A desk supporting computer-based interaction with paper documents," in *Conference on Human Factors and Computing Systems*, Monterey, California, USA, 1992, pp. 587-592.

[16] B. Brown, I. MacColl, M. Chalmers, A. Galani, C. Randell, and A. Steed, "Lessons From The Lighthouse: Collaboration In A Shared Mixed Reality System," in *Human Factors in Computing Systems*, Fort Lauderdale, Florida, USA, 2003, pp. 577-584.

[17] M. Chalmers, K. Rodden, and D. Brodbeck, "The Order of Things: Activity-Centred Information Access," in *World Wide Web*, Brisbane, Australia, 1998, pp. 359-367.

[18] M. Chalmers, "Information Awareness and Representation," *CSCW,* vol. 11, pp. 389-409, 2003.

[19] C. Greenhalgh, "EQUIP: a Software Platform for Distributed Interactive Systems," in *Equator Technical Report 02-002* Nottingham: University of Nottingham, 2002.

[20] G. Fitzpatrick, S. Kaplan, T. Mansfield, A. David, and B. Segall, "Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections," *CSCW,* vol. 11, pp. 447-474.

[21] M. Weiser, "The World is not a Desktop," *ACM Interactions,* vol. 1, pp. 7-8, 1994.

[22] M. Bell, M. Chalmers, L. Barkhuus, M. Hall, S. Sherwood, P. Tennent, B. Brown, D. Rowland, and S. Benford, "Interweaving mobile games with everyday life," in *Proceedings of the SIGCHI conference on Human Factors in computing systems* Montreal, Canada: ACM, 2006.

[23] M. Chalmers, I. MacColl, and M. Bell, "Seamful Design: Showing the Seams in Wearable Computing," in *IEEE Eurowearable*, Birmingham, United Kingdom, 2003, pp. 11-17.

[24] M. Flintham, R. Anastasi, S. Benford, T. Hemmings, A. Crabtree, C. Greenhalgh, T. Rodden, N. Tandavanitj, M. Adams, and J. Row-Farr, "Where On-Line Meets On-The-Streets: Experiences With Mobile Mixed Reality Games," in *Human factors in computing systems*, Fort Lauderdale, Florida, USA, 2003, pp. 569-576.

[25] W. W. Gaver, J. Beaver, and S. Benford, "Ambiguity as a Resource for Design," *CHI Letters,* vol. 5, pp. 233-240, 2003.

[26] K. Mitchell, D. McCaffery, G. Metaxas, and J. Finney, "Six in the City: Introducing Real Tournament - A Mobile IPv6 Based Context-Aware Multiplayer Game," in *NetGames '03*, Redwood City, California, USA, 2003, pp. 91-100.

[27] S. Benford, J. Bowers, P. Chandler, L. Ciolfi, M. Flintham, M. Fraser, C. Greenhalgh, T. Hall, S. O. Hellstrom, S. Izadi, T. Rodden, H. Schnadelbach, and I. Taylor, "Unearthing virtual history: using diverse interfaces to reveal hidden virtual worlds," *Lecture Notes in Computing Science,* vol. 2201, pp. 225-231, 2001.

[28] L. Barkhuus, M. Chalmers, P. Tennent, M. Hall, M. Bell, S. Sherwood, and B. Brown, "Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game," in *Ubicomp*, Tokyo, Japan, 2005, pp. 358-374.

[29] A. Luis von and D. Laura, "Labeling images with a computer game," in *Proceedings of the SIGCHI conference on Human factors in computing systems* Vienna, Austria: ACM, 2004.

[30] E. G. Rebecca, W. K. Edwards, W. N. Mark, and D. Nicolas, "The work to make a home network work," in *Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work* Paris, France: Springer-Verlag New York, Inc., 2005, pp. 469-488.

[31] F. Siegemund, "Spontaneous Interaction using Mobile Phones and Short Text Messages," in *Workshop on Supporting Spontaneous Interaction in Ubiquitous Computing Settings at Ubicomp*, 2002.

[32] A. J. Nicholson, I. E. Smith, J. Hughes, and D. Noble, "LoKey: Leveraging the SMS Network in Decentralized, End-to-End Trust Establishment," in *Pervasive*, 2006, pp. 202-219.

[33] A. L. Murphy, G.-C. Roman, and G. Varghese, "An Exercise in Formal Reasoning about Mobile Communications," in *Software Specifications and Design*, Ise-Shima, Japan, 1998, pp. 25-33.

[34] A. Beaufour, M. Leopold, and P. Bonnet, "Smart-tag based data dissemination," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, Atlanta, Georgia, USA, 2002, pp. 68-77.

[35] J. Brucker-Cohen, K. Moriwaki, and L. Doyle, "UMBRELLA.net: Exploring Coincidence Ad-Hoc Networks," in *Ubicomp*, Nottingham, UK, 2004.

[36] M. Östergren, "Sound Pryer: truly mobile joint music listening," in *ICEC*, 2004.

[37] A. Bassoli, J. Moore, and S. Agamanolis, "TunA: Synchronized Music-Sharing on Handheld Devices," in *Ubicomp*, 2004.

[38] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall, "When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks," in *First International Conference on Peer-to-Peer Computing*, Lingköping, Sweden, 2001.

[39] P. Persson, J. Blom, and Y. Jung, "DigiDress: A Field Trial of an Expressive Social Proximity Application," in *UbiComp 2005*, 2005, pp. 195-212.

[40] L. Holmquist, J. Falk, and J. Wigstrm, "Supporting Group Collaboration with Inter-Personal Awareness Devices," in *Personal Technologies*, 1999.

[41] W. Alexandra and H. Lars Erik, "Hummingbirds Go Skiing: Using Wearable Computers to Support Social Interaction," in *Proceedings of the 3rd IEEE International Symposium on Wearable Computers*: IEEE Computer Society, 1999, p. 191.

[42] S. Milgram, "The Familiar Stranger: An Aspect of Urban Anonymity."

[43] L. Brunnberg, "The Road Rager: making use of traffic encounters in a mobile multiplayer game," in *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia* College Park, Maryland: ACM, 2004.

[44] L. Brunnberg and O. Juhlin, "Movement and Spatiality in a Gaming Situation - Boosting Mobile Computer Games with the Highway Experience," in *Interact*, Zürich, Switzerland, 2003.

[45] G. Kortuem and Z. Segall, "Wearable communities: augmenting social networks with wearable computers.," in *Pervasive Computing*, 2003, pp. 71-78.

[46] T. Michael, D. M. Elizabeth, R. Kathy, and L. Darren, "Social net: using patterns of physical proximity over time to infer shared interests," in *CHI '02 extended abstracts on Human factors in computing systems* Minneapolis, Minnesota, USA: ACM, 2002, pp. 816-817.

[47] P. Dahlberg, J. Redstr, and H. Fagrell, "People, places and the newspilot," in *CHI '99 extended abstracts on Human factors in computing systems* Pittsburgh, Pennsylvania: ACM, 1999, pp. 322-323.

[48] E. O'Neill, V. Kostakos, T. Kindberg, A. F. g. Schieck, A. Penn, D. S. Fraser, and T. Jones, "Instrumenting the city: developing methods for observing and understanding the digital cityscape," in *Ubicomp*, 2006.

[49] T. Kindberg and T. Jones, "“Merolyn the Phone”: A Study of Bluetooth Naming Practices " in *Ubicomp*, 2007, pp. 318-335.

[50] M. Esbjörnsson and M. Östergren, "Hocman: supporting mobile group collaboration," in *CHI '02 extended abstracts on Human factors in computing systems*, Minneapolis, Minnesota, 2002, pp. 838-839.

[51] V. Kostakos and E. O’Neill, "Quantifying the effects of space on encounter," in *Space Syntax Symposium*, Istanbul, 2007.

[52] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Duke University 2000.

[53] L. Aalto, N. G, thlin, J. Korhonen, and T. Ojala, "Bluetooth and WAP push based location-aware mobile advertising system," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services* Boston, MA, USA: ACM, 2004.

[54] T. Finin, O. Ratsimor, A. Joshi, and Y. Yesha, "eNcentive: A Framework for Intelligent Marketing in Mobile Peer-To-Peer Environments," in *International Conference on Electronic Commerce (ICEC)*, 2003.

[55] Anonymous and Cachelogic, "The true picture of peer-to-peer filesharing," in *a presentation available online at: http://www.cachelogic.com/research/index.php*, 2005.

[56] M. W. Newman, A. Voida, R. E. Grinter, N. Ducheneaut, and K. Edwards, "Listening in: practices surrounding iTunes music sharing," in *CHI '05: Proceeding of the SIGCHI conference on Human factors in computing systems*, Portland, Oregon, USA, 2005, pp. 191--200.

[57] J. Hightower, S. Consolvo, A. LaMarca, I. Smith, and J. Hughes, "Learning and Recognizing the Places We Go," *Lecture Notes in Computer Science,* vol. 3660, pp. 159-176, 2005.

[58] N. Marmasse and C. Schmandt, "Location-Aware Information Delivery with ComMotion," in *2nd international symposium on Handheld and Ubiquitous Computing*, Bristol, UK, 2000, pp. 157-171.

[59] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "MovieLens unplugged: experiences with an occasionally connected recommender system," in *8th international conference on Intelligent user interfaces*, Miami, Florida, USA, 2003, pp. 263-266.

[60] R. Opperman, M. Specht, and I. Jaceniak, "Hippie, A Nomadic Information System," in *Proc. 1st international symposium on Handheld and Ubiquitous Computing*, Karlsruhe, Germany, 1999, pp. 330-333.

[61] B. Schiele, T. Jebera, and N. Oliver, "Sensory-Augmented Computing: Wearing the Museum's Guide," *IEEE Micro,* vol. 21, pp. 44-52, 2001.

[62] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou, "Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences," in *Human Factors in Computing Systems*, The Hague, Amsterdam, 2000, pp. 17-24.

[63] K. Cheverst, N. Davies, K. Mitchell, and A. Friday, "Experiences of developing and deploying a context-aware tourist guide: the GUIDE project," in *Mobile Computing and Networking*, Boston, MA, USA, 2000, pp. 20-31.

[64] K. Cheverst, K. Mitchell, and N. Davies, "The role of adaptive hypermedia in a context-aware tourist GUIDE," *Commun. ACM,* vol. 45, pp. 47-51, 2002.

[65] N. Mitrovic and E. Mena, "Adaptive User Interface for Mobile Devices," in *Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification*, 2002, pp. 29--43.

[66] D. Vogel and P. Baudisch, "Shift: A Technique for Operating Pen-Based Interfaces Using Touch," in *CHI*, San Jose, CA, 2007, pp. 251-260.

[67] D. Crow and B. Smith, "The role of built-in knowledge in adaptive interface systems," in *Proceedings of the 1st international conference on Intelligent user interfaces*, Orlando, Florida, United States, 1993, pp. 97-104.

[68] F. Linton, D. Joy, H.-P. Schaefer, and A. Charron, "OWL: a recommender system for organization-wide learning.," *Educational Technology & Society,* pp. 62-76, 2000.

[69] J. M. Carroll and C. Carrithers, "Training wheels in a user interface," *Communications of the ACM,* vol. 27, pp. 800-806, August 1984 1984.

[70] J. B. Black, J. M. Carroll, and S. M. McGuigan, "What kind of minimal instruction manual is the most effective," in *SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface*, Toronto, Ontario, Canada, 1986, pp. 159-162.

[71] L. Findlater and J. McGrenere, "A comparison of static, adaptive, and adaptable menus," in *Proceedings of the 2004 conference on Human factors in computing systems*, Vienna, Austria, 2004, pp. 89-96.

[72] E. Horvitz, "Principles of mixed-initiative user interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, Pittsburgh, Pennsylvania, United States, 1999, pp. 159-166.

[73] L. Birnbaum, E. Horvitz, D. Kurlander, H. Lieberman, J. Marks, and S. Roth, "Compelling intelligent user interfaces: how much AI?," in *Proceedings of the 2nd international conference on Intelligent user interfaces*, Orlando, Florida, United States, 1997, pp. 173-175.

[74] A. Cypher, "Eager: Programming Repetitive Tasks by Example," in *CHI*, New Orleans, 1991, pp. 33-39.

[75] Bao, "Fewer Clicks and Less Frustration: Reducing the Cost of Reaching the Right Folder," in *Intelligent User Interfaces*, 2006.

[76] W. E. Mackay, "Patterns of sharing customizable software," in *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, Los Angeles, California, United States, 1990, pp. 209-221.

[77] M. Allan, C. Kathleen, L. Lennart, vstrand, and M. Thomas, "User-tailorable systems: pressing the issues with buttons," in *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people* Seattle, Washington, United States: ACM, 1990, pp. 175-182.

[78] M. W. Newman, S. Izadi, W. K. Edwards, J. Z. Sedivy, and T. F. Smith, "User interfaces when and where they are needed: an infrastructure for recombinant computing," in *Symposium on User Interface Software and Technology*, Paris, France, 2002, pp. 171-180.

[79] M. W. Newman, J. Z. Sedivy, C. M. Neuwirth, W. K. Edwards, J. I. Hong, S. Izadi, K. Marcelo, T. F. Smith, and J. Sedivy, "Designing for serendipity: supporting end-user configuration of ubiquitous computing environments," in *Symposium on Designing Interactive Systems*, London, England, 2002.

[80] W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith, and S. Izadi, "Challenge: Recombinant Computing and the Speakeasy Approach," in *International Conference on Mobile Computing and Networking*, Atlanta, Georgia, USA, 2002, pp. 279-286.

[81] J. A. Kim, O.-C. Kwon, J. Lee, and G.-S. Shin, "Component adaptation using adaptation pattern components," in *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, Tucson, AZ USA, 2001, pp. 1025-1029.

[82] J. Kramer and J. Magee, "The Evolving Philosophers Problem: Dynamic Change Management," *IEEE Trans. Softw. Eng.,* vol. 16, pp. 1293-1306, 1990.

[83] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems,* vol. 14, pp. 54-62, 1999.

[84] S.-W. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, and P. Steenkiste, "Exploiting Architectural Style for Self-repairing Systems," 2002.

[85] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *Proceedings of the 20th international conference on Software engineering*, Kyoto, Japan, 1998, pp. 177-186.

[86] H. Cervantes and R. Hall, "A Framework for Constructing Adaptive Component-Based Applications: Concepts and Experiences," in *CBSE*, 2004.

[87] D. Linthicum, "B2B Application Integration: e-Business-Enable Your Enterprise," 2001.

[88] A. Gaddah and T. Kunz, "A Survey of Middleware Paradigms for Mobile Computing," Carleton University Systems and Computing Engineering 2003.

[89] B. Brumitt and J. J. Cadiz, ""Let There Be Light" Examining Interfaces for Homes of the Future," in *INTERACT*, 2001.

[90] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Principles of distributed computing*, Vancouver, BC, Canada, 1987, pp. 1-12.

[91] M. Hall and P. Gray, "Mobile Support for Team-Based Field Surveys," in *6th International Symposium on Mobile HCI*, 2004, pp. 431-435.

[92] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta, "Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones," in *Mobisys*, 2007.

[93] A. Bassoli, J. Brewer, and K. Martin, "undersound: Music and Mobility Under the City," in *Ubicomp*, 2006.

[94] M. Esbj, rnsson, O. Juhlin, Mattias, and stergren, "Traffic encounters and Hocman: associating motorcycle ethnography with design," *Personal Ubiquitous Comput.,* vol. 8, pp. 92-99, 2004.

[95] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*: MIT Press, 2004.

[96] J. Ametller, S. Robles, and J. A. Ortega-Ruiz, "Self-Protected Mobile Agents," *Proc. Joint Conference on Autonomous Agents and Multiagent Systems,* vol. 1, 2004.

[97] J. Page, A. Zaslavsky, and M. Indrawan, "A buddy model of security for mobile agent communities operating in pervasive scenarios," *Proceedings of Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation,* vol. 32, 2004.

[98] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Trusting Mobile User Devices and Security Modules," *Computer,* vol. 30-2, pp. 61-68, 1997.

[99] P. L. Kocher, R. McGraw, G. Raghunathan, A. Ravi, S. , "Security as a new dimension in embedded system design," in *Design Automation*, 2004, pp. 753-760.

[100] E. Guy, A. Anurag, and C. Vipin, "History-based access control for mobile code," in *Proceedings of the 5th ACM conference on Computer and communications security* San Francisco, California, United States: ACM, 1998.

[101] J. Goldenberg, Y. Shavitt, E. Shir, and S. Solomon, "Distributive immunization of networks against viruses using the ‘honey-pot’ architecture," *Nature Physics,* vol. 1, 2005.

[102] F. Chen and W. Yeager, "Poblano: A Distributed Trust Model for Peer-to-Peer Networks," in *Technical Report, TR-I4-02-08* Palo Alto: Sun Microsystems, 2002.

[103] M. Saeb, M. Hamza, and A. Soliman, "Protecting Mobile Agents against Malicious Host Attacks," in *Smart Objects Conference*, Grenoble, France, 2003.

[104] D. Birsan, "On plug-ins and extensible architectures," *Queue,* vol. 3, pp. 40–46, 2005.