UNIVERSITY *of* GLASGOW

Guidelines and infrastructure for the design and implementation of highly adaptive, context-aware, mobile, peer-to-peer systems

Marek Bell

Doctor of Philosophy

Department of Computing Science

Faculty of Information and Mathematical Sciences

University of Glasgow

2007

Abstract

This work investigates why much of the mobile software available today largely fails to fulfil earlier ubicomp visions of over a decade ago. Despite the fact that modern hardware is more than capable of delivering the systems detailed in early ubicomp visions, modern mobile software is found to be generally static and inflexible, rather than adaptive and context-aware as ubicomp had expected. As a result, this work attempts to answer two research questions:

RQ1 *How can mobile developers design and develop more flexible and context-aware mobile systems?*

RQ2 *Are there software components lacking in the mobile field, hindering the development of flexible, context-aware mobile systems?*

Through a thorough review of existing literature, and extensive study of two large ubicomp systems, problems are identified with current mobile design practices, infrastructures and a lack of required software. From these problems, a set of guidelines for the design of mobile, peer-to-peer, context-aware systems are derived.

Four key items of software infrastructure that are desirable but currently unavailable for mobile systems are identified. Each of these items of software are subsequently implemented, and the thesis describes each one, and at least one system in which each was used and trialled. These four items of mobile software infrastructure are:

An 802.11 wireless driver that is capable of automatically switching between ad hoc and infrastructure networks when appropriate, combined with a peer discovery mechanism that can be used to identify peers and the services running and available on them.

A hybrid positioning system that combines GPS, 802.11 and GSM positioning techniques to deliver location information that is almost constantly available, and can collect further 802.11 and GSM node samples during normal use of the system.

A distributed recommendation system that, in addition to providing standard recommendations, can determine the relevance of data stored on the mobile device. This information is used by the same system to prioritise data when exchanging information with peers and to determine data that may be culled when the system is low on storage space without greatly affecting overall system performance.

An infrastructure for creating highly adaptive, context-aware mobile applications. The Domino infrastructure allows software functionality to be recommended, exchanged between peers, installed, and executed, at runtime.

Table of Contents

List of Figures

List of Tables

Acknowledgements

As my research is part of the Equator group I have been fortunate to have met and worked closely with an extremely large number of excellent and respected researchers whilst undertaking my Ph.D. I would like to thank all those in Equator that I have worked with on projects or who have been kind enough to give advice and guidance on my research.

As is clear from the thesis text, the majority of my work has been conducted with my colleagues at the University of Glasgow and I thank all of them, including those who have since moved on from Glasgow, for their high levels of commitment and dedication at every stage of each project.

In addition to his work in coding systems and writing papers, I would like to thank Barry Brown for offering consistently sound advice in focusing my research on particular areas, and for always making time to offer this advice when I needed it.

Thanks to my second supervisor, Phil Gray, who has been an invaluable mentor; always providing extremely concise and logical opinions on my ideas and work as well as clearly identifying the most important aspects of my research.

Throughout the entire duration of my research for this thesis two people, Malcolm Hall and Matthew Chalmers, have provided more help and advice than I could reasonably have expected. As can be seen from the work in this thesis, Malcolm, a fellow Ph.D. student, has worked equally with me on almost every single application and project described. He is the finest coder I know and an excellent and determined researcher; as well as a trusted and valued friend.

Finally, my supervisor, Matthew, has been a great inspiration and adviser at every level of my work. His devotion to his own research and his enthusiasm for others' work in the field is the greatest inspiration a student could have.

With the exception of the Navizon application, the work described within this thesis has been funded by EPSRC grant GR/N15986/01.

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, particularly Malcolm Hall.

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, with the exceptions of the *Lighthouse* and *FarCry*. Moreover, contributions from others have been made only to system design and implementation – the work and research related to the thesis itself are entirely the author's own.

The concept of *Seamful Design*, discussed in section 4.2.2 of Chapter 4, is primarily that of Matthew Chalmers and not the author's. The author's work did not lead to the idea of *Seamful Design*, rather the author merely applies it in the mobile environment and highlights its importance within the mobile area.

List of Publications

The following is a list of publications for which the author has been either a primary author or a co-author, and which are related to, or have influenced, the work in this thesis.

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.

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 Exploting Context Histories in Smart Environments (ECHISE), 2005

Barry Brown and Marek Bell, *CSCW at play: 'There' as a collaborative virtual environment,* Proc. CSCW 2004, Chicago IL, ACM Press

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

Barry Brown and Marek Bell *Social Interaction in 'There'* Proc. CHI 2004, Vienna, Austria, p1465-1468, ACM Press

Matthew Chalmers, Ian MacColl and Marek Bell, *Showing the Seams in Wearable Computing,* Proc. IEEE Eurowearable 2003, Birmingham, pp. 11-17

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

1 INTRODUCTION

Throughout the last decade, mobile and embedded digital devices have begun to proliferate among average users, becoming part of their everyday lives. While less than a decade ago it was uncommon for the average person to carry any digital device with them—other than perhaps a watch—today it is not at all surprising to find people carrying digital cameras, music players and PDAs. Not only are these items becoming far more common, many in the western world view them as necessities and would not travel without their digital devices. A modern mobile phone typically has significantly more processing power than the average desktop machine of less than a decade ago whilst many of today's PDAs are more powerful than the desktop systems of less than five years ago.

Just as the devices we carry have vastly improved and multiplied, the digital infrastructure that surrounds us has grown exponentially. At any location in a high street of modern city it is common for twenty to thirty digital devices to be in range—many capable of transmitting and receiving data to other devices. These may be others' digital devices carried on their person, 802.11 access points, GSM cell towers or Bluetooth beacons.

Along with the spread of mobile and embedded devices have come vast improvements in the methods through which such devices communicate. Wireless 802.11 has experienced an almost unbelievable jump in speed, and the current version is over 100 times faster than the original 1997 version while the impending 802.11n, already available in draft versions, is planned to be over 500 times faster. Bluetooth is now available on nearly every mobile phone and PDA; embedded in everything from standard desktop computers to in-ear headphones, it can provide a relatively low power communication solution to body or room sized environments.

Mobile and embedded devices are continually using their communication devices to scan the area—they are capable of detecting and communicating with the multitude of different device types that surround them. They are often synchronised with an owner's email, files and schedule and, through GPS, wireless or cell positioning techniques, can be aware of their location as they are carried throughout the day.

The modern mobile device—be it a powerful phone or a PDA—is in a unique and novel position. It is capable of sensing more of its surroundings than any other type of computer has previously been, loaded with its owner's schedule, email and files, it is able to detect, and more importantly, communicate with the other devices it passes, and it is able to find its own position in the world. With this wealth of contextual information about itself, its owner and its surroundings, the modern mobile device might have brought about a vast change in the manner in which we interact with technology in our work lives, social encounters and daily travels.

However, despite huge improvements in the digital devices we carry and those embedded in the environment around us—and the remarkable amount of context information these devices are capable of gathering—the applications that run on the mobile devices are commonly just reduced versions of desktop software. They remain static and inflexible, often completely unaware of the mobile device's environment or a user's current situation. As these devices do not bring anything particularly unique or novel further to anything a desktop machine does, they do not affect our lives to any great extent. For example, on the latest version of the Pocket PC, the most widely used mobile operating system, the installed applications include Calculator, Excel, Notes, PowerPoint, Tasks and Word – all of which are just cut-down versions of desktop applications rather than location or context-aware applications.

A standard PDA is capable of detecting other PDAs, phones and computers nearby. Why then, should it not interact with these devices? If, for example, a user wishes to find the latest news but cannot connect to the Internet why can their device not communicate with those nearby to find this information? Similarly, if a user requires a certain software codec or module to play an audio file, why can their device not request and obtain a module from a nearby peer? With the exception of a cut down version of the desktop MSN messaging client and a very basic file 'beam' utility, there are no applications included as standard with the majority of mobile devices that even allow the discovery of other devices or communication with peers.

As mobile devices can now be almost constantly aware of their surroundings and constantly connected to the Internet, why is it not an easy task to locate a friend who is currently carrying his or her mobile device? If a user has been to a particularly good restaurant in the past but can not remember its location why can their mobile device not remind him or her? Mobile systems, and mobile applications, have failed to fulfil earlier promises of ubiquitous computing, which envisioned that mobile devices and software would learn a user's context, adapt and behave appropriately within that context, and remain unobtrusive until functionality was desired or apt.

Barkhuus and Dourish seem to be aware of the current situation and describe both opportunities and problems with current mobile devices [5]:

> *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.*

Dourish later considers, along with Genevieve Bell, that perhaps early visions of ubicomp will never be fulfilled, or that we should now accept that ubicomp is evident in the world today, but that it has taken

a form distinct from that imagined previously [56]. Dourish and Bell argue that if, as they propose, ubicomp is already present in the world today, developers and designers should concentrate more on solving problems evident now—rather than continually planning for a proximate ubicomp future that may never come. That is, rather than assuming problems such as communication between mobile devices, context-sensing, or adaptation will become insignificant in the future, we should attempt to address such problems now. The work in this thesis generally aligns with this view, and attempts to analyse and address many current issues experienced with mobile devices.

It seems that we are on the edge of a paradigm shift in how mobile devices are used. There is no hardware lacking for us to achieve a large number of tasks that account for a user's context and would make his or her mobile life much easier by providing timely and suitable information in the many different situations he or she may encounter whilst mobile. Indeed, many in the western world are already familiar with or already own the devices that could run these types of applications. However, as current hardware is indeed capable of supporting flexible, context-aware mobile applications there must be problems which remain in software, business models or in the development community that are obstructing the creation of such applications.

While the progress in IDEs and compilers for desktop machines has almost matched the progress of mobile device hardware over the last decade, developing for mobile devices remains a struggle, and the knowledge and infrastructure to do so is often lacking. The design space of the mobile device has not been fully explored, and the software tools and libraries that would aid developers in writing powerful and flexible mobile applications simply do not yet exist. There are few guidelines aimed specifically at the mobile developer, and many software libraries and techniques that are well documented and easily implemented on desktop machines have not even been attempted on mobile devices. Without the knowledge and software infrastructure to utilise the potential of the mobile device fully, developers are left with little choice but to take existing desktop applications and squeeze them down to fit onto mobile devices. This leaves mobile device applications rather bland and far less useful than they could be. Currently, the mobile device—and the information it carries—becomes redundant whenever a desktop machine is available and this should not be the case if the mobile device were providing a uniquely novel service.

This thesis aims to identify what particular failures in design and infrastructure have led to the situation where the majority of mobile applications largely neglect the opportunities to utilise the full spectrum of contextual information available on mobile devices. By identifying previous failures and investigating the current state of the mobile research field, the thesis hopes to provide suitable guidelines and infrastructure that address existing problems and lead to the design and implementation of more flexible, context-aware, mobile, peer-to-peer systems.

1.1 Research Questions

In order to support the future development of mobile applications that are more mobile, make better use of the sensing technologies they have and communicate well with their peers this thesis investigates two issues. These are:

RQ1 *How can mobile developers design and develop more flexible and context-aware mobile systems?*

RQ2 *Are there software components lacking in the mobile field, hindering the development of flexible, context-aware mobile systems?*

Through a study of existing literature, the thesis attempts to address the first research question by identifying why mobile applications have failed to develop in unique and novel ways, distinct from desktop systems. By identifying current problems and why they exist, it is possible to suggest solutions or alternative design strategies for mobile development that may lead to the design and implementation of more flexible and context-aware mobile applications. This process is aided by the analysis of one existing mobile system and the implementation and analysis of a second mobile system. Through practical experience of developing and trialling mobile systems in this way, further issues may be identified and addressed which, in turn, contribute to addressing the first research question.

The same process of reviewing existing literature and implementing and analysing mobile systems helps to identify software components and infrastructure that may be lacking currently in the mobile field. After identifying missing components they are implemented and trialled to demonstrate their benefit and their applicability to the second research question.

1.2 Scope and Approach

Many of the strengths of the research within this thesis are the result of work that has been conducted as part of the Equator IRC[1]. Equator has been fortunate in that it has an extremely large number of resources and researchers, with numerous and varied skills. Equator provides a unique environment for a student, one in which a pragmatic approach to a wide variety of systems is possible, and in many ways necessary.

A substantial understanding of the design process, user experience, and system context of mobile systems has been gained over the course of many systems and trials, which facilitates a holistic approach to issues within the mobile field. Since core goals of this thesis are to explore the mobile peer-to-peer field, offer guidelines for the development of mobile systems, and identify and implement infrastructure that may benefit the field, a broader scope and holistic approach are indeed applicable. The approach of this thesis to mobile peer-to-peer systems spans architectural, interactional and user acceptance issues—rather than focusing solely on one single aspect of technology or infrastructure.

[1] http://www.equator.ac.uk/

Each of the individual systems described, and studies thereof, contribute an important but partial understanding of the mobile field. However, when analysed together, a clearer holistic view develops—one in which issues that apply to mobile systems generally become more readily apparent. This broader view contributes a solid basis for subsequent work, allowing issues to be identified more clearly, and generally applicable solutions to be generated.

The author believes that a holistic approach, spanning many systems and examining them pragmatically, is necessary in order to gain the insight and experience required to establish general guidelines, and identify and implement required infrastructure, as this thesis does. The research in this thesis aims to tackle the problem within modern mobile systems of applications and systems generally failing to fulfil earlier expectations related to ubiquitous computing. The majority of modern mobile systems remain generally static reduced functionality versions of desktop applications, rather than providing appropriate and novel functionality to the mobile user based on his or her current task and context. It is unlikely that research focused on only one system, or a single aspect of technology, could adequately identify the range of issues that have contributed to this situation in the course of a single thesis. Rather, the thorough investigation and testing of a large number of mobile systems, exploring many varied issues, is more appropriate, providing a more complete answer to why modern mobile systems still suffer from problems of inflexibility and immobility.

Whilst a broad holistic approach has been of great benefit to the research described in the thesis, in retrospect, it may also have been a necessity of any work conducted within the Equator Glasgow group. Equator's rapid progress and large number of researchers result in an environment in which many systems and ideas are implemented, tested and analysed in relatively short periods of time. Consequently, this continual flux of systems means that it has been challenging to maintain an in-depth focus on only a single aspect of design or implementation within one system, or across several systems. The Equator environment favours a holistic approach in general, and the research presented here relies on the pragmatic testing of an extensive number of systems to drive such an approach.

1.3 Thesis Walkthrough

The thesis starts with a review of existing literature primarily based in the field of mobile, peer-to-peer systems. However, where necessary literature from context-aware computing and legal fields are presented and discussed. The findings of the literature review highlight several problems remaining in the mobile, peer-to-peer field; including design problems and problems in rapidly and efficiently developing applications due to the lack of several key infrastructure components for mobile devices.

Chapter 3 uses the results from the literature review to focus the analysis of two mobile systems. This analysis results in the confirmation and strengthening of many of the findings of the literature review, as well as the formation of several guidelines that aim to address some of the failings in existing mobile systems.

After Chapter 3 the focus of the thesis changes from investigating mobile systems and extracting guidelines to the actual implementation and trialling of four key pieces of infrastructure which appear to be required for the creation of adaptive, context-aware mobile peer-to-peer systems but are currently missing from the field. However, when experience with or trials of this infrastructure do raise or draw attention to new issues further guidelines are still drawn out where appropriate. Each of the following four chapters (4-7) details the implementation and demonstration of one of the pieces of infrastructure identified as important from the first half of the thesis.

Finally, Chapter 8 reviews the thesis and the contributions it has made, as well as contemplating possible future work in the field.

2 Initial investigation of mobile, peer-to-peer systems

Although there have been numerous research-based mobile systems—many of which are referenced throughout this thesis—there have been relatively few attempts to analyse or categorise the features that led to successful mobile systems or to categorise the possible types of mobile systems. The lack of material that directly contrasts and compares the infrastructure and technology used in mobile systems, rather than discussing a single, isolated system, makes it challenging to identify the areas that contribute to a mobile, peer-to-peer system's success or failure. Although an investigation and trials related to these issues are part of this thesis and are described in Chapter 3, it is prudent to begin the investigation by examining existing systems and attempting to discover factors that led to their success—even if authors often do not explicitly state or discuss these issues themselves.

This review of existing work attempts to pull together isolated mobile, peer-to-peer systems into some high-level categories in the hope of identifying some clear topics that affect a mobile system's performance.

2.1 Mobile context-aware systems

One of the aims of the work in this thesis is to address the problem that mobile device applications are often static, reduced functionality versions of applications available for desktop computers. It has been stated that the primary way to achieve this is to take advantage of the greatly increased level of context information available on mobile devices. Therefore, this section reviews some of the literature in context-aware systems.

The use of context is common in many desktop applications – perhaps the most well-known being systems which make people aware of each other's context, such as Portholes [53] and RAVE [67]. However, Schilit et al. were among the first to examine the value of context in mobile computing [149]:

> *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.*

It is clear from the onset of the work involving context in mobile environments that a system which is reactive and flexible is desirable. Schilit et al. propose several methods in which a mobile system can react to context such as proximate selection in which a device's location is used to filter a list of possible choices [149]:

> Proximate selection *is a user interface technique where the located-objects that are nearby are emphasized or otherwise made easier to choose. In general, proximate selection involves entering two variables, the "locus" and the "selection." However, of particular interest are user interfaces that automatically default the locus to the user's current location.*

Proximate selection demonstrates how contextual information can be employed to filter a list to a more manageable size or adapt the interface to provide a simpler, and more efficient experience to the end user.

Hull et al. similarly discuss the importance of context-awareness for mobile devices, and highlight how correct adaptation is often not just desirable but necessary in many situations [90]:

> *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.*

The quote reveals that in many circumstances it is both appropriate and necessary for mobile devices to be actively *sensing* their environment and *reacting* to it. Although Hull et al.'s research on the subject is almost a decade old, many of the statements they made seem extremely prophetic given the current state of mobile applications [90]:

> *The potential for wearable computers is to be perceived not only as a physical extension of the user, but also as a mental extension. The difference between the two is whether the wearable computer is able to share our awareness of our surroundings and thus operate within a shared context, or whether the device remains "a stranger in the dark", albeit one that you are taking everywhere.*

Examining the majority of current mobile devices and applications, it is apparent that they have failed to become a 'mental extension' as defined and do instead remain "strangers in the dark". Modern mobile devices have the technology to sense their environment, track their owner's context and adapt appropriately but they fail to utilise these features. Instead, nearly all functionality on modern devices is presented statically. The idea of a 'mental extension' seems to conform to Weiser's ubicomp vision of invisible computing [172]:

> *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.*

If, as Hull et al. state, mobile computing is unable to employ context-awareness to adapt reactively and reconfigure, it will always fail to become an invisible mental extension. Thus, it is clear that if mobile devices are to achieve Weiser's vision for ubicomp then it is crucial that they become more adaptive and flexible than they currently are.

Hull et al. identify that one of the crucial items of context to sense is the presence of other people using the system. They identify a requirement of 'a small, low-powered detector that will fit into a wearable computer' as well as stating that size, cost, power-consumption and range are among the most important factors in selecting a technology for achieving this. They implement a prototype system using an external radio tag that is sensed by detector units. Whilst modern mobile devices have evolved to contain 802.11 and/or Bluetooth capabilities that may more appropriately fill the sensing role, the assertion that discovering peer devices is crucial remains valid, as do the important factors identified in selecting a sensing technology. These issues are discussed again later in this chapter as well as in Chapter 4.

While Hull et al. foresaw the problem of mobile devices failing to utilise context in 1997, Messeter et al. provide a more current description of the problem that still exists today [116]:

> *Even if connectivity and location-based services receive a lot of attention in the mobile technology industry, the dominating rhetoric still revolves around providing the functionality of the conventional office environment 'anytime and anywhere', regardless of contextual factors. The nomadic user handles multiple contexts, not by adapting to the particularities of each use situation, but simply by disregarding contextual factors. ...The user becomes nomadic not by adapting to context but rather by detaching from it – 'the context-free user.'*

Messeter et al. are essentially claiming that because modern systems still fail to adapt to the current context, mobile users are forced into the possibly uncomfortable and inefficient situation of simply ignoring context if they are to complete their task. They identify that the majority of mobile applications simply attempt to recreate the functionality found in the standard office environment rather than adapting to integrate smoothly with the varying contexts a mobile device encounters. Messeter at al. describe an ideal system as adapting around the user in this way:

> *However, an ideal context aware system would account for five contextual factors... In this perspective, the vision of the nomadic user is based on continuous adaptation of technology to the specific needs of the current situation. The user becomes nomadic by the system's capability of adapting to constantly changing contexts.*

Messeter et al. have realised that the onus for adaptation is not solely on the user. Instead they make it clear that the system should continually monitor the external environment and adapt around the user.

Pascoe points out that context-aware systems should not make the mistake of concentrating solely on observing the external environment [132]:

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

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. Pascoe also reiterates the sentiments of Hull et al. on the potential value of context information to mobile devices [132]:

> *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 goes on to define a set of core generic capabilities that he views as vital to context-aware systems. These are:

- *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.
- *Contextual adaptation*: the ability of a system to tailor itself to the current situation.
- *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).
- *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 key generic elements of context-aware systems are highly relevant to the work in this thesis and are discussed throughout. Specifically, context sensing and resource discovery with respect to the location of users and the discovery of other devices is discussed later in this chapter in sections 2.2 and 2.4. The discussion therein leads to much of the implemented work discussed further in the thesis. Contextual augmentation is an inherent part of many of the systems implemented as part of this thesis such as *George Square* and *Feeding Yoshi* discussed in Chapters 3 and 4 respectively.

Pascoe believes contextual adaptation to be of core importance:

> *Applications can leverage this contextual knowledge by adapting their behavior 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 difficult 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.*

In short, Pascoe believes contextual adaptation can be of great value but did not have time to implement it in his prototype.

In 1999, Dey and Abowd conducted a survey of existing literature on context-aware systems [50] and attempted to make clearer definitions of what context is, 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.*

As with the previous literature, an emphasis seems to be on the *system services* adapting based on the user's current scenario and past usage habits. This is reinforced after Dey and Abowd review and merge the existing definitions of context to come up with their own [50]:

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

As with previous literature, the authors feel it necessary to state specifically that it is not just external factors, but also the application and system itself, which forms the context.

Dey and Abowd additionally identify what they believe are normally the most imperative items of context [50]:

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

The authors later identify, and show in a table, which of these context types much of the previous work in the field has utilised—allowing readers to make assumptions about how this has affected the systems. It is clear that in the general case the more of these types of context a system makes use of the more it can adapt to a specific user's needs. These findings are important as they allow developers of context-aware systems to focus their efforts on these particular areas.

Dey and Abowd also give their own definition of context-aware [50]:

> *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.*

As will be apparent later in this thesis, the *George Square* and *Domino* systems, produced as part of the work for this thesis, both fit this definition well—and make appropriate use of the four context types identified. Specifically, *George Square* makes use of context to provide relevant information to the user whilst *Domino* uses context information to deliver both relevant information and services.

In 2001, Burrell et al. specifically highlight that context information is highly relevant in the mobile environment [25]:

> *The ability to detect context seems especially relevant to mobile and ubiquitous computing systems which may be used in a variety of different locations, by different users, and/or for different purposes.*

The fact that Burrell et al. identify that mobile systems can be used by different users and for different purposes is of relevance later in the thesis, since an investigation of the *Lighthouse* in Chapter 3 leads to the proposal of a guideline concerning the roles mobile users may assume.

Burrell et al's reminder that context is vital to mobile systems seems to have been heeded and in 2002 there was an increase in the number of context-aware mobile systems being researched. Gellersen et al. examined the viability of augmenting everyday objects with context sensors, and subsequent fusion

of the data from such sensors [69]. From their work, the finding most relevant to this thesis is perhaps that of two types of awareness, direct and indirect.

> *Devices may have direct or indirect awareness of context. In the case of indirect awareness, the entire sensing and processing occurs in the infrastructure while the mobile device obtains its context by means of communication. In contrast, a device has direct awareness if it is able to obtain context autonomously, (more or less) independent of any infrastructure. We will not further consider indirect awareness in this article, as the reliance on sensing infrastructure limits mobile device use to specifically designed smart environments.*

As this thesis is concerned only with mobile devices operating in a mobile environment it is prudent for it to consider primarily devices with direct awareness, for the same reason Gellersen et al. specify. Gellersen et al. also point out that many of the context-aware systems discussed earlier, such as Active Badge, all relied on indirect awareness. However, they also state that direct-awareness applications are becoming rapidly more prevalent [69]:

> *However, these pioneering projects all employed indirect awareness with sensors located in the infrastructure, listening for beacons sent out from the mobile devices.*
>
> ...
>
> *More recent work has increasingly considered the embedding of direct awareness in mobile devices. This has been boosted by rapid advances in sensor technologies...*

It is indeed clear that recently there has been a drive toward devices that are "all-in-one" and come with many sensing technologies. It is now not surprising to find mobile devices that have Bluetooth, 802.11, GSM/GPRS, GPS, IR and a camera all built-in on a single device. Each one of these sensing technologies can detect different information about the surrounding environment and, in addition, much of the user's data, such as calendar information, is likely to be available on the device. In short, the technology has now advanced so that there is little or no need to rely on indirect awareness for most systems. This is yet another reason for this thesis to concentrate primarily on direct awareness systems.

To support direct-awareness, there is a drive in the research community to embed sensors into everyday artefacts and clothing [89], [159], [94], [103], [142]. Randell and Muller describe the *Well Mannered Wearable Computer* which is a jacket with embedded GPS unit and accelerometer. The additional sensors allow behaviour suitable for the user's current context to be exhibited [142]:

> *...we control the media rendering to ensure that the presentation of information is appropriate to the user's activity and consistent with the sensed event. For example, we do not wish to render any information if the user is running; we also know that if the user is not moving then an event triggered by a change in location cannot be generated. A*

> *simple set of rules can thus be formulated to form an etiquette for our wearable computer.*

Whilst it is apparent that sensors are indeed being increasingly embedded into everyday devices the work in this thesis concentrates primarily on PDAs and phones – utilising only their inbuilt sensor functionality or that of upgrades that can be easily applied to augment them. Whilst Randell and Muller demonstrate just how appropriate adaptation facilitated by increased embedded sensors can be in aiding a system in selecting appropriate behaviour for the user's current context, the basic concepts are certainly applicable to the mobile devices used throughout this thesis and do not rely on embedded sensors specifically.

An example of a system that makes good use of context information made available by the increased number of sensing technologies available in a mobile environment was developed by Baus et al. in 2002 [8]. Their system attempts to provide a seamless changeover of both UI and sensing technologies in a way-finding application when a user crosses from an indoor to an outdoor environment or vice versa. The authors detail the ways in which their system must adapt:

> *When a change of the means of transportation is detected, the system has to adapt its interface to the new situative constraints. One essential issue in this context is the switch between different positioning technologies... the presentation of the way description has to be adapted to different output devices that may be used. This includes adaptations to the screen size, resolution, and colour capabilities of mobile and stationary devices.*

It is clear that the level of system adaptation is extensive, yet this seemingly extreme level of adaptation may be common and necessary in the mobile environment where context changes can often be extreme. For example, users can easily and quickly move from a quiet, isolated office environment to a noisy, busy street, and any mobile devices they are carrying should rapidly sense and adapt to the context change if they are to behave appropriately in the new environment.

For over a decade the literature has almost unanimously agreed that context-aware systems can be of significant value in the mobile environment. It is apparent that any context-aware system must be capable of continually monitoring not only the external environment but also its own state and history of use. In order to integrate smoothly in the many differing contexts a mobile device may be used in, the level of adaptation a system must be capable of is often extremely extensive. Thus, mobile applications should be highly dynamic, at the core system level if they are to prove successful in such an environment.

Context, context-awareness, and the benefits context awareness may provide to mobile and ubiquitous systems have been thoroughly investigated in previous work. However, it is clear that there are few mobile systems that are actually able to achieve both the awareness and flexibility required to succeed.

As Hull et al. warned, and Messeter et al. agree, this is borne out today in that most mobile applications remain static, inflexible versions of desktop applications—forcing mobile users simply to ignore contextual factors if they wish to use those applications.

If truly mobile systems are to gain popularity, this problem must be addressed, and an infrastructure which allows extreme adaptation of a system's core functionality to suit the user's activity must be implemented. The literature points out that such a system must have sensing, adaptation and resource discovery capabilities, and that location, identity, activity and time are vital pieces of information to base such a system around. The work in this thesis attempts to make initial, but useful, steps in investigating and implementing such an infrastructure. The implementation, an example game, and trial are presented in Chapter 7.

Whilst it is apparent that there are many researchers who have investigated context and context awareness, it is clear that there remains a tension between two main views of how context may be interpreted. Both Dey and Abowd [50] and Schilit et al. [149] seem to perceive context as a flow of static elements that can be sensed as they occur but, as time passes, remain constant in definition. Both Dourish [55] and Chalmers [30] take an alternative view that whilst contextual information can be sensed, it may be continually reinterpreted as time passes and new activities and information occur. In this view context at any specific time is not static in meaning and may require re-examination if it is to be used at a later date. Later work in this thesis, particularly that in Chapter 7, relies more on the latter view that contextual information must be continually reinterpreted if it is to be reused as users often extend and redefine contexts with updated information. This view results in the *Domino* infrastructure, described subsequently, that accounts for such reinterpretation, as well as opening the possibility of future work that attempts to support the use of context in this way even further.

2.2 Communication technologies

Perhaps the most surprising omission in the previous research in mobile peer-to-peer systems is in the topic of communication technologies. Although every mobile, peer-to-peer system must employ some form of communication technology to discover and communicate with peers, most work simply states the type used for the particular implemented system without stating the benefits or failings of the technology—or giving detailed reasons for selecting it above other possibilities. Whist much previous work covers many technical issues involved in mobile systems, few apply or discuss the effects or problems of specific technologies or infrastructures in implemented, demonstrable mobile systems. Furthermore, there has been little work in directly comparing the various technologies available with one another. To understand this problem, it may be beneficial to give some examples.

For example, whilst Bassoli et al. generally describe their tunA system in great detail, they briefly gloss over the choice of communication technology used in tunA [7], stating that:

> *Nevertheless among the existing technologies [802.11] Wi-Fi seemed the most suited to obtain the ad-hoc connectivity between devices, allowing a higher bandwidth and range compared to other wireless standards (i.e. Bluetooth).*

This single sentence description of the communications technology used is insufficient to explain the advantages and disadvantages that arose in the system as a result of this choice. Furthermore, although it states 802.11 has a higher bandwidth and greater range than Bluetooth, it does not examine any of the other factors that might have led to the decision, such as peer discovery rates or reliability of the connection.

Similarly, although Flintham et al. provide a generally thorough investigation of the *Can You See Me Now* game, go into much depth about the positioning technology used, and the use of audio and the strategies employed by players of the game, the actual description of the communication infrastructure is extremely brief and does not provide a clear picture of the setup used [64]. Indeed, the only references to communications on the devices themselves within the paper are: '...talk was then digitally encoded and streamed to the players over the Internet' and 'The runners' interface was delivered to them... from a server in a nearby building over a 802.11b wireless local area network'. This short description poses many questions about the communication infrastructure and setup. For example, basic issues like reliability and the density of coverage required to allow smooth operation of the system are left unanswered. Furthermore, it is impossible to tell from the paper whether existing 802.11 infrastructure already present was used in the game area or if it was necessary to install additional equipment.

These examples are typical of the problem where work applying certain technologies to useable mobile systems is thorough in many areas but makes sparse commentary on the communication technologies, or other technologies, that drive them. As most authors omit details about the communications infrastructure and technology they use to implement mobile peer-to-peer systems, it is extremely difficult to draw comparisons between them in order to discover which are most suitable for use in peer-to-peer environments. For this reason, a direct comparison is conducted as part of this thesis in Chapter 4.

Obviously, as the mobile area is already extremely popular and continues to grow, with so many papers and articles being written there are exceptions. Cheverst et al. provide uncommonly detailed descriptions and opinions on the communications technology they used in [37] and [38] – both papers on the GUIDE system. The papers describe the 802.11 setup used, stating that 6 APs had to be installed in the target area and that each provided bandwidth of 2MB/s. They also give diagrams of the network setup and explain that a centralised server was required to drive the system. This level of detail on underlying mobile technology and infrastructure, and how it affects a real system is, unfortunately, rare. The majority of work on mobile applications is either purely technical and theory

based or concentrates solely on the user experience of a finished system—ignoring discussion of infrastructure and how it affects the development and use of a finalised system.

The GUIDE system is an appropriate topic within this thesis for other reasons – namely that it has a similar setup to the *George Square* system with a central server and installed access points and that it suffered from the same issues in user positioning. These topics are discussed further in Chapters 3 and 5.

Despite the fact that there is a lack of research into the use of communications technologies actually employed within working mobile systems and their effects, there are a great number of mobile, peer-to-peer systems using numerous different communication technologies. The main two technologies primarily considered are 802.11 and Bluetooth, so these are discussed first.

2.2.1 802.11

Much of the recent leading research in applications which employ 802.11 as a communication medium in mobile systems has come from researchers at The Interactive Institute in Stockholm: such as HocMan, Sound Pryer and Road Rager. Hocman [62] is a system that plays audio notifications to bikers as their vehicles pass on the road. Hocman faced challenges as the bikes are likely to be travelling at high speed and thus the connection time available may be extremely short – a matter of seconds. This short connection time imposed the necessity of pre-configuring each device to be constantly in ad hoc mode with a predefined static IP address and constantly associated with a specific network SSID. This configuration has become known as a Mobile Ad Hoc Network or MANET. The authors 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 utilise 802.11 MANETs successfully as the communication technology of a mobile, peer-to-peer system. The authors also state that 'Operation without an infrastructure fits well with biking since traffic encounters can take place anywhere'. However, this statement can be extended to any mobile system as the location of devices, and that they will be near an infrastructure node when they encounter one another, can never be known.

Whilst the MANET configuration worked well in the Hocman system the research does reveal one of the weaknesses of MANETs: 'MANETs are limited to device-to-device operation, they cannot rely on an infrastructure at any level of networking, such as base stations, routers, or servers'. Another problem with the MANET configuration in Hocman which was not described in published work but is known to exist from conversations with the authors was that the reliability of devices being able to communicate successfully with one another in the trials was lower than expected. It is likely that 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 BSID that was used as a fake infrastructure node two peers in range could both address. As this BSID was created randomly if the two devices were not in range 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. The problems raised from Hocman – that mobile encounters can take place anywhere, that MANETs limit communication only to peers and the BSID problem – are addressed later in this thesis in Chapter 4.

Sound Pryer [128] is another system that was implemented and trialled almost alongside Hocman, and the two clearly share much of the same design, code and infrastructure. Sound Pryer allowed car drivers to "listen in" to music that was being played in a proximate vehicle. Thus, the drivers of two cars travelling in the same direction or waiting in traffic could be aware of the musical tastes of one another. As well as the music delivered over the audio system, Sound Pryer also displayed the colour and silhouette of the nearby car the music was being received from in order to allow users to identify which of the nearby vehicles contained the peer Sound Pryer system the data was coming from.

Sound Pryer primarily differs from Hocman in the average length of time peers were connected to one another. The system was designed to take advantage of the longer connection times by transferring far more data than Hocman did. If time allowed, entire music MP3s could be streamed in addition to the basic information about the peers (such as the car colour and type). The Sound Pryer system demonstrated that 'wireless ad hoc networking is a promising technology for streaming MP3 music files'. Thus, the Hocman and Sound Pryer systems in combination offer preliminary evidence that 802.11 is a suitable choice for both small and large data transfers between peers in mobile, peer-to-peer communities. The fact that Hocman and Sound Pryer are very similar systems, yet differ greatly in the length of time peers are connected to one another, raises an interesting issue: the length of connection time mobile, peer-to-peer systems should be designed to utilise or how they can be designed to make efficient use of both long and short connections. The appropriateness of 802.11 for mobile systems is discussed in greater detail in Chapter 4.

Examining the user experience in Sound Pryer, the authors note that 'an impression of the source of music through vehicle shape and colour gives is satisfying. Many users did understand and use the hints in their attempts to identify the source.' This reveals that users can experience tangible benefits by revealing information to the user about the sources of data and how the infrastructure of the system operates. For example, rather than attempting to create a seamless experience in which the music being delivered from a peer continued to play after the peer moved out of range, Sound Pryer instead provided the information for a user to identify the source of the music. Such information allowed a user to track and identify the original source of the music they were listening to and to accept and, perhaps most importantly, understand why music stopped playing (having identified a peer they would not be surprised that the music stopped when they moved out of range). This revealing of information about infrastructure and data sources can prove more beneficial to the user experience than the alternative of attempting to hide the infrastructure, as when the system fails it can leave the user frustrated and confused as they do not have the necessary information to understand the failure or the knowledge required to rectify it, or to avoid it in the first place. This topic is discussed later in section 2.6 and again in Chapter 4 (section 4.2.2).

Road Rager [22] [23] builds on the Hocman and Sound Pryer systems, creating what is perhaps the most complex 802.11 mobile, peer-to-peer system to come from The Interactive Institute. Road Rager is a game child car passengers can play whilst travelling in the back-seat. They carry PDAs disguised as wands and augmented with rows of LEDs. When players in nearby vehicles are in range, the LEDs light up to alert the player that others are nearby and to give some basic directional information about where they may be located. Various gestures can then be made with the wand that represent various types of attack that can be launched against the peers. Whilst information in both Hocman and Sound Pryer was not time critical, the data transferred in Road Rager is. The authors report a typical game scene observed in their trials that lasts only seven seconds but consists 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 802.11 over MANETs performs extremely well in this scenario to support smooth interaction of this type. 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. Again, this feature is discussed later in Chapter 4.

The three systems from The Interactive Institute discussed here all use 802.11 in ad hoc mode but mobile, peer-to-peer systems can also elect to use infrastructure mode 802.11. Returning to the GUIDE system discussed earlier it is clear from its thorough description of the communications used therein that it relied on an infrastructure 802.11 setup. Traditionally, mobile systems tend to rely on infrastructure 802.11 rather than ad hoc when a large data store is required, as having a permanent server connected to the infrastructure may appear to be the simplest way to provide such a store. However, the GUIDE authors describe some severe disadvantages that result from such a configuration.

Primarily, most areas do not have the required density of coverage to support a continual connection to the server and so must be augmented with additional access points that the system developers install themselves. This problem is repeatedly encountered and was seen in the GUIDE and Can You See Me Now [64] research systems, as well as being common in commercial systems such as Ekahau (www.ekahau.com). The problem occurs due to a heavy reliance on a central server which results in individual devices providing little or no functionality when the server is unavailable, even if there is a large community of peer devices in range. Furthermore, this requirement constrains all these systems to a predefined area – seriously limiting their mobility. In both GUIDE and Can You See Me Now, the access points used had to be adapted with larger and more powerful antennae as well as undergoing time-consuming calibration and range testing. Similarly, the Ekahau system requires that users purchase and install a large number of new access points if the system is to work accurately. However,

such augmentation is costly and the improvements are relatively small, increasing the useable area only marginally.

Mitchell et al. employ a similar 802.11 configuration to that used in GUIDE for their *Six in the City* game [118], which builds on much of the work conducted in GUIDE. As with GUIDE, the authors elect to install their own infrastructure:

> *...a major thrust of the work has been the deployment of a series of wireless 802.11b networks around the University campus and city centre.*

The fact that many developers and designers do go through the costly and time-consuming process of constructing their own 802.11 infrastructure when trialling a system is indeed likely due to the reliance on centralised architectures – *Six in the City*, like GUIDE, does require a central server. Whilst alternatives to centralisation may negate the use of infrastructure access points altogether, and are discussed in section 3.2.6, a secondary cause of designers being forced to implement their own 802.11 networks may be that of the configuration required. In order to use public access points to access central servers, devices must be able to detect public networks, join them and configure their network settings appropriately. Currently, this process is not automated and requires much input from the user at every stage. As the process can be quite complex it is not viable to require trial participants to have the knowledge to enter such details. Furthermore, even if users could potentially achieve the task, it is distracting and time-consuming. Thus, it can be cumbersome in a system which the user is actively interacting with, and can lead to the system failing due to not having a connection if it is designed to be carried in a pocket, periodically providing notifications through audio or vibration alerts. This requirement of continual configuration, often by the user, frequently makes the use of public access points infeasible when relying on the current hardware and software available for 802.11 networking.

As part of the investigatory work for this thesis, two systems which do rely on 802.11 infrastructure setup were studied in detail, the Lighthouse and *George Square* systems, and are discussed in Chapter 3. Subsequently, a solution to configuration problems of centralised 802.11 mobile systems is presented as part of a peer discovery method in Chapter 4.

2.2.2 Bluetooth

Bluetooth was originally conceived as a system for communication between mobile devices, by creating small networks between devices called piconets. It is now extremely common in mobile devices, being shipped within 2 million products every week [86]. However, whilst Bluetooth is widely discussed as a possibility for systems, such as for the *Jabberwockies* in the *Familiar Stranger* system [134], it is actually rarely implemented into mobile, peer-to-peer systems. Four of the rare research mobile systems that were implemented using Bluetooth are MobiTip, DigiDress, Social Net and BlueAware.

In MobiTip, users are able to share recommendations and comments with peers on locations such as shops or museums [146]. When peer devices discover one another they exchange 'tips' which users are immediately notified of but can browse at any subsequent time. Thus users are notified of interesting locations. For example, after encountering a peer in a shopping mall the user may be notified of comments referring to a particular shop or restaurant such as "Great sale on here" or "Serves excellent coffee". MobiTip was initially designed as a direct peer-to-peer system but the addition of permanent Bluetooth stations was later made. This was because it was found that the initial configuration did not seem to exchange enough information during chance peer encounters, and so it was hoped the addition of permanent stations which would collect and share data to all those passing by would aid this situation. Users were told the location of such stations and so, if they found they were not receiving as many tips as they would like, they could actively seek out a station from which to gather a large number of new tips. However, having to make such an addition may lead to many of the problems experienced with infrastructure 802.11 – such as investment in additional nodes, increased setup time due to configuration and calibration of the nodes, and a greatly reduced area in which the system provides full functionality.

DigiDress aims to support social interactions by allowing users to share personal profile information before actually meeting face-to-face [136]. For example, in a bar or a club, DigiDress users can search for peers and, if any are found, view personal information about the owner such as their preferences (favourite music, food, etc). Bluetooth was selected as it is widely available on phones (Nokia phones were used in the trials) and consumes little power when compared to other technologies. DigiDress proved to be extremely successful and gained over 35% of its users through direct peer-to-peer transfer of the application itself using Bluetooth. However, users did report that the Bluetooth was a "barrier" to smooth interaction due to the long scan times and unreliable scan results. Additionally, users rarely made use of the Bluetooth messaging system DigiDress provided, which worked much like an instant messaging client. Instead, after using DigiDress to view a profile of other users, they would normally elect to approach others immediately and interact face-to-face rather than send any messages, even simple ones, over Bluetooth. Unfortunately, it is unclear whether the avoidance of the messaging system was purely due to slow or problematic Bluetooth performance, or was simply because face-to-face interaction is much richer and a more desirable way to introduce oneself.

Terry et al. describe another system that attempts to use Bluetooth to support social interactions. Social Net attempts to identify users' proximity to one another in order to identify opportunities for introducing friends who are commonly in range of one another, but do not yet know one another. Unfortunately, work presented on Social Net, both in [162] and [163] is extremely vague on the technology used to enable peer discovery and communication to occur. However, it is clear that users had to be extremely physically close and stay together for some time before an encounter was recorded. Therefore, it can be assumed that Bluetooth was neither rapid nor robust in discovering peers within the system.

BlueAware [57] uses Bluetooth to provide similar functionality to the Social Net application. BlueAware periodically scans for peers in Bluetooth range, and notifies a server when it encounters one. The server maps the Bluetooth MAC addresses to user profiles, checks if the owners have any similar interests based on their profiles, and notifies both users if they have shared interests. The BlueAware application does a scan for peers once every 5 minutes. Obviously, such a low scan rate results in BlueAware missing most instances when users come into Bluetooth proximity of one another. However, it is adequate to detect when users are together over longer periods; such as students in lectures, customers in a café, or colleagues or friends travelling together. The use of Bluetooth in the BlueAware application highlights Bluetooth's suitability to lower, rather than higher, scan rates.

Ritter et al. provide an analysis of the advantages and failings of Bluetooth before making the unusual move of augmenting standard Bluetooth with EWS (a 433MHz RF system commonly found in automotive remote controls) to create a dual-wireless communications system [145]. A game, *tipspromenad* is described in which multiple-choice questions are posted around the environment and players must search for the questions and answer them when found. The questions are presented on displays connected to small units with both Bluetooth and EWS capability, and the players' mobile devices have both Bluetooth and EWS capability. When players answer questions on their devices the answer is transferred to the nearby question unit using Bluetooth. The unit then uses its EWS unit, which has a longer range than Bluetooth, to transfer the answer to a more distant central server. EWS is also used to limit the play area and detect players attempting to leave (which is considered cheating as whilst outside the game area they are unobserved and may attempt to search for answers from an outside source).

The game configuration allowed the authors to experiment with Bluetooth for diverse communication needs. The authors state that they first considered Bluetooth as it 'is attractive for ad-hoc gaming since it is both available on modern PDAs and mobile telephones, and because it 'facilitates the formation of ad-hoc networks'. However, they find it disadvantageous as it only allows 'a maximum of eight devices' to be used in any one Bluetooth network and as it only has a transmission range of 10 metres. Whilst the range and low number of supportable devices can be overcome with the use of scatternets which combine and bridge Bluetooth piconets, the authors correctly identify problems with scatternet configurations. Firstly, scatternet support must exist both in the hardware and firmware of all Bluetooth devices involved and it is, unfortunately, still quite rare to find off-the-shelf versions that have this support. Secondly, the density of Bluetooth devices required to cover any substantial area would be high, costly and prone to subnets of the network becoming isolated if any one device failed.

The problems the authors encounter also negate some of the advantages of Bluetooth that they initially identify. For example, they state that [145]:

> *While Bluetooth facilitates the formation of ad-hoc networks, its time-consuming inquiry operation leads to long handover times. These handover times restrict the types of games this architecture is suitable for.*

Thus, it is their opinion that Bluetooth requires some form of augmentation if it is to be used over large distances or in mobile applications or games.

2.2.3 GSM/GPRS/3G

GSM/GPRS and 3G are extremely rare in mobile, peer-to-peer systems—although they are common in mobile systems which are not peer-to-peer.

A typical non peer-to-peer system which uses GPRS is that described by Hallberg et al. In their system, designed to examine the enhanced experience of sports events, skiers wore sensors to track their activity levels and the data was sent to a central server over GPRS using a phone they carried [81]. Whilst GPRS is effective since it generally provides a high availability rate due to having a high level of coverage, its low bandwidth makes it unsuitable for continual transference of high levels of data. Hallberg et al's system worked well with GPRS as the sensors generated extremely small amounts of data (only pulse rate, position and speed), which were gathered and updated to the server relatively infrequently.

A research system that uses 3G communication is *I Like Frank* [61] which used '...a 3G mobile phone, a Motorola A835, rather than a PDA' [13]. *I Like Frank* was similar to *Uncle Roy [26]* in that it involved two types of player. Online players using a web browser were able to search a map to find the location of clues scattered around the city and then had to enlist a street player to visit the location physically to reveal the content of the clue.

3G communication technology was used in *I Like Frank* as it provides a reliable, always-on connection which was required in order for players on the street to be always contactable. Again, as with Hallberg et al's system, *I Like Frank* did not suffer from the reduced bandwidth 3G has when compared to 802.11 or Bluetooth as only textual information and map coordinates were transferred frequently and the information was not time-critical. It is important to note that all communication in *I Like Frank* was conducted via a central server and so was not peer-to-peer, although it may have appeared as such to users. Indeed, it is impossible for GSM/GPRS or 3G devices to "discover" one another without a central server – one of the primary reasons it is simply not suitable for, or intended for, peer-to-peer configurations.

Having examined the existing literature, it is clear that although the majority of theoretical systems seem to assert that Bluetooth is a prime choice for communications technology as it provides low power consumption, is built-in on virtually all mobile devices and has been designed for mobile communications, the fact remains that most implemented mobile, peer-to-peer systems rely on 802.11

rather than Bluetooth. This contradiction may leave a developer or designer of mobile systems unclear on which is the most suitable communications technology to employ in his or her system.

In order to identify the true performance of these technologies for peer-to-peer systems in mobile environments a study of peer discovery times, bandwidth, power consumption and reliability is conducted and described in Chapter 4. In addition, to overcome the problems in power consumption and reliable peer discovery, a wireless driver is designed and implemented and is also described in the same chapter.

2.3 Network topologies

Once a communication technology (e.g. 802.11 or GPRS) has been selected for use in a mobile device, there are often still many possibilities for the type of routing and routing algorithms to employ on that technology, in order to allow connections to peers or other devices. The selection of a network topology and routing methods can greatly affect the performance and user-experience of mobile systems. Whilst many mobile systems, such as that described in [42], simply assume network connectivity will be easily available, high bandwidth and uninterrupted, this is particularly short-sighted when it comes to the mobile environment.

The effects that choices on networks and network topologies may have is analysed as part of the design and testing process in the *Six in the City* system [118]. *Six in the City* is a game in which players must use PDAs, modified to look like guns, to work together in hunting down virtual monsters that exist digitally. Players wander around the physical environment but use their PDA screens to "see" the digital monsters in order to know where to aim their guns. Mitchell et al. identify that there are several possibilities and that the choices will affect the game [118]:

> *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*

Although the authors find both advantages and disadvantages in centralised and mirror-server architectures, they suggest that P2P architectures are the most suitable in a mobile environment [118]:

> *In summary, both client and mirrored server approaches are not well suited to mobile environments, as the network topology is too dynamic to efficiently predict effective locations for those servers. P2P approaches are far more applicable as they can be* dynamically reconfigured to suit the current topology.

Simply, mobile devices are unable to rely on infrastructure configurations, as a constant connection cannot be guaranteed. Mitchell et al. explicitly state this [118]:

> *...since state is stored on the server [in centralized and mirrored server architectures] clients have no support for continued operation upon disconnection from the network which may be fairly frequent in a wireless setting.*

It is apparent that mobile devices must either operate in an isolated way or make use of P2P connections if they are to provide continued functionality.

In the same year as the paper on *Six in the City* was published, Triantafillou et al. described *NanoPeers* – an architecture for peer-to-peer communications on lightweight devices [165]. As part of their research they draw attention to the fact that many aspects of designing for mobile devices, including the unique network constraints, are often ignored or overlooked [165]:

> *However, what researchers usually take for granted (i.e. average processing/storage/network capacities and power supply of modern computers) may not exist when we take a step further and deal with devices other than personal computers.*

Triantafillou et al. attempt to address this problem and, as part of their work, categorise peer-to-peer systems into three distinct groups—pure, centralised and hybrid—and apply them to the mobile environment. They define each as follows [165]:

> Pure *P2P systems are systems in which all peers are of the same stature and execute the same algorithm; no peer exposes any special functionality and the operation of the system is fully decentralized.*

> *In* Centralized *control systems, peers are of the same stature and execute the same algorithms, like in the Pure P2P case. However, specific operations are executed in a centralized manner (e.g. central authentication, indexing and searching server...)*

> Hybrid *P2P systems are a median between these two architectures. Peers are not all equal; a subset of the peers' population implementations is assigned special tasks. Selection of such peers is usually based on processing capabilities and available network bandwidth.*

These definitions are extremely useful to many of the systems in this thesis, as the majority of them have a peer-to-peer architecture. Although Triantafillou do define the three types, they do not advocate the use of any particular one over the others or state appropriate application types where each might be

used. Re-examining the Six in the City system and applying these categorisations, it is clear that Six in the City is a hybrid peer-to-peer system [118]:

> *Within each peer group, there exists a super-node which has the added responsibility of forwarding highly consistent events which effect state outside a peer group...*

It is also apparent that a centralised peer-to-peer system was considered and rejected for use with *Six in the City*. Again, although Mitchell et al. do give more detail about their reasoning for selecting hybrid peer-to-peer over centralised, they do not fully explore the area – nor do they consider or discuss the possibility of using pure peer-to-peer.

The concept of pure peer-to-peer predates the work of Triantifillou. Although Tveit does not use the terminology 'pure peer-to-peer' it is clear that his mobile recommendation system from 2001 does have a pure peer-to-peer architecture. He describes the system as follows [166]:

> *In this paper an approach for making a scalable recommendation system for mobile commerce using a Peer-to-Peer (P2P) [sic] is considered.*

It seems that the high scalability of this particular system is a result of the pure peer-to-peer architecture employed. There is no requirement for proximity to a central server and no need for devices to have a current connection for the application to provide its functionality. In essence, data in the system can spread between devices limitlessly.

A more recent system that has already been discussed in section 2.2.2, *DigiDress*, takes even greater advantage of the pure peer-to-peer architecture it employs [136]. *DigiDress* users are not only able to exchange the information that drives the application in a peer-to-peer manner but also the application itself. This ability for code or functionality to be spread through peer-to-peer encounters is novel and is enabled by peer-to-peer communication technologies such as Bluetooth. The authors of *DigiDress* view the ability as a great benefit in growing the community of users, stating that [136]:

> *DD application also allowed users to distribute the software to non-DD users via Bluetooth or infrared. This was thought to facilitate the uptake of the application which was critical to the success of DD.*

By the end of the short trial 36.7% new users had received the application from this method. It is clear that the transfer of functionality or code itself can aid in rapidly growing user-base – the size of which is crucial to many peer-to-peer systems. The success of *DigiDress* led to it being renamed *Nokia Sensor* and being used as a commercial product for Nokia [137].

Korteum et al. express strong opinions on the network and peer-to-peer topology a mobile system should employ, as well as agreeing with many of the other authors mentioned in this chapter on the state of current systems [100]:

> *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. make the bold claim that mobile ad hoc information systems are, by definition, fully decentralized, negating the use of hybrid and centralised peer-to-peer topologies. Whilst this is perhaps a little narrow, it does highlight 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 an ad hoc network topology in a pure peer-to-peer environment [100]:

- ***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.*

Whilst the decentralized architecture is powerful, as will be seen later in this thesis, it is not appropriate for every mobile scenario.

As the literature reveals, the type of peer-to-peer architecture (pure, hybrid or centralised) employed by a mobile system has a significant impact on both its usability and scalability. However, there has been little, if any, research into the types of architectures that are best suited to particular peer-to-peer applications or, indeed, peer-to-peer in general.

Kortuem et al. voice strong opinions that mobile systems *should* be pure peer-to-peer in topology but the use of "super-nodes", and hence hybrid systems, appears to have been necessary and of benefit in the *Six in the City* game. Certainly, the use of hybrid and decentralized systems cannot be easily dismissed.

The choice of peer-to-peer network type is discussed further in Chapter 3.

2.4 Positioning

Given that a fundamental characteristic of any mobile system is that it can be in any location at any time, it is hardly surprising that Mantoro and Johnson [113] believe awareness of a device's location to be *the* most vital element of context in a mobile system:

> *Location-awareness is the most important part of context-awareness for mobile computing systems.*

This is highly relevant to the work of this thesis which has as an aim of investigating how mobile, peer-to-peer systems can take greater advantage of the context information mobile devices provide.

LaMarca et al. concur that location is of core importance to a mobile device's context [105]:

> *Location has long been established as a key element of a mobile device's context.*

Similarly, Bahl and Padmanabhan [3] believe that location is a distinguishing feature of any context-aware mobile application:

> *A key distinguishing feature of such systems is that the application information and/or interface presented to the user is, in general, a function of his or her physical location.*

Jiang and Steenkiste seem to concur, claiming at the beginning of their paper on location models [93]:

> *Location information is critical contextual information for ubiquitous computing and mobile computing applications.*

Jiang and Steenkiste go on to categorise location models into two classes: hierarchical and coordinate, and claim that neither model is directly suited to context-aware mobile applications. For example, they claim purely hierarchical models are disadvantageous as they often rely on self-defined names, which can be misleading to other users or the same user at a later date, and do not allow accurate distance calculations to be made. They state that the only negative aspect of a coordinate model is that it hides hierarchical relationships and thus needs extra specification to enable spatial relationships to be deduced. Whilst the hybrid model they present may be useful in certain applications, their dismissal of coordinate systems seems harsh. Indeed, most hierarchical models must rely on coordinate positioning systems to drive them. Furthermore, there are many successful context-aware applications for mobile devices that do rely on only coordinate systems such as the Road Rager and GUIDE systems previously discussed.

[153] presents a study of the mobility levels of email users and discovers that 70% of users access their email from only one or two locations, which most often correspond to their home and office. However, they present this finding in a negative way claiming that this demonstrates users are less mobile than we may believe. The study does show that *at least* 30% of users check email at more than just their home and office and this figure may actually be higher. The research overlooks the fact that many users now check email using mobile devices. Such devices frequently use push email over GPRS or 3G and with this configuration the device is normally assigned a static IP regardless of its location. Thus, if one was accessing email this way it would appear to be from a single location despite the fact that one may actually use it in many locations.

It is clear, however, that regardless of the current mobility of users, the mobile market is rapidly growing, and both Mantoro and Johnson, and Jiang and Steenkiste believe sound location technologies to be of fundamental importance to any applications that will be context-aware in the mobile environment. Thus, an investigation of previous research in the use of location in mobile systems is appropriate.

Zeimpekis et al., realising that mobile devices are increasingly being used in new ways, and that a mobile device's wireless capability promises to 'enable the development of advanced mobile location services', investigate the currently available mobile indoor and outdoor positioning systems [174]. They state that if new mobile application types are to be enabled it must be noted that:

> *...a key factor that has been identified towards this perspective is the need for accurate knowledge of mobile terminal position.*

They go on to deliver a taxonomy of existing technologies, and an attempt is made to match specific categories of mobile positioning system to particular types of mobile applications. The positioning techniques are split into the two main categories of self-positioning and remote positioning. For example, self-positioning might be achieved through the use of a built-in GPS unit whilst remote-positioning might be achieved when a phone service provider uses its network of cell towers to locate the position of a handset on its network. It is clear that systems that fall under the remote positioning category of this taxonomy must rely on an external server or service to obtain a location for a mobile device. To subsequently deliver the location from the remote service to the device itself requires an infrastructure connection. As many mobile services do not have a constant connection to this kind of infrastructure or to the Internet, particularly those based primarily in peer-to-peer communities, it is felt that for the work in this thesis the self-positioning systems and techniques are most relevant. Thus, the remainder of this section examining the literature on positioning techniques leans more heavily towards self-positioning rather than remote-positioning techniques.

2.4.1 Early indoor location systems

One of the seminal location systems is undoubtedly the Active Badge system [169]. At the time, the closest system for locating people was the then standard pager system, which caused an audible beep

on the pager device and displayed a phone number for the receiver to call. Thus, it relied on action taken by both the requester and receiver to reveal a user's location successfully. It was also extremely slow and obviously not automated. The Active Badge system directly aimed to improve location finding by making this process automatic and constantly available within a building.

Users wore Active Badges which contained a small IR transmitter broadcasting a unique signal every 15 seconds. Receiver stations embedded throughout the building could detect these beacons and thus identify when an Active Badge was within 6 metres. Obviously, by modern standards accuracy was poor as a user's location could only be determined as being in range of a pre-installed receiver. Furthermore, as the badges only broadcast beacons every 15 seconds, it was possible for users to walk through a receiver's range without being detected at all. Each sensor station was wired to a central server that tracked and logged the users' locations. Users could then locate others by querying the server from their own workstations connected to the server through standard wired Ethernet, or by clicking on the badge's button to prompt the display of names and locations on a nearby video monitor [126].

Despite its limitations, the Active Badge system proved extremely popular with its users when it was set up at four sites in Cambridge. The setup was considerable, reportedly containing 100 badges and 200 sensors, and actively used over many years. The ability to locate others easily at any time was considered a great benefit, and the system has been referenced as the inspiration and source of nearly every location system since.

In 1997, Ward et al. recognised that none of the current position technologies was 'well suited to the task of generating fine-grain location information for use in context-aware computing' [171] and began creating a location system they hoped would be accurate to within 15cm. The system they created was called the ORL positioning system (after the Olivetti and Oracle Research Laboratory where the work was carried out) and used ultrasonic signals to locate devices. The detectable devices were small (10cm×6cm×2cm) transmitters and were detectable within a single room. The room had to be augmented with an array of receivers placed into the ceiling at 1.2m intervals. Every 200ms the transmitters would broadcast a beacon that would be subsequently detected by the receivers in the ceiling. By analysing the arrival times of the beacon signal at different receivers, distances from the transmitter to multiple receivers could be determined. The positions of the receivers and the distances from these to the transmitter could then be used to triangulate the position of the transmitter.

Whilst the ORL system achieved 95% accuracy at 14cm, it is clear that it requires a vast amount of equipment, calibration and commitment if it is to be used. The authors report that in their trials they required 16 ceiling units to cover a $75m^3$ area. Whilst the ORL system achieved new levels in accuracy, it could still not be used as the basis of a mobile system which aimed to be used in any location as the expensive setup costs meant it could only be deployed in extremely constrained areas.

The ORL system was later upgraded with smaller transmitters (5cm×3cm×2cm) and reused to explore the concept of *Follow-me* applications [85]. At this point the transmitters were also renamed *Active Bats* and the system is now more commonly referred to as the *Bat Location System*. One of the motivations for continuing the development and upgrade *ORL* to *Bats* was that the authors of *Follow-me* identify, as others subsequently have, that [85]:

> *Radio-based location techniques (e.g. GPS), which are successful in the wide area, are afflicted by severe multipath effects within buildings.*

It has long been clear, as is subsequently discussed, that GPS is not a good choice for indoor environments. However, it is conversely true, because of the setup requirements listed previously, that ultrasonic positioning systems such as *ORL* and *Bat* are poorly suited to outdoor environments.

Priyantha et al. built upon the work in *ORL* and *Bats* in an attempt to significantly reduce the deployment costs of ultrasonic positioning techniques. They realise the power of location to drive mobile context-aware applications, stating [139]:

> *A compelling set of applications enabled... are context-aware, location dependent ones, which adapt their behavior and user interface...*

Their Cricket system [139] is, unlike previous ultrasonic positioning systems, completely decentralized. The Cricket system puts the onus on users to install "beacons" in areas they control. For example, it would be the responsibility of the occupants of an office to install a beacon if they wanted the system to be available there. Beacons can be installed anywhere throughout a building and do not have to confirm to the strict 1.2m spacing the *Bat* system required. The *Cricket* system provides benefits at the cost of accuracy as it is typically accurate to within a few feet – far less accurate than *Bat*. However, *Cricket* remains one of the most appealing indoor positioning techniques due to its reduced setup costs, decentralised (and therefore more scalable) architecture and easier manageability.

2.4.2 GPS

One of the most readily available and widely used positioning systems for mobile devices in the outdoors environment is GPS. GPS, originally designed in 1978 for military use, was made available to civilians in 1996 when Bill Clinton declared it a US national asset and created an executive board to manage it as such. Since then, a plethora of commercial GPS units have become available and GPS functionality has recently started to be built-in in modern PDAs and cell phones (for example, in the HP iPaq hw6515 and the BenQ-Siemens SXG75). GPS has long been used with mobile applications – systems which rely on GPS include Cyberguide [1], GUIDE [36], Opportunity Knocks [133], FIASCO [32], WatchMe [115] and CampusAware [25]. A detailed description of GPS, its history and future plans for upgrading the network of GPS satellites is presented by Lammertsma in [106].

In 1997, the Cyberguide [1] system relied on GPS, augmented with an electronic compass to provide orientation information, to locate the tourists using the application. This system was used in three outdoor areas around a campus where it proved effective and positioning failures were not reported as an issue in any of the trials. However, GPS failed to work indoors as the authors found that 'GPS signals are weak or not available'. This is the most widely known problem with the GPS system and it is commonly believed that GPS is a good solution whilst outdoors but does not work when indoors. The authors summarise this belief when they write [1]:

> *GPS is unreliable indoors and the IR-based beacon system is impractical for us to implement outdoors.*

The GUIDE system demonstrated that GPS is not only unsuitable for indoor use but that GPS reliability outdoors is not always guaranteed [36]. GPS was seriously considered and tested as a positioning system for GUIDE but the authors finally decided not to use it for several reasons. The main problem was that GPS is often extremely unreliable in city environments. As the authors say: 'the position of tall buildings can prevent the GPS system from 'seeing' a sufficient number of satellites to obtain a fix on location.' This problem, as well as multipath GPS problems in which satellite signals are bounced off high buildings, can often make GPS a poor choice for any systems which are targeted for use in large cities. Indeed, this problem was experienced during the work on this thesis and is described in Chapter 3 (section 3.2.5.3).

A secondary problem is that GPS is not as yet commonly found in mobile devices and so external units must be used. This negative feature becomes more severe if standard GPS is found to be unreliable in the target location and must be supported by DGPS (Differential GPS). The GUIDE authors found that alternative positioning systems were more attractive as they worked 'without requiring bulky differential GPS equipment'.

Enge et al concisely describe many of the current problems with GPS—including obstructions, indoor use and multipath. The quote that follows is rather lengthy, but is necessary as multipath problems are indeed common and are experienced in systems described later in the thesis. Therefore, it is prudent to explain fully now so that the problem is understood subsequently. In the quote, SNR refers to Signal-to-noise Ratio, which is a measure of the ratio between the signal containing the information that is actually desired to be transmitted and the background noise or interference [60]:

> *Under foliage, many satellites have SNRs below 25 dB-Hz even when they are 40 degrees above the horizon. Inside a hotel, the SNRs really suffer... The urban and indoor challenge is compounded by multipath. As the name implies, the signal from the satellite has followed multiple paths to the receiver. In addition to the direct path, the signal has arrived after one or more reflections. In open environments, the reflected signals are almost always weaker than the direct signals, but this is not always the case in cities and*

> *indoors. Reflections from buildings and other structures are commonplace, and this multipath can have any of the three undesired effects. The reflected ray may destructively interfere with the direct ray and* fade *the composite signal power. The reflected ray may have approximately the same power as the direct ray and distort the correlation peak used by the receiver to make the GPS measurements. The reflected ray may be much stronger than the direct ray and cause the receiver to assume that the reflected ray is the direct ray.*

Multipath is indeed an inherent problem with GPS and there is no completely effective solution. Whilst, Enge et al. propose novel algorithms for further filtering GPS data to increase the sensitivity of receivers, they openly state that the algorithms have yet to be formally assessed and that there is a great deal of work yet to be done before GPS will be reliable in urban environments.

Rakkolainen et al. propose a system in which a 3-dimensional model of a city can be used to predict where multipath may occur and to attempt to correct for the errors it may introduce, mainly by using the model to work out the height of the receiver [141]. Again, whilst such a technique can reduce the errors inherent in GPS due to multipath it cannot eliminate them completely. Furthermore, the technique requires a considerable amount of storage for the model of the city, or alternatively a wide connection to a server to stream the model from and is thus not generally suitable for mobile devices. It is clear that despite attempts at improving GPS, GPS alone cannot yet offer a comprehensive positioning solution for outdoor environments.

Whilst future upgrades to GPS, such as Galileo and GNSS [106], are likely to improve accuracy and reduce signal interference by the end of the decade, it seems clear that GPS will continue to suffer from signal loss and dropout in built-up or indoor locations.

2.4.3 802.11

The first use of 802.11 wireless as a positioning system may be that of the RADAR system [3]. By sampling the strength of the signals to the 802.11 base stations and calculating the mean, standard deviation and median, the RADAR system is later able to use that information to triangulate a position based on the strengths to multiple base stations. One important point that the authors discovered was that the orientation of a mobile user can greatly affect the signal strengths recorded to access points. Variation of up to 5dBm is possible with the user standing on the same spot depending on whether or not their body mass (containing mostly water which is particularly obstructive to wireless signals) is between the access point and the mobile device. RADAR successfully managed to track both stationary and mobile users indoors with an accuracy of approximately 2-3 metres.

In the same year work on RADAR was published, Small et al. published similar work. They describe two positioning techniques they trialled at Carnegie Mellon University [155]:

> *The first approach consists of discovering the active access point for a mobile client and mapping that information onto a two-dimensional campus map. An access point covers a sphere of approximately 75 feet in diameter. The second approach improves resolution by triangulation based on measured signal strength from several nearby nodes.*

Just as Bahl et al. discovered earlier, Small et al. found the same 5dBm variation in sampled signal strengths but also found that accuracy fell as distance from the access point grew. This leads them to suggest that [155]:

> *... it may be advantageous to place a higher degree of confidence in stronger signals and weight them accordingly.*

As will be clear later in the thesis, this suggestion is an appropriate one and is built in to the positioning system, *Navizon*, developed as part of the thesis.

Small et al. describe a similar problem in their work as the RADAR system exhibits, that of requiring an extremely large number of samples to determine adequately the location of the nodes [155]:

> *To cover an area of one acre at one sample every 10 feet, we would require 441 samples. If we were to generate a table to determine a user's location in a large area such as Carnegie Mellon's 103 acres, we would require 44,100 samples. This introduces two problems. The first is the problem of taking these samples. This averages to approximately 110 samples per access point. The second of these problems is searching through all of the samples to determine the location.*

The requirement for such a large number of samples to be collected before the system can be used introduces substantial set up cost in using systems such as RADAR and that described by Small et al. and makes them unattractive for mobile systems. Furthermore, the search through such a large sample set can subsequently be time consuming if not efficiently implemented, particularly on a mobile device where processing power is often low.

High set up cost is also seen in commercially available 802.11 positioning technologies. InStory [40] is a system that provides interactive cinematographic narratives and entertainment related activities to mobile devices. As part of this process involved 'real persons and virtual characters that perform actions and also move in physical and virtual spaces' and as the trials were built to run in a council owned palace in Sintra, Portugal, a position system that worked indoors was required to support the system. The authors elected to use the commercial Ekahau software which is a position system based on 802.11 wireless. They state that [40]:

> *Mobile networks combined with positioning techniques provide a new channel for a radically new form of cinematographic narratives that are navigable in space.*

Although the authors do not state so themselves, Ekahau requires a lengthy calibration period during which the calibrator must repeatedly scan for access points, rotate 90 degrees, scan again until a full circle is completed and then move forward one metre and start the process again. This must be done for each square metre that is to be covered by the positioning system. It is clear that the Ekahau system is a direct descendent of RADAR, as the requirement of the same sampling in 90 degree orientations is discussed in earlier RADAR work [3]:

> *In one orientation, the mobile host's antenna may have line-of-sight (LoS) connectivity to a base station's antenna while in the opposite orientation, the user's body may form an obstruction.*

Furthermore, if the network setup is changed at all, such as a single access point being removed or added, the process must be repeated from scratch for the entire area. If Ekahau is calibrated correctly in an area with a sufficiently high density of access points it can provide positioning accuracy to within a few metres.

Thus, although 802.11 can be used to triangulate a position in this way, it requires an extremely lengthy and costly setup process. The fact that 802.11 positioning of this type is not available unless an area is calibrated means that it cannot provide a global, generic positioning system if used in this way.

After rejecting GPS for the reasons detailed previously, the authors of GUIDE also elected to utilise 802.11 positioning but in a different manner. Instead of attempting to triangulate a position, they simply coupled each unique infrastructure access point MAC address to a name rather than geographic coordinates. This equates to the selection of a discrete model over a continuous coordinate one, as described in [93]. This proved extremely effective within GUIDE as, although it was not as accurate, GUIDE did not require information to be delivered at fine-grained locations. Furthermore, the location was guaranteed to be known whenever the system was connected to an access point.

A similar setup to that used in GUIDE was employed in ActiveCampus, specifically the ActiveCampus Explorer [77]:

> *ActiveCampus currently detects location through the PDA's report of currently sensed 802.11b access points. Their reported signal strengths and known locations are used to infer the user's location by a least-squares fit.*

The location services available through this nearest-beacon positioning, which results in reduced accuracy when compared to methods used in RADAR or Ekahau, are a subset of those available in

more accurate systems. Such systems are identified by Sood et al. as *Low-fidelity location based information systems* [157] – these are discussed further in section 2.4.5.

When using 802.11 for positioning it is apparent that previous systems either suffer from an extremely high set up cost, such as in *RADAR* and *Ekahau*, or suffer from substantially reduced accuracy, as in *GUIDE* and *ActiveCampus*.

2.4.4 Bluetooth

Madhavapeddy and Tse conducted an in-depth study into the viability of using Bluetooth for a position system [111]. In their trials they rely on the Active Bat system [85] previously discussed as a 'location oracle' to compare their Bluetooth position techniques to. Early on in the paper the authors correctly draw attention to one of the main disadvantages of using 802.11 for positioning [111]:

> *Bluetooth is especially important due to its ubiquitous and "always-on" presence... in contrast to the more power-hungry WiFi, which is generally only switched on in stationary devices.*

However, their methodology section is quick to point out the difficulties that exist in the Bluetooth protocol [111]:

> *[Bluetooth] chipsets make no guarantees about the accuracy of the magnitude. Therefore the only source of signal strength information we can rely on for Bluetooth devices is the link quality.*

The problem is compounded by the fact that the link quality reading is only reported as an average over the length of the Bluetooth scan rate which is, by default, ten seconds on most devices. Thus, raw RSSI strengths are not available and the average reading is updated far too infrequently to be of much use in discovering distances to access points to use in position triangulation.

This is later reinforced when the authors state [111]:

> *Compared to WiFi, none of the reported link quality values give high enough accuracy and dependability to enable location-sensing based on signal strength alone.*

This problem of not having direct access to accurate, timely signal strength information is perhaps the largest obstacle to using Bluetooth for positioning as it results in any form of triangulation being based on unreliable sampling.

Despite conducting their experiments in a quiet building after normal working hours, and taking care to ensure there was always a clear line-of-sight between the Bluetooth device and the Bluetooth access points being used to triangulate positions, one of Madhavapeddy and Tse's early discoveries is that the

BER (Bit Error Rate) between two Bluetooth devices is too high for communications to be of use when one of the devices is being moved at a brisk walking pace [111]:

> *...users walking around a building at a normal walking pace would have an adverse impact on their link quality and available bandwidth. This is of concern to location-based services deployed in public buildings such as shopping malls – in order to push high-bandwidth content reliably, the user would have to be relatively stationary rather than walking through the Bluetooth zone.*

This is a crucial discovery as it immediately negates Bluetooth positioning for use in any form of transport other than a slow walking pace. Indeed, after this early finding the rest of Madhavapeddy and Tse's research for the paper was gathered from devices being moved at below average walking pace or being held completely stationary.

Madhavapeddy and Tse also examine the useable range of Bluetooth communications, finding [111]:

> *...the Bluetooth transmitter has a practical range of around 2 rooms, after which the BER exceeds 2% and the bandwidth on the connection drops below useful levels.*

Compared to 802.11 this range seems extremely low and thus means that if Bluetooth position triangulation were possible it would require an extremely high-density of nodes to operate efficiently. In any event, the authors also point out that most Bluetooth devices can only detect and connect to one other device at a time and thus triangulation from multiple nodes is simply not feasible.

Madhavapeddy and Tse are quite clear that Bluetooth, in its current form, is not viable as a means of locating mobile users, summing up by giving a list of enhancements to the Bluetooth protocol that would be required before it were [111]:

> *(i) expose a fine-grained RSSI dBm value via HCI (similar to 802.11); (ii) accept multiple simultaneous Bluetooth connections in order to assist triangulation; and (iii) update RSSI values per-packet without a significant time lag.*

The results Madhavapeddy and Tse gather are in many ways relevant to the use of Bluetooth as a general mobile, peer-to-peer communications system and concur with many of the findings from other literature in section 2.2.2. The use of Bluetooth is further investigated in this thesis in Chapter 4 (section 4.1.2).

2.4.5 Mobile Phone Cell

As with Bluetooth, positioning that relies on mobile phone cells has not yet become widely popular and it is difficult to find any research systems that employ it as the sole technology to drive a positioning system in the literature. Whilst there are commercial systems that use mobile phone cell information to

locate a mobile phone[2], they are not widely used due to the fact that they are often expensive and extremely inaccurate. Certainly, they are currently far less accurate than the systems discussed in this section. Furthermore, virtually all of the commercial systems require that one user must actively request the location of another, and the location is normally returned in a proprietary format in an SMS message. Perhaps most importantly, all these services rely on information delivered from the operator—resulting in an external dependency on the operator's servers and resulting in a system in which the end user has little or no control over the privacy of their location. These problems negate the use of these commercial services to provide a *continual and automatically delivered* location, which is often required to drive a location-aware application.

Despite mobile phone cell positioning being a relatively new area and not implemented into many research systems, Trevisani and Vitaletti do provide an investigation into the limits and benefits of positioning using cell-ID [164]. Trevisani and Vitaletti examine cell-ID performance in urban, suburban and highway areas in both the U.S.A. (New York) and Italy (Rome). At the beginning of the paper they point out [164]:

> *Studying Cell-ID's performance by simulation requires operators to provide information on network planning. This data is not made public by operators. ... We remark that our experiments do not try to be complete...*

This is perhaps one of the largest inhibitors to research in the area and, as the authors themselves remark, it has hampered their own work. Another inhibitor, not directly stated in their work but implicit throughout, is that the phone developers, possibly at the behest of the network operators, do not provide APIs or expose the functionality to allow other developers to gain access to the full information about what cell towers a phone can currently detect. Most mobile phones continually scan and analyse the signal strength to six cell towers in order to make decisions about which is the most appropriate to use. However, there is currently no phone on the market for which there is a generally available API which exposes any more than the cell-ID of the single cell-tower the phone is connected to. Thus, in Trevisani and Vitaletti's research, they concentrate only on identifying which is the currently connected cell-ID in order to locate a user within a particular cell.

After collecting over 6000 samples of cell-ID and location (tracked using a GPS unit), Trevisani and Vitalitti discover that the average distances in metres from the mobile device to the nearest cell towers in Italy and the U.S.A. are as follows:

	URBAN	SUBURBAN	HIGHWAY
ITALY	480	750	1000
U.S.A.	790	490	2910

[2] Such as http://www.traceamobile.com/, http://www.followus.co.uk/, and http://www.mapamobile.com/.

Higher densities of cell towers are, unsurprisingly, generally placed by operators in dense population areas, thus the Italy results demonstrate the expected fall in average distance from urban to suburban and to highway. According to the authors, the discrepancy in the U.S.A. results, where the suburban average distance to the nearest cell tower is lower than the urban, may be due to several reasons. The main two are that in the New York area the high-density of large buildings often obscure the signal to the nearest cell-tower and so cell-phones may spend longer periods of time connected to more distant towers and that suburban cells in the U.S.A. are more closely packed at high-population areas rather than spread more uniformly as in Europe.

Whilst the authors are in no doubt that the 500m and 800m levels of positioning accuracy gained from identifying which is the nearest cell in U.S.A. and Italy respectively are not suitable for the use of all mobile applications in general, they do identify a sub-group where it may be convenient to rely on cell-sized accuracy [164]:

> *...in our opinion there is a set of significant services, the so called* resource discovery services, *for which Cell-ID's accuracy might be sufficient. Resource discovery services answer questions like:* "Which Chinese restaurants are in my vicinity?"

As the size of a phone cell is rarely greater than 2km, any services found in the cell are highly likely to be within walking distance.

Sood et al. further discuss how this type of inaccurate location information may still be of value in some application areas [157]:

> *In many cases, we are able to use cell information to obtain longitude and latitude information down to a one block radius. Our efforts in the area of low fidelity (Low-Fi) location information is aimed at using this less than perfect data to still provide users with services that are more powerful, effective, and easy to use than those that lack any location information at all.*

Sood et al. describe a phone application called *StickyNotes* in which users are able to leave notes about the location they are currently at. As the position granularity is approximately the size of a city block the relatively large imprecision can, in some situations, actually prove beneficial in allowing the system to search quickly and identify all the markers available within this short walking distance.

The most accurate phone cell location technique in published work is likely that of Otsason et al. [129]. They begin by listing a number of reasons they believe GSM positioning is advantageous over other forms of positioning, particularly for use indoors where a location from GPS is normally not available [129]:

> *GSM-based indoor localization has several benefits: (i) GSM coverage is all but pervasive, far outreaching the coverage of 802.11 networks; (ii) the wide acceptance of cellular phones makes them ideal conduits for the delivery of ubiquitous computing applications. A localization system based on cellular signals, such as GSM, leverages the phone's existing hardware and removes the need for additional radio interfaces; (iii) because cellular towers are dispersed across the covered area, a cellular-based localization system would still work in situations where a building's electrical infrastructure has failed. Moreover, cellular systems are designed to tolerate power failures. ...(iv) GSM, unlike 802.11 networks, operates in a licensed band, and therefore does not suffer from interference from nearby devices transmitting on the same frequency (e.g., microwaves, cordless phones); and (v) the significant expense and complexity of cellular base stations results in a network that evolves slowly and is only reconfigured infrequently. While this lack of flexibility (and high configuration cost) is certainly a drawback for the cellular system operator, it results in a stable environment that allows the localization system to operate for a long period before having to be recalibrated.*

Otsason et al. achieve accuracy ranging between 20 to just under 5 metres depending on the density of buildings in the area (the buildings in downtown areas create more reflections and interference than suburban areas). However, the system they implement does not work on standard phones and they are instead forced to use specialised hardware rather than standard phones in order to sample the GSM beacon data they require to drive their system [129]:

> *We collected 802.11 and GSM fingerprints using a laptop running Windows XP.*
>
> *...We collected GSM fingerprints using a Sony/Ericsson GM28 GSM modem, which operates as an ordinary GSM cell phone, but exports a richer programming interface.*
>
> *...To collect the measurements, we placed the laptop on an office chair and moved the chair around the building.*

Furthermore, Otsason et al. rely on a specific Sony/Ericsson API and thus the technique only works with their proprietary hardware. Clearly, Otsason et al's system is not yet ready for use in small mobile devices such as PDAs or phones in general. However, they do clearly demonstrate that a much higher accuracy, improved from earlier work, is indeed possible using GSM technology.

Despite the high accuracy achieved, Otsason et al. realise that GSM is not accurate enough in all situations and briefly discuss a technique for using a combination of GSM and 802.11 to achieve higher accuracy and availability. Combinations of positioning technologies of this nature are discussed in the next section.

The phone positioning techniques discussed in this section so far have relied primarily on triangulation of a phone's position based on information on detected signal strengths and cell identification numbers, using centroid algorithms to achieve this. Chen et al. report on an alternative method known as 'fingerprinting' [34]. This technique assumes that detected cell identification numbers and signal strengths at a given location are stable over time. By recording this information a single time, along with a text label to name the location, the device can later be identified as being at that location whenever the cell identification numbers and signal strengths match in the future. Thus, the fingerprinting technique constructs a record of text labels matched to cell identification numbers and signal strengths, and by searching this can ascertain if the current information matches any of the stored locations. Chen et al. find that fingerprinting techniques provide an accuracy of 94-313 metres, depending on the density of cell towers in the area and whether information from cell towers owned by providers other than the one the phone is contracted to can be used.

Fingerprinting techniques are employed in the Whereabouts Clock [151]. The Whereabouts Clock aims to display the current status of all the members of a family through a clock-shaped display mounted in an often-used room within the family's home. Information about the current location of each member of the family is determined through fingerprinting techniques used on the mobile phones owned by each family member. Fingerprinting is applicable to such an application as the vague information about location that it provides through simple text labels—such as "Home", "School" or "Work"—is more desirable to display to family members than map coordinates. However, such vague information is not generally applicable to all mobile applications. Whereas a text label can be assigned to a region if map coordinates are known, map coordinates cannot be determined from the information fingerprinting algorithms store. Therefore, whilst the Whereabouts Clock demonstrates that fingerprinting techniques are suitable in some mobile situations, a coordinate-based system is more generally desirable, as it can be used to drive a larger number of location-aware, mobile applications—and can be augmented with text labels similar to that provided by fingerprinting if required.

As will be shown later in Chapter 5, accurate GSM positioning of the nature Otsason et al. described, which primarily calculates coordinate-based locations, is implemented within a system called *Naivon* as part of this thesis.

2.4.6 Combinations

Whilst it is obvious that there are a large number of positioning technologies, it is also clear that no single one manages to provide mobile applications with an accurate, always-available location service. Furthermore, even those developed solely for outdoor use, such as GPS, do not manage to provide a location in every outdoors scenario (for example, as has been discussed, GPS performs poorly within cities).

Reviewing the existing literature in 2002, Baus et al. realise that [8]:

> *None of the discussed systems allows for a seamless switching from indoor (passive location sensitivity) to outdoor (active location sensitivity)...*

This leads them to create one of the first, if not the first, system that works both indoors and outdoors by combining GPS and IR positioning into a single system [8] which seamlessly switches between the outdoor ARREAL and indoor IRREAL systems without requiring interaction from the user. A more advanced system that follows an almost identical concept is later presented by Kruger et al. [101]. Their system again uses GPS outdoors and infrared positioning indoors and, unsurprisingly, their system suffers the same accuracy and high-cost problems encountered by Baus et al.'s earlier work.

In 2004, Benford et al. describe the problem with existing positioning technologies as follows [13]:

> *...there is currently no universal tracking system that can provide reliable, accurate and extensive coverage across a city with the result that game developers and players have to cope with considerable uncertainty with regard to location.*

The fact that reliable and accurate positioning is unavailable does have clear implications for the types of mobile systems that can be developed and the experiences can be provided to users. The fact that current positioning technology cannot be relied upon leads Benford et al. to create a system that relies purely on the user's own direct input. Instead of relying on an external device or hardware to locate a user, the user's position is inferred as the centre of any map they are currently viewing or explicitly marked by the user through clicking on the map. Surprisingly, this works extremely well in a game Benford et al. describe, Uncle Roy is All Around You. The fact that self-reported positioning in this way proves better or equal to existing positioning technologies does not necessarily demonstrate that self-reported positions are a successful strategy. Rather, it further emphasises that existing positioning technologies are simply not viable solutions in mobile systems. Indeed, Benford et al. make clear the problems in self-reported positioning [13]:

> *Two potential limitations of self-reported positioning are that the mobile player has to know where they are and/or where they are heading, and that they may cheat, that is deliberately choose to lie about their position.*
>
> *...A further issue for self-reported positioning is that it demands the constant engagement of the user in order to maintain an up to date position, and even then remote users may be frustrated at the low frequencies of updates. While this may be acceptable for tasks that are highly fore-grounded – such as playing an absorbing game – it may be less suited to more background tasks, for example where a context aware system spontaneously interrupts the user.*

It is apparent that when a system with these severe problems outperforms or equals existing technologies, the existing technologies require substantial improvement. A potential solution is the combination of several existing technologies in order to reduce or negate the problems of any one system.

It was not until 2005 that concentrated effort began in combining many positioning technologies into one system in order to provide a more comprehensive solution. Virtually all of the published work in this area has been as a result of the *Place Lab* initiative whose authors echo the problem stated by Baus et al. earlier [104]:

> *...current location systems do not work where people spend most of their time: coverage in current systems is either constrained to outdoor or environments or limited to a particular building or campus with installed sensing infrastructure.*

They also describe a secondary problem in that:

> *...existing location technologies have a high cost of entry to both users and application developers. Many location systems require expensive infrastructure, time-consuming calibration, or special tags, beacons, and sensors.*

To overcome both the problems that no single positioning system provides a comprehensive solution and that many have a high cost of entry, *Place Lab* aims to combine several existing positioning technologies in the hope of greatly increasing the percent of time location fixes are available whilst also reducing the cost of entry. *Place Lab*'s goals are most succinctly asserted in the *Place Lab* home page's introductory text [138]:

> *Place Lab is software providing low-cost, easy-to-use device positioning for location-enhanced computing applications. Place Lab tries to provide positioning which works worldwide, both indoors and out (unlike GPS which only works well outside). Place Lab clients can determine their location privately without constant interaction with a central service (unlike badge tracking or mobile phone location services where the service owns your location information).*

LaMarca et al.'s research shows that for three types of user (an immunologist, home-maker and retail clerk) location fixes in *Place Lab* using GPS are available for only 4.5% of their day, GSM for 99.6% and 802.11 for 94.5% [104]. However, it is found that by fusing the three technologies a location fix is available for 100% of the day.

Interestingly, measurements with Bluetooth positioning were trialled but omitted from the research results [104]:

> *...non-mobile Bluetooth devices have not reached sufficient density where they are eminently useful for beacon-based location estimation in the wild. ...Although our results do not report Bluetooth beacon densities, we did scan for them during our data collection and we saw virtually no fixed Bluetooth in any of our test locales. The sparseness of Bluetooth beacons is further exacerbated by the fact the each scan for nearby Bluetooth beacons takes approximately 10 seconds to complete.*

The initial statements here provide further reasons why Bluetooth is generally a poor choice for positioning techniques whilst the last, referring to the long scan time, echoes findings revealed earlier in the literature review. LaMarca et al. go on to provide their view on the future of mobile context-aware systems, which emphasises the need for a more reliable and available positioning system [104]:

> *We believe that many emerging location-aware computing [sic] application are going to require 100% availability of location information in real people's lives, similar to the way cellular phones are held to a 100% availability standard.*

Despite *Place Lab*'s success as a fusion of multiple location technologies, one of the main problems it has is that the location of the nodes it relies on to drive its 802.11, GSM and Bluetooth positioning must be pre-sampled in an area before any of these forms of positioning is available. In another paper from the *Place Lab* authors, LaMarca et al. themselves describe the problem as severe [105]:

> *Both the initial mapping and subsequent re-mappings are time consuming and represent the biggest cost in deploying and maintaining the system.*

They attempt to generate a solution in the form of 'self-mapping' that allows new node positions to be automatically calculated from only a few initially known nodes. When a new node is detected a scan for known nodes may result in successful hits. If this is the case then the newly discovered node's location can be estimated by analysing the signal strengths to the known nodes and attempting to triangulate the mobile device's current, and therefore possibly the newly detected node's, location. The authors' tests demonstrate that, from a 10% seed set of beacons, self-mapping methods can build up coverage to between 87-90% of an area. They compare this to war driving, which involves the detection of access points in the area by a person, or people, systemically driving through as much of the area as possible whilst carrying a PDA or laptop with a wireless card capable of detecting 802.11 networks in order to discover the location of access points in the area. The authors also make the claim [105]:

> *With as little as 50% of the beacon locations, self-mapping can produce a radio map that estimates a user's location as well as a war driving database. Further, we showed that when new beacons are introduced, self-mapping estimates their positions nearly as accurately as war driving.*

Kim et al. conduct a study of the accuracy and reliability of war-driving techniques which are relied upon to gain the node locations, or seed node locations, to drive the systems described. They describe the need for such information [97]:

> *...researchers have started using 802.11 beacon frames from access points (APs) to locate wireless network users. Intel's Place Lab provides software that can track users both indoors and outdoors. Skyhook Wireless provides a similar commercial solution for locating Wi-Fi users. These approaches require knowledge of the (actual or estimated) location of APs. In addition to user-location tracking, researchers also use the location of APs to analyze wireless network characteristics such as the coverage range of APs or interference among APs.*

Whilst not directly relevant to this particular section, it should be noted that the technology which drives the Skyhook Wireless system referred to in the quote directly stems from work carried out for this thesis and was developed by myself and Malcolm Hall as part of Skyhook's *Bertha* application.

Kim et al. discover that the method of transport used whilst war-driving can greatly affect the quality of the results, finding that war-driving an area detected 38% of the previously known access points with a median error of 40.8m whilst war-walking detected 59% with a median error of 31.6m. They state of both war-driving and war-walking [97]:

> *We observed that estimated AP locations are often biased towards the war-driving paths, which makes the maximum signal range of APs to appear shorter and the interference among APs to appear more severe than in reality.*

This is a strong caution against using normal war-driving techniques and placing discovered nodes at the exact location they are discovered. This problem is addressed later by the work in this thesis and a possible improvement is discussed in Chapter 5.

As mentioned, some of the work referred to was conducted as part of this thesis. The work in this thesis resulted in similar position combinations and was built into the now commercially available product named *Navizon*[3]. Although similar to *Place Lab* in many ways, the research and work for the positioning technology Navizon utilises was conducted and implemented completely independently of the *Place Lab* software. Indeed, during the research, design and implementation stages of the

[3] www.navizon.com

technology both Malcolm Hall and myself were not aware of the work by *Place Lab*. The first use of our technology in a system outside our own, Skyhook's *Bertha*[4] application, was in early 2004 whilst the majority of *Place Lab* research was published in 2005.

The positioning technology we developed, which now drives the Navizon system, is discussed in far more detail in Chapter 5.

2.4.7 Identifying important locations

A good positioning system can locate mobile devices and provide the location to end-users but this information, such as latitude and longitude coordinates, is often not enough to prove useful to the user. Marmasse and Schmandt sum up the problem [114]:

> *...coordinates must be translated into positions that are relevant to the user, and these obviously vary greatly from person to person. Users neither know nor care about such coordinates; rather they identify "home", "work", "school", "post office" etc. Although a map could be used to specify such points, why should users spend valuable time filling out detailed property lists for a system which* has yet to prove its value to them?

They describe a system they implemented called comMotion in which the system attempts to discover locations that may be of importance to the user automatically. The system relies on GPS for its raw location data and uses a simple algorithm to identify if locations are salient. If the GPS signal is lost it is assumed that this is because the user has entered a building. If this occurs three times at approximately the same location then the system identifies the location as possibly important. It is hoped that over a short period of time this method will be able to identify the majority of buildings that are of importance to the user in their daily lives. Users are notified when a location is identified as important and can enter their own label for the location immediately or wait until a more convenient time and enter it by selecting it from a list of recently identified locations. Whilst this technique is simple, it proves extremely effective and has been implemented into other systems such as that described in [160].

Hightower et al. build on the work in comMotion – stating almost the exact same motivation for their work [88]:

> *Many emerging location-enhanced applications, however, want colloquial place names like "Home," "Work," "Movie Theater," or "Tony's Pizzeria" instead of latitude and longitude coordinates.*

[4] *Bertha* is an application that Malcolm Hall and myself developed for Skyhook Wireless. The application was designed to be used inside vehicles as they travel around, in order to map out the locations of access points they pass. The application continually scans for 802.11 access points, attempts to work out their approximate location, and uploads all recently scanned access points to a central server whenever an access point Skyhook Wireless are permitted to use is in range. *Bertha* is still used by Skyhook Wireless, who pay delivery drivers to carry the application with them during their normal working day.

Hightower et al. implement an infrastructure, BeaconPrint, which identifies salient places by continually monitoring the 802.11 and GSM beacons currently in range, which many of these positioning systems already do. If a preset amount of time passes with the set of beacons remaining the same, or almost the same, then the location is marked as being of possible importance. Hightower et al. claim a 90% success rate in identifying locations of importance to users when employing this technique. One of the most significant improvements of this technique over the one used by comMotion is its success rate in identifying locations visited infrequently. While comMotion required a minimum of 3 visits to the same location before being capable of identifying it as salient, Hightower et al. claim [88]:

> *BeaconPrint patches this deficiency by demonstrating an accuracy rate of over 63% even for places someone returns to only once or visits for less than 10 minutes, increasing to 80% accuracy for places visited twice.*

Both these techniques prove useful, and slight variations have been built into a few of the systems later described in this thesis.

Thus, it is clear from the review of literature that accurate and always available positioning is a fundamental requirement for mobile, context-aware systems. It is also apparent that although there are extremely good solutions for either indoor or outdoor positioning, there exists no single positioning technique which can provide the required availability to drive such systems.

The work carried out on the *Place Lab* project begins to address this problem but, as described later in Chapter 5 (section 5.2), still has several deficiencies that hinder its widespread adoption and use in the mobile environment. Therefore, this thesis will explore the issue of providing accurate and highly available positioning for mobile applications. As will be shown, this research leads to the creation of the Navizon system described in Chapter 5.

2.5 Recommendations and adaptation

Recommendation, or collaborative filtering, techniques can be employed to select the most relevant subset of data to present to a user or system in order to provide a focus and improve efficiency [144], [72], [65], [167], [152]. Such techniques seem appropriate for use in mobile environments where, as has been shown, there is a greatly increased level of context information and thus a greater density of data in general. In a mobile environment new data may continually be arriving from sensors, peers and servers and without filtering techniques this data can quickly overload a user or application. As early as 1994, Schilit et al. proposed adaptive techniques based on context information for mobile devices [149]:

> *Reconfiguration is the process of adding new components, removing existing components, or altering the connections between components... In the case of context-*

> *aware systems, the interesting aspect is how context of use might bring about different system configurations and what these [sic] adaptions are.*

Whilst Schilit et al. recognise that, as previously discussed, location is likely the most important item of contextual information for mobile devices, other contextual information is still often vital [149]:

> *Reconfiguration could be based on other information in addition to location, for example, the people present in a room.*

As has been discussed previously, the level of adaptation required for mobile systems to behave appropriately may be considerably higher than that of desktop systems as mobile systems experience a wider range of contexts. It seems suitable, therefore, that a system for filtering data or employing it to drive adaptation on mobile devices should be primarily based around the context information available.

Many previous mobile systems have attempted to filter or recommend data in differing methods. In Tveit's work on recommendations in the mobile environment, discussed earlier in section 2.3, he theorises how a pure peer-to-peer recommendation system could be constructed but fails to implement it and is aware that there are many unanswered questions [166]:

> *How to deal with fraudulent behaviour? How to maintain consistent and unique identification of products and services in the voting vectors used in the queries? How to keep cache consistency? How to deal with privacy and encryption of queries and the results? Where are the bottlenecks for true scalability? Is a true P2P-based system [sic] for recommendations or is a hybrid approach like Napster or Gnutella with Gnutellahosts needed?*

Some of these questions relate back to issues already discussed. For example the last question clearly refers to the problem of deciding which type of peer-to-peer architecture to use (pure, hybrid or centralised). However, it is obvious from this list of unanswered questions that there are many issues yet to be addressed in mobile recommendation systems.

The MobiTip system, described earlier in section 2.2.2, aims to provide recommendations to users about interesting locations such as restaurants, shops, theatres or sporting venues. MobiTip was originally planned as a pure peer-to-peer system but was augmented with specialised statically placed caching nodes, as in the trials not enough users were encountering one another and thus there was a low turnover of new recommendations arriving on users' devices. In MobiTip 'tips' are exchanged between peers. The authors identify a particularly powerful feature that mobile recommendation applications can take advantage of [146]:

> *If the user is in a shopping mall, tips will be influenced by others in the mall. In a different setting – such as a conference – MobiTip will produce a different set of tips, based on another group of people.*

It is to be expected that users co-present in a location are likely to share similar interests. Thus, mobile recommendation systems can take advantage of this information simply by realising that information delivered from co-present peers is more likely to be of interest than information delivered second-hand or from a statically placed infrastructure server.

The idea that location alone can be used to activate recommendations is common in existing systems. For example, Newcomb et al. describe a mobile shopping assistant which recommends items of interest when the user is proximate to them [121]. In the system Newcomb et al. describe, items which are on sale and which the user has previously purchased are brought to the attention of the shopper through the use of audio notifications. Systems in which the only trigger for information is location have been referred to as 'walk-up, pop-up' (a term introduced by Brown and Chalmers [19], [31]) and are common in the literature [161], [102]. Whilst the form of recommendation employed by Newcomb et al. uses some background information about the user's history, it cannot be said to be purely walk up, pop up and thus is perhaps more a form of mobile recommendation. However, this system and the many which do rely on just walk-up, pop-up techniques highlight the power of location alone in the mobile environment. This is, of course, reinforced by many of the statements in section 2.4. Thus, it is apparent that in pure peer-to-peer recommendation systems location, and in particular its proximity to other devices, may become one of the greatest weighting factors to the recommendations delivered. However, this should be viewed as a benefit rather than a problem as the large crossover in proximate users further focuses the recommendation data.

Whilst recommendations seem an appropriate technology in the mobile environment, few systems have managed to implement them successfully and even fewer have successfully managed to utilise successful desktop recommendation techniques in the mobile environment. For example, the *MovieLens Unplugged* system [117] attempts to provide the same functionality of the *MovieLens* web-based recommender [143] in a mobile environment. However, whether it can be termed mobile is questionable since the system simply generates and caches recommendations on a desktop machine and copies this data to a mobile device when synchronised through a wire. When in the movie rental store, the mobile device simply displays these previously generated recommendations. The system fails to be highly dynamic, one of the requirements necessary to the success of mobile systems previously identified. The failure of *MovieLens Unplugged* system to adapt to the environment it is in may be due to the complexities involved in moving a large database onto a mobile device. Certainly, no single mobile device has the storage required to hold the entire *MovieLens* database. It is apparent that mobile devices must employ novel database techniques to those of desktop systems if they are to provide recommendations generated from significantly sized data stores.

Want et al. also make the requirement for a novel storage technique and management of storage space in mobile systems clear. They propose the *Personal Server*, which is a mobile device that stores all of a user's data and continually monitors their activity. Obviously, such a device requires a substantial amount of storage and although Want et al. envision such storage to be available on the device itself, they realise that there are substantial benefits of a data store, which is essentially distributed throughout a community of mobile devices [170]:

> *Although a Personal Server device can potentially store a user's entire personal data collection, a distributed storage system such as Coda or Bayou would be very useful in case of theft, loss, damage, or concurrent access. Using such a system, the user could modify their mobile data while disconnected, and automatically transfer changes to the infrastructure when able. Such a capability would allow easy recovery in the case when a device becomes inaccessible. Similarly, these systems would allow data to be directly modified in the infrastructure, eventually propagating to the mobile device. Without such a capability, the user would be required to manually manage their data backup and migration, which would significantly detract from the user experience.*

The work discussed as a result of Want et al's *Personal Server* is of relevance elsewhere to this thesis as the envisioned uses and usage environment of the device raise questions relating to usage models, adaptive user interfaces and device discovery—all topics which are discussed in this thesis in sections 3.2.3.1, 7.1.1 and 4.3.2 respectively.

Later work by Goren-Bar and Kuflik reinforces the view that mobile recommendations can and should be highly adaptive. They identify that previous work came to the same conclusion and cite some of the benefits the adaptation can provide for mobile recommendation systems [74]:

> *Previous studies have shown that adaptive mobile devices are more effective than non adaptive ones. Billsus et al. (2000) found that adaptation reduces the amount of information that needs to be transmitted, and helps users access relevant information with minimal effort.*

Goren-Bar and Kuflik also study the user experience and find that 90% of users prefer an adaptive mobile recommendation system to a non-adaptive one. This further strengthens the requirement for highly-dynamic mobile recommendation systems.

Mobile recommendations are later used in this thesis to drive a mobile adaptive infrastructure, *Domino*, described in Chapter 7.

2.6 Seamful design

One interesting element of mobile systems is often the unexpected methods users employ when using them. Such behaviour was observed during trials of *Can You See Me Now* [64]. The game consists of two types of participant: runners, who are physically present within a city, and online players who utilise a web interface to interact with the game. The runners must chase and capture the online players by moving their slow-moving avatar over the online player avatars on a map presented to them through their web browser. The runners' positions are tracked using GPS units connected to PDAs which they carry.

Whilst GPS inaccuracy is often thought to be a problem, the runners in *Can You See Me Now* found it advantageous [64]:

> *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...*

Such use of inaccuracies or shortcomings in the underlying infrastructure of a system are often more apparent or more exploitable in the mobile environment, simply because mobile infrastructure is generally less reliable than desktop infrastructure. It is interesting to note that whilst knowledge of the GPS infrastructure and its drawbacks was advantageous to the runners in *Can You See Me Now*, lack of knowledge of the same items 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 [64]:

> *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 predict better 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.

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 drawbacks when they occur. Gaver et al. realise similar benefits found when approaching ambiguity from a similar angle [68]:

> *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 encourage users actively to understand better the concepts and infrastructure behind them and thus, in turn, to understand better how they come together to provide a working system.

This is particularly relevant in the mobile environment as mobile devices frequently use an extensive amount of sensors, and often rely on peers and infrastructure to complete tasks. This leads to an increased chance of breakdowns which can leave a user frustrated or confused if a suitable level of information about the underlying technology has not been delivered previously. Flintham et al. and Gaver et al. both theorise that users can be trusted to understand and appropriate information about infrastructure to allow them to overcome breakdowns in novel or known ways.

This topic, *Seamful Design*, is later discussed in Chapter 4 (section 4.2.2) when, after experimenting with it in a mobile system, it becomes apparent that its application can positively influence mobile systems.

2.7 Conclusion

The review of literature presented here makes it clear that many researchers share a belief that mobile computing has failed to fulfil expectations of mobility, flexibility and usefulness that were common in the early nineties. There is a shared belief throughout much of the work discussed that modern mobile systems fail to be reactive to users' context, and to adapt appropriately to the many situations mobile users may find themselves in. This failure prohibits devices and applications from being used as mental extensions of the form Hull et al. propose [90], or from becoming invisible as in Weiser's vision of ubicomp [172]. Instead, most modern mobile devices and applications are inflexible versions of standard desktop applications, incapable of sensing context and thus forcing users to alter their behaviour and work around the application's failure to adapt to social and situational context.

Much of the existing literature points out that the level of adaptation required to allow mobile applications to have the flexibility to behave appropriately in the many situations in which they may find themselves is extensive. Adaptation must be a reaction to not only the external context sensed by the device, but also the system's own state. The topic of adaptation in mobile systems is discussed throughout the thesis but is primarily addressed in Chapter 7.

It was found that previous work has identified a user's or device's location as being the most critical item of context information to mobile systems. However, it has also been shown that reliable, accurate and available position information has not yet been achieved. Without a positioning system that can determine a user's location in all mobile situations, both inside and outside, the number of possible

useful mobile applications is severely reduced. As position is so fundamental to mobile systems it is discussed in detail in Chapter 5.

The literature review also identified that peer discovery is important for mobile systems which operate within peer communities on a frequent basis or periodically enter the range of peer devices. However, it is clear that there is, as yet, no standard or robust way of discovering peer devices in mobile environments. Furthermore, it is also apparent that mobile developers often fail to consider the advantages and disadvantages of underlying communication infrastructure and often make poor choices that may later hinder the system and the user experience. For example, this was experienced with the DigiDress system in which users reported that the reliance on Bluetooth was a barrier to smooth interaction with other users. Peer discovery and appropriate selection of communication technologies is discussed further in Chapter 4.

Whilst previous work draws attention to the failure of mobile applications to be flexible and adaptive, and does highlight issues that may be tackled to aid this problem, it rarely addresses the issue of mobility itself. It is clear that the majority of mobile applications actually fail to be available or useful in many situations and locations. Whilst failures in context-awareness and subsequent reaction to context can account for some of these problems, there are many systems that seem to fail to be useful in many mobile environments for other reasons. For example, the GUIDE system is only available at locations where information has previously been authored for it and at which special GUIDE network nodes have been placed.

In the following chapter a more in-depth investigation and analysis of two systems is presented for two reasons. Firstly, to verify that the findings of the literature review are valid and actually do occur in, and influence, mobile systems and secondly, to identify other issues not found in the literature review which may affect mobility, availability and flexibility in mobile systems.

3 Investigation of two mobile applications

As an initial examination of whether the findings from the literature review are indeed applicable and relevant to improving the context-awareness and mobility of mobile, peer-to-peer systems, two mobile systems were examined. The first, the *Lighthouse*, is a system that was designed to have a degree of mobility and, although tested at one particular location, was hoped to be re-deployed. The second, *George Square* was trialled in the city of Glasgow but was designed to be useable in any city without requiring any setup whatsoever.

By studying the infrastructure and technologies each of these systems depended on, a summary of key findings and features important to mobile, peer-to-peer systems are extracted.

3.1 The Lighthouse

An important point to note is that the *Lighthouse* project was designed, implemented and trialled before any research for this thesis had begun. Indeed, most of the papers published about the *Lighthouse* had already been submitted. The author did not personally contribute in any way to the design or implementation of the system. However, as it was apparent that the system suffered from a number of deficiencies and problems typical of mobile systems, it proved a prime opportunity. Analysing the results of the trial, inspecting the positive and negative aspects of the system and deciding on how to improve them was fundamental in identifying many of the key issues on which this thesis concentrates. In short, although it was not designed to be a mobile system, analysing it as such quickly uncovered some of the areas that it is vital to take into consideration when attempting to create a flexible and useful mobile system.

The *Lighthouse* system was the first ever, and today still remains the only, museum co-visiting system that allows a user who is physically present in a museum to share their visit with multiple remote users who utilise either a web browser interface or a three-dimensional, virtual reality interface. The system was designed for a specific exhibition: the Mackintosh Interpretation Centre in The Lighthouse—Scotland's Centre for Design, Architecture and the City. The Interpretation Centre is devoted to the life and work of Charles Rennie Mackintosh (1868-1928)—an architect, designer and artist. Although the exhibit room itself is rather compact (less than 10x20 metres), it manages to contain a surprisingly large amount of information about Mackintosh. This is partly due to the fact that many different types of media are used to deliver the information. In addition to the drawings, models, furniture, paper documents and pictures on display; there are over twenty screens (some of which are touch sensitive) presenting video and interactive material through which a visitor can navigate a substantial amount of information on Mackintosh.

Although creating accurate and detailed copies of the many individual exhibits in the museum was an important part of designing and implementing the system, after a thorough analysis of how people

ordinarily visit museums, it was decided that the most critical aspect to concentrate on was providing tools to enable smooth interaction in the communication channels that all the visitors used. Whilst the exhibits within the museum provide great interest, our studies found that it is often discussing them with friends or colleagues that provides the most enjoyable aspect of a museum visit. In order to support this smooth communication in the *Lighthouse*, a substantial effort was made to provide three main resources for awareness and interaction that would aid the visitors in interacting with one another. These were: a shared audio channel, awareness of others location and orientation, and a common information space. As each of the three visitors in the system would be using a different resource for interaction through which to conduct their visit, it was necessary to provide each of these three features in a unique way to each type of user.

Whilst the other primary goals of the *Lighthouse* system were to experiment with co-visiting and delivering a heterogeneous range of information to the users, the system also provided vital insight in determining the weaknesses a mobile system may suffer from and ultimately led to attempts to create techniques that would overcome these problems in future systems. In particular, much of the core design and infrastructure was reused in later mobile systems with only slight variations on how they were used within the *Lighthouse* project. For example, the *George Square* system, discussed in a later chapter, directly built on the majority of the components that composed the *Lighthouse* system. Therefore, it would be remiss not to investigate what these system components were originally designed for, how their original use and findings helped improve them in subsequent systems, and how they were later altered and adapted to be more generic in order to allow them to fit into a greater number of mobile environments.

3.1.1 System overview

The *Lighthouse* co-visiting system allowed three people to visit the Interpretation Centre simultaneously: one physically at the location and two situated at remote locations and relying solely on the technology we provided as their means to visit the museum. The physical visitor is in the Interpretation Centre itself and carries a PDA equipped with wireless headphones and microphone (Figure 1).

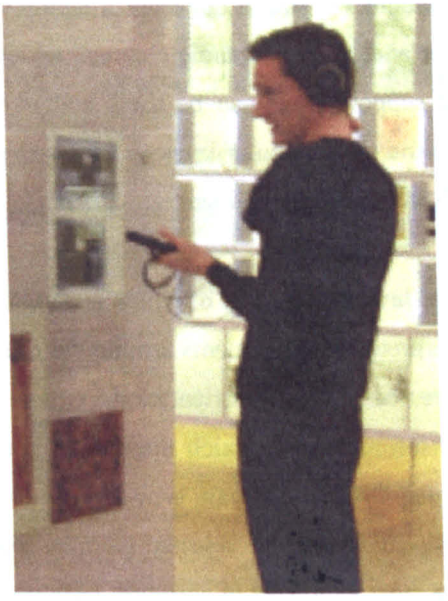

Figure 1: Visitor using the Lighthouse system in the Mackintosh Room

As this visitor is physically located in the museum he or she does not rely on the system to provide the entirety of the visit experience. That is, he or she does not need to rely on it to view or interact with the exhibits, but does rely on the system to stay in contact with, and keep awareness of, his or her co-visitors. The PDA he or she carries includes a sensor package that is part of an ultrasonic positioning system that allows his or her own location and orientation to be tracked (Figure 2). This information is displayed on a map of the Centre on their PDA, along with the locations and orientations of the other two visitors. This, along with the audio channel shared through the wireless headset to the other visitors, allows the physical visitor to determine if their co-visitors are nearby in their respective representations of the museum space and whether or not they are currently viewing the same exhibit. This continual awareness of what the remote visitors are doing in the museum space permits a free-flowing and smooth discussion of the exhibits as all the participants wander around the museum.

Figure 2: The PDA with attached positioning equipment used by the physical visitor.

The web visitor may use any standard PC and in our trials a laptop was used. Their visit is entirely conducted through a web browser and, again, an audio channel, which they shared through a headset. Any standard web browser that supports Java can be used, and in the trials Internet Explorer was employed simply because it was the most popular browser at the time and so the one in which trial participants were most likely to have previous experience.

The web site used displays several Java applets, one of which is a variant of the physical visitor's map. Mouse clicks on the map are interpreted as movements around the Centre, with the direction from the old location to the new location treated as the new orientation. Whenever the user clicks a location the nearest exhibit is calculated and information – which may be comprised of text, pictures, video or audio – relevant to the exhibit is displayed in the main browser frame (Figure 3). If there are multiple exhibits in close proximity to the location the user selected then, by selecting hyperlinks, the user may be asked to select the one they are most interested in currently viewing. Information about a particular exhibit may span several web pages and the user can simply navigate this information within the main frame as he or she would with any other website.

As with the physical visitor's map, the other visitors' locations are displayed on the map, shown at the bottom-left of the display, with differently coloured icons. Again, through using this interface along with the shared audio channel, the web visitor can be kept continually aware of what his or her covisitors are currently viewing and enjoy the feeling of a shared visit, much as they would if all visitors were conducting a traditional museum visit where all the users are physically present in the museum space.

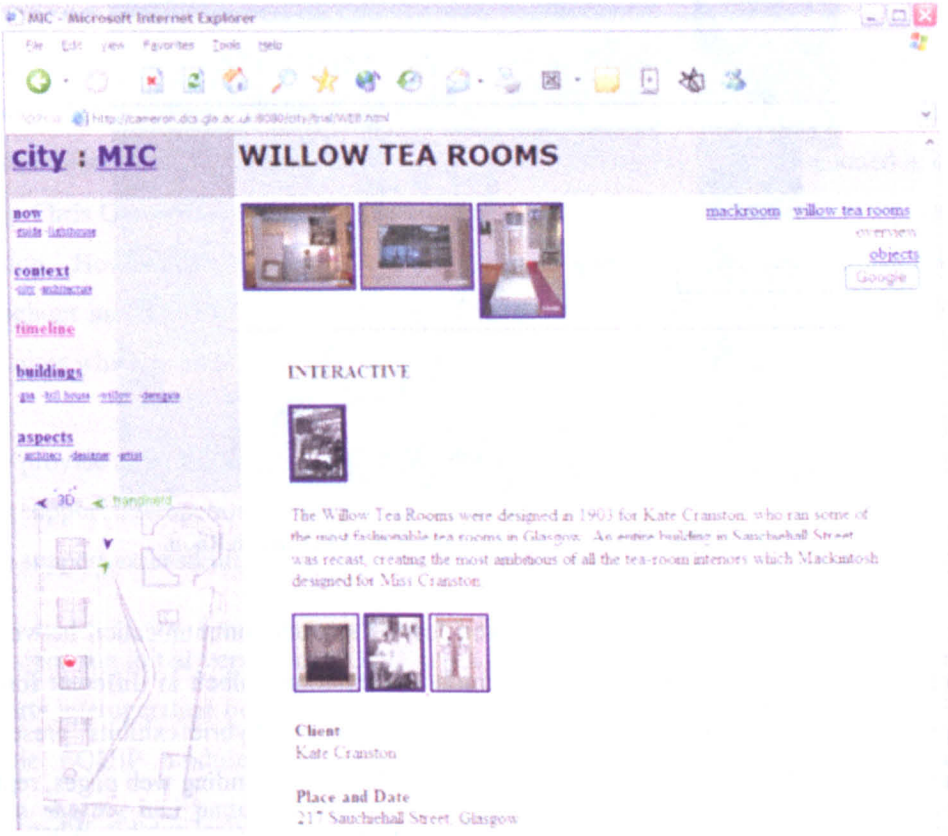

Figure 3: The web interface to the Lighthouse system

The final visitor uses a first-person, 3D display with avatars representing the other visitors (Figure 4) and, just like the other visitors, has a headset through which he or she shares audio. The textured 3D model of the gallery for this visitor's system had to be created from building plans and photographs of the exhibition area. Exhibits are modelled at a crude level showing form, but not fine detail. For example, the textual descriptions on the plaques by exhibits are unreadable within the 3D environment. In order to allow this visitor to experience the more intricate details of some of the exhibits, a cut-down version of the web visitor's interface is also provided. When the visitor moves close to an exhibit in the 3D environment detailed information about it is automatically displayed on his or her web interface.

The web interface utilised for this final type of user was significantly different from the pure web user as it only displayed a single frame showing information about an exhibit itself. As it was designed to be purely passive and simply update with information about the current exhibit as the user moved his or her avatar around the 3D environment, it did not display the map of the area or a hierarchical overview that the pure web visitor had access to. It was included as a simpler alternative to embedding all the information within the 3D environment, which would have been preferable if time and resources had allowed. Although not a perfect solution, splitting the screen in half with one half displaying the 3D environment and the other showing the detailed information in the web browser permitted the 3D user to have access to the same level of information as the other visitors.

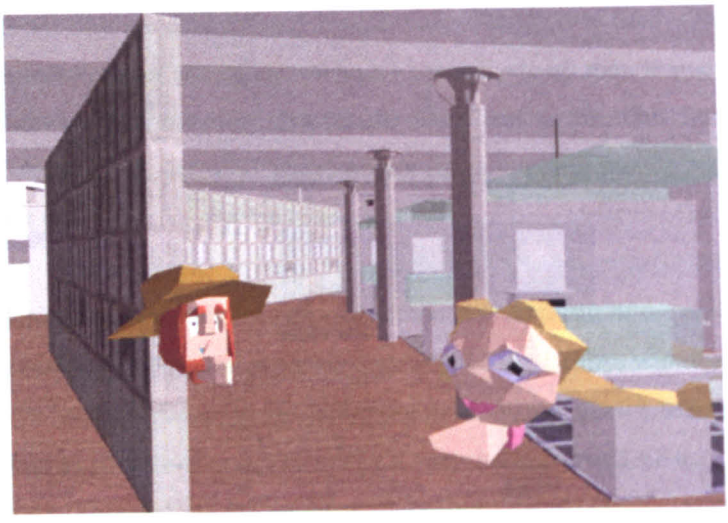

Figure 4: The 3D virtual environment of the Mackintosh Room

One of the main features of the system that allowed for the smooth communication between all the users to occur was that each user had access to similar information—albeit in different formats. In order to support looking at exhibits as a group, the system supports 'hybrid exhibits' presented both physically and digitally. Each physical exhibit in the Centre has corresponding web pages, reproducing the artefact as text and 2D images to stand as a digital version of the physical exhibit. When a physical and a digital visitor are both at a particular location they see comparable versions of the exhibits associated with that location. For the digital visitors, an exhibit is presented as HTML pages in a web browser. The spatial location of each digital visitor is converted to a name that represents a spatial extent, or zone, in the centre. A browser applet responds to a zone change by loading a new HTML page corresponding to main exhibit in the new zone.

While not designed to be a fully mobile system that could run anywhere, it had initially been hoped that it would be easy to deploy quickly the *Lighthouse* infrastructure at other exhibits and museums and that the VR and web visitors would be able to access the existing *Lighthouse* system and any newly supported museums from any remote location. Furthermore, the system was based in one small room and it was envisioned that the system could, in theory, be scaled-up and deployed over entire museums rather than just a single exhibit.

Thus, many of the system's main aims were the same, or closely related to, the aims a fully mobile system would have. For example, the system aimed to be continually available throughout the visit so that visitors could wander through the physical or digital environments and always remain in contact – sharing their visit regardless of where they were located in the physical or digital space.

3.1.2 Novel Infrastructure

This section gives a brief overview of the main pieces of infrastructure that were, at the time, unique and novel to the *Lighthouse* system and which greatly influenced the type of system the *Lighthouse* was and the type of experience it delivered to users.

3.1.2.1 EQUIP

EQUIP is an adaptive infrastructure created to support information sharing between heterogeneous devices. In its simplest form EQUIP can be thought of as providing shared tuple spaces which various clients can simultaneously share and listen for changes in. It was originally envisioned and then later designed by Chris Greenhalgh, who was also responsible for the majority of the implementation in the initial version. However, as with any complicated computer system, many people contributed small parts throughout its, still ongoing, development. At the time of the *Lighthouse* project EQUIP had three main aims which were [75]:

- to provide a run-time infrastructure for multiple domains
- to support interoperation between Java and C++
- to support extensibility

However, since the initial version of EQUIP used in the *Lighthouse* it has continued to progress and now supports interoperation between more languages and has become far more flexible. The number of available EQUIP modules—supporting such things as networking for servers and clients, information sharing and pattern matching—has also grown, further expanding the areas in which EQUIP proves useful.

Despite the many changes, EQUIP's core functionality of providing a shared tuple space between clients remains basically the same. By combining a messaging system with multiple private and public state spaces, EQUIP can provide comprehensive event distribution and state sharing. Clients are able to hook into others' state spaces and be notified of current events as they happen, changes to stored items in the state space or changes only to items matching a particular pattern. Due to EQUIP's ability to notify instantly any interested clients of changes in a tuple space, EQUIP can easily take on the additional role of a messaging system as any communication can easily be transmitted inside a tuple. Indeed, as previously stated, in the *Lighthouse* project the majority of information transmitted through EQUIP was simple communication messages between clients which would traditionally have been sent over a direct point-to-point connection utilising either TCP or UDP.

As EQUIP supports dynamic rediscovery of other clients and their respective state spaces, and as it maintains a record of all previous states and events, it can smoothly handle periodic connection dropouts between devices. Reconnection is carried out automatically when possible and each client will receive only the changes to state that were missed in the interim. Thus, EQUIP allows not only for smooth reconnection but also the seamless connection and catch-up of a late-starting client that will simply be delivered all the relevant information from the session that it missed prior to joining. EQUIP is also intelligent enough to send these messages in order so that any function that relies on a strict ordering of events will not fail.

In the *Lighthouse* EQUIP was used in a server-based configuration. One machine ran EQUIP and maintained a single state space within it. The clients each visitor ran left information in this one space and monitored it for messages and state information from the other clients. Thus, in the *Lighthouse*, EQUIP was utilised only as a simple database and as a messaging system. The state space also played the role of the database – storing all the information about the digital versions of the exhibits and the data required for the VR and web users model of the museum.

3.1.2.2 Auld Linky

Auld Linky is a lightweight contextual link server that can be used to store and serve hypermedia structures and filter search results based on context information. Data in Auld Linky is encoded as an XML linkbase that can then be queried through the standard HyperText Transfer Protocol (HTTP). The main advantage of Auld Linky's ability to filter results based on context is that it can serve the same source information in an appropriate format for a specific device and can tailor the delivered document to the user's experience. For example, if two users, one with a powerful device capable of playing audio and video, and one with a far less powerful device request the same document, the first may receive a movie whilst the second would receive a text document. Both the movie and the text document would describe the same information; Auld Linky is simply delivering it in the format most suitable to the user's device. Similarly, Auld Linky is capable of considering contextual information about users and may omit sections of information, or alternatively provide expanded versions of a description, based on the information it has about the user's knowledge level on a topic and what documents the user has read in the past.

In the *Lighthouse*, all the information about the exhibits in the museum was stored and served from Auld Linky. This meant the exhibits and information about them could be presented to each of the three types of visitor in a manner which was suitable for the device each was using and his or her location in the museum. Information could also be adapted depending on what other exhibits the visitor had already seen or had yet still to view. For example, for the web visitor using a laptop machine with a relatively large screen and good speakers, information on an exhibit may be shown mainly as pictures or a movie whilst for the visitor physically present in the museum, who can see the exhibit first-hand, only some supplementary information, not available at the exhibit, could potentially be presented in textual format on their PDA. Similarly, for a visitor viewing their first exhibit the description delivered to them may be appended with general information about Mackintosh and also point them to other related exhibits. Alternatively, if a visitor has already viewed the other exhibits that are related to the one they are currently viewing, the delivered document could potentially be focused only on the current exhibit or describe related ones more in the format of a reminder rather than suggesting the user revisit something he or she has recently viewed.

3.1.2.3 Positioning System

The locations of the three types of visitor in the *Lighthouse* system were each determined in a different manner, but their positions were all eventually mapped to the same underlying model of the museum. This model was stored on a central server and the users' positions were continually uploaded to the

same server so that each client could read them, thus allowing each user to be continuously aware of the others' locations.

The 3D environment visitor was easily located as their position was simply taken to be that of their avatar in the virtual environment. As this environment was built from the scale replica of the museum stored on the server, there was no need to translate their coordinates further. This user's orientation was interpreted from the viewpoint his or her avatar currently had within the 3D environment. That is, if in the 3D environment his or her avatar was directly facing North then his or her orientation was displayed as North to the other users of the system.

Web visitors navigated the museum model using a two-dimensional map. Clicking on the map placed their icon, an arrow, in a location and their orientation was interpreted as the direction of the previous location they had clicked to their current position. Again, as this map was constructed from the replica of the museum stored on the server, this location required no further translation and could be stored on the server in its native format.

The physical visitor proved to be the hardest to locate and it was here the most additional work was required. For the other two visitors the location was easy to obtain and required little or no extra code other than that required to upload the location to the server. However, locating the physical visitor required a substantial amount of infrastructure, setup time and novel technology.

An ultrasonic positioning system, developed at the University of Bristol, provided a solution for locating the physical user in the museum space. Eight ultrasonic transmitters were placed around the Mackintosh room (which is approximately 10x20m in size). As the transmitters are quite small it was possible to place them in corners, on the ceiling and on top of a divider wall in the centre of the room, in such a way that they were not noticeable to the casual visitor. Indeed, even with knowledge that they had been installed it still proved difficult to find them. A single RF transmitter was used to synchronise these ultrasonic transmitters embedded around the museum. When the ultrasonic transmitters received the RF signal, sent at set intervals, they all emitted an ultrasonic pulse. A receiver attached to the user's PDA listened for these pulses and delivered them to the PDA through a standard, wired, serial connection. Once on the device, the differences in the flight times of the signals were compared and used to calculate the user's positioning. The receiver attached to the PDA was also augmented with a compass from which the user's orientation could be obtained. Once the position and orientation were known on the PDA, they were displayed to the user on the map shown on the screen and also went over 802.11 wireless to the server and subsequently shared to the other visitors.

From the user's perspective, the ultrasonic location system was extremely lightweight and required no input or effort from them. As the required receiver was attached to the back of the PDA and continuously and automatically obtained the user's location and delivered it to the PDA, the user was largely unaware of the efforts being made to locate them in this passive method.

Despite the rather unusual layout of the Mackintosh room, using the ultrasonic positioning system provided a 50% accuracy of 0.52m, 95% accuracy of 1.83m and an overall standard deviation of 1.29m.

3.1.3 Factors influencing the mobility of the system

Although, as previously mentioned, the *Lighthouse* system was not designed with mobility, portability or scalability as primary goals, the lessons learned from the system were fundamental in understanding how certain weaknesses in a system could seriously impede usability in a mobile system. Unsurprisingly, the most immediate problems with the mobility of the system became apparent as soon as moving the system to another museum or expanding it to encompass a larger physical area were considered. Ultimately, these problems were severe enough for the system to be abandoned after a single set of runs in the *Lighthouse*, and not used again. This was a substantial disappointment as it had been hoped the system would be more widely implemented and, as it had been extremely successful in the *Lighthouse*, there was considerable interest from other museums who would have liked to trial the system and from peers in academia who were interested in gaining further results on how the system would run in general. Although following the *Lighthouse*, the Equator group at Glasgow continued to work in the same area of supporting co-visiting, to create a more mobile experience, it proved necessary to build a completely new system from scratch rather build upon the *Lighthouse* implementation.

Obviously, after experiencing the loss of such a substantial amount of work it became critical that the new system would not suffer from the same immobility that the *Lighthouse* had. Indeed, much of the incentive in creating the new system, titled *George Square*, was to address directly the limitations to mobility seen in the *Lighthouse* in order to create a system which provided the same rich co-visiting experience but was far more mobile and usable in a greater number of situations.

3.1.3.1 Number of users

One of the main constraints to the flexibility of the system was that the entire experience of visiting the museum through the *Lighthouse* system could only be achieved if there was exactly one of each of the three types of visitor sharing a visit simultaneously. This limitation was mainly caused through initial design decisions in which it was envisioned there would be no need to test the system in any other configuration. Simply, all throughout the design and implementation only a single scenario was considered and once the system was completed it would have been a substantial task to retrofit it to work with any other number of visitors.

This decision later proved extremely problematic as while running the trials there were many instances when one of the three trial participants cancelled at short notice - on these occasions the entire trial had to be abandoned. Clearly, if the system were permanently installed at the Mackintosh Centre it would be extremely frustrating for two tourists to find themselves in the museum yet unable to use the system simply because they could not find someone to take the role of the required third visitor. There were

also instances when after using the system for a period of time users expressed desires to change roles. For example, the visitor physically walking around the museum may become tired yet wish to continue their interaction with the system and their fellow visitors, and so take on the role of the web or VR visitor. However, such a switchover was not feasible as the system interfaces were considerably different—changing role required time spent learning the new interface—and so users were unable to transition smoothly from one role to another. The difficulty of changing roles was additionally compounded as, other than location, no information about other users' context was displayed by the system. The result of this was that it was impossible for a user assuming a role to "continue on" with any support the previous occupier of that role had been supplying. For example, it was common during the trials for the physical visitor to support the other visitors by describing specific details of an exhibit's visual appearance whilst it was common for the web visitor to supply the textual descriptions they had available. However, when a new user assumed a role no information on what their co-visitors had been viewing recently or what they were currently looking at was available. This essentially resulted in new users to a role having to begin afresh and gradually learn their co-visitors' context rather than being able to continue smoothly on from the last occupier of the role.

Unsurprisingly, research shows that collaborative user bases are extremely fluid and group members are transient, often only remaining in the group for a short period of time, during which time they may also be frequently changing the role they play within the group [52]. As mobile, peer-to-peer systems often result in collaborative groups of people coming together to drive the experience, these fluid group dynamics directly transfer into the mobile environment. Therefore, it can be expected that a system that requires an extremely specific number of users may be uncharacteristic of the majority of mobile systems, and exhibit reduced flexibility and mobility as a result.

This suggests that a mobile system must not only be designed to allow a flexible number of users but that it should additionally support the ability of users to change rapidly the *role* they play while using the system. As the problems with the *Lighthouse* demonstrate, and as the *George Square* system later confirms (section 3.2), this can be facilitated in two ways: maintaining a consistent interface between different roles and, if users are collaborating, making high levels of contextual information about other users available.

3.1.3.2 Time

Similar to the number of users, the times in which users could use the system were also extremely specific and caused further problems. Despite the fact that EQUIP has support for logging a large amount of state information and maintaining a history of it, the *Lighthouse* system only provided functionality to a group of people *simultaneously* sharing a visit. That is, there would be no benefit delivered by the system for two people visiting the Mackintosh Room at different times – something that is a common occurrence as one may visit alone and subsequently recommend the museum to a friend.

Another issue with time relating to the *Lighthouse* was that all users had to initiate their visits at the same time. It was not possible for two users to start using the system and have a third join them part-way through their visit. Again, previous literature shows that collaborative groups are extremely fluid with users both continually joining and dropping out of using the system [52]. Requiring a group of users to start and stop simultaneously their use of the system is simply unrealistic within a mobile environment.

Furthermore, regardless of whether a physical, web or 3D environment visit was made, there was no information maintained between visits, and thus the system did not allow subsequent visitors to take advantage or learn from the experiences of prior visitors. Each visit could potentially have added valuable information about the exhibits. For example, questions such as what exhibits were particularly interesting or what was a good order to view the exhibits in could easily have been answered by simply showing a user an overview of the visits previous users had conducted.

In short, information about visits could have been permanently stored and used to connect a series of visits or to shape richer experiences for later users. The system ran as a one-time event and any information the system generated and used during a visit was transient and completely lost after the visit ended.

The requirement for all users of a community to be present to utilise any functionality of a system is serious encumbrance for a mobile system. Increasingly, mobile communities are becoming more disparate in both location and time of use, and it is becoming preferable to provide as much functionality as possible to the isolated user, logging their use and any data they generate, and to synchronise opportunistically any data that is to be shared with the community when the user's device encounters peers.

To support this functionality it is clear that a mobile system should support both the dynamic discovery of peers and the dynamic formation of connections to peers. It must allow users to join and integrate with existing networks of peers, and to leave these groups subsequently, without any negative impact on the user's device or on the communities it connects to.

Attempts to overcome these time constraints in order to support both pre-visiting and post-visiting, and to allow users to join and leave a visit at any time were designed into *George Square*. In addition, *George Square* allows single visit experiences and reuses the information generated from them to improve subsequent visits for all users. These improvements are described in the subsequent section on *George Square*.

3.1.3.3 Network

The network configuration required by the *Lighthouse* system was extremely strict. All the clients required a constant and flawless connection to each other and to the server – which was responsible for running EQUIP, Auld Linky and a database containing information about the exhibits in the museum.

Due to the *Lighthouse* being one of the first projects in which EQUIP was used, and as EQUIP was still maturing at the time and under heavy development, EQUIP's abilities to support late-joiners and bring them up-to-date with others were not integrated into the system. Thus, if during the visit experience, one of the clients lost network connectivity for a period of time, it was not guaranteed that it would be able to rejoin without intervention from one of the trial organisers. Therefore, it was vital that the network connections to all machines participating in the trial were as robust as possible so that such failures were extremely unlikely.

In addition to requiring a robust connection, the *Lighthouse* system required a minimal amount of other traffic to exist on the network. Again, this was due to EQUIP's immaturity as some of the protocols it utilised at the time did not resend or gracefully recover from lost or heavily delayed messages. For example, whenever the system was run on the standard University network during in-house trials it would invariably fail within a few minutes as the relatively moderate level of traffic would interfere with EQUIP's own messages. Thus, the *Lighthouse* system required both a robust and an extremely quiet network.

To ensure the cleanest network environment possible, when the system was set up in the Mackintosh room a separate, small, isolated network was installed despite the fact that the building already had an available Ethernet network. The isolated network consisted of one router into which the server, web visitor's machine, VR visitor's machine and a wireless access point were connected by wire. As wireless 802.11 Ethernet is far less reliable than wired Ethernet communication, the level of wireless traffic was kept to a minimum and the only device that communicated with the wireless access point was the single PDA the physical visitor carried. The only traffic permitted over this small network was that which the system itself created.

The result of this restrictive network setup was that all the users suffered decreased mobility. Although the web and VR visitors both ran their clients on laptops that were capable of connecting to wireless access points, an Ethernet cable was required to ensure a robust connection; with the result that users could not move from the desks where the machines had been set up. Similarly, the user in the museum itself, who was the sole wireless user, could not use the system once they left the Mackintosh room as the wireless network did not stretch much further than the room itself. Not only did the users suffer from reduced mobility, the functionality of their machines was also reduced while using the system in that, as they had to use a particular network and network configuration, they were not able to connect to the World Wide Web or reach the Internet at all during the trial. This proved frustrating for some of the users who knew further information about the exhibits and Charles Rennie Mackintosh would be available on the Web but were unable to find it during the trial.

A final, frustrating network constraint that the trial organisers experienced was that the system had no peer-discovery mechanism built into it. This resulted in every machine involved having to be set with a specific IP address and programmed with the IP addresses of their peers in order to communicate

with them during the trials. In a more mobile system it would obviously be hoped that there would never have to be intervention from an organiser – yet requiring an average user to set their own IP address and discover others is clearly not acceptable.

These issues demonstrate some unique network characteristics of mobile systems. Firstly, mobile systems must be able to handle multiple network configurations intelligently and robustly—adapting to the current network types, topologies and traffic levels. Secondly, they must be able to do so without requiring input from the user. This is an extremely important point as mobile systems may often encounter substantial periods of time without attention from the user (for example, when they are carried in a pocket). If they are to continue to prove useful during such periods it is vital that they automatically sense and adapt to their current network surroundings.

Although improvements to the network were made in *George Square*, it is not until the later chapter on a novel type of wireless driver (Chapter 4) that a suitably adaptable solution to this problem is presented in this thesis.

3.1.3.4 Centralisation

Just as all the clients relied on a particular network configuration, they all also relied on a particular machine to drive the system. The server was critical to the operation of the *Lighthouse* system as it hosted: EQUIP spaces in which state information was held and messages between clients were transferred; an Auld Linky linkbase that stored the content about the exhibits; and an Auld Linky server that was required to serve the content in the correct format to the two types of remote clients. The necessity of such a server resulted in the standard problems of reduced flexibility and mobility. The most critical problem was that each mobile client had to have a constant connection to the server if it was to function. Without access to the server, a client may only provide extremely reduced functionality, none at all or crash the PDA. For the clients in the *Lighthouse* system, loss of communication with the server would invariably result in loss of all functionality for the web and VR visitors, and a complete lockup of the PDA for the mobile user.

The server also introduces a single point of failure – if it crashes or if there are network problems preventing it from being contacted then it may cause loss of functionality to not only a single client but to an entire community of mobile users.

3.1.3.5 Location constraints

Although it had been hoped that the *Lighthouse* system, or the majority of the infrastructure used in it, could be expanded or adapted to work in other physical locations with a minimal amount of effort, it was tied to the particular location of the Mackintosh Centre. This was mainly because the positioning system used to locate the user physically present in the museum required suitable areas where ultrasonic pingers could be placed, hiding out of sight yet themselves able to view large regions in order to provide a high level of coverage for the area. Even if the target area met these requirements, a substantial amount of calibration and set up was required to obtain a useable level of accuracy.

Clearly, it is not practical, due to time and cost constraints, to cover every possible area a tourist may visit with calibrated ultrasonic pingers. Therefore, although the ultrasonic positioning system proved valuable in the confined space of the Mackintosh Centre, it was unlikely to be suitable if multiple, or larger, locations were to be considered.

It is self-evident that not being tied to a particular location is of utmost importance to a mobile user. The ideal for such a system is that it be carried with the user and available at any time and in any location. From the literature review it is clear that hybrid positioning systems, which can reliably provide locations both indoors and outdoors, are most suitable for mobile systems. The *Lighthouse* positioning system fails to meet this requirement but further it also proves to be extremely costly and time-consuming to install. It was realised that even if the system were to be used at another indoor location this installation cost would be a significant barrier.

Therefore, new requirements for mobile positioning systems have to be identified based on the failings of the system used in the *Lighthouse*. These are: that the cost and setup in any positioning system, even hybrid ones, must be low for any location in order to provide coverage in as many areas as possible; and that any calibration be achievable through automatic techniques.

At the time the *Lighthouse* and *George Square* were implemented such a system did not exist. There is therefore a clear need for this positioning infrastructure for mobile, peer-to-peer systems. The requirements gleaned from the *Lighthouse* system and supported by the findings in the literature review became the core goals of a novel hybrid positioning system, the design and implementation of which is subsequently discussed in Chapter 5.

3.1.3.6 Content constraints

One of the main reasons users found the system enjoyable to use was because it allowed them to explore and discuss the exhibits in the Mackintosh Centre with their co-visitors. For the web and VR visitor, this meant that a massive amount of content authoring was required to create suitable versions of the exhibits that could be viewed online and in a 3D environment. As well as information about the exhibits – such as photographs, text, links to other related exhibits and content-matching patterns for Auld Linky – the entire Mackintosh Centre had to be mapped in order to create a digital representation of the space. The amount of authoring required in the *Lighthouse* system is perhaps its biggest constraint on mobility, as it requires the most time and effort on behalf or the designer or a dedicated content author before the system can be used at all in a new area.

In order to recreate the *Lighthouse* experience in another museum, every exhibit therein would have to be examined and, with the help of someone knowledgeable about the exhibits, a digital version including information about the exhibit's location in the museum, textual description and image would have to be designed and input, in the correct format, into the *Lighthouse* system. Additionally, possible contexts a user may find themselves in during their visit would have to be considered and entered into

Auld Linky and, if the data to be delivered to the end-user was in a format that EQUIP did not already understand, appropriate structures to hold the data would have to be created in EQUIP.

Clearly, the amount of content that has to be created at the outset to allow the *Lighthouse* system to function is one of the main inhibitors to its flexibility. It is simply not possible to cold-start the system nor is it possible to boot-strap it with a low level of information as the system itself does not support adding additional material or altering existing information.

This problem leads to an extremely clear and previously unreported requirement for mobile systems: that pre-authored content should be avoided wherever possible to allow users to roam to new areas without requiring a designer or dedicated content author to have visited the location beforehand. A method in which pre-authored content can be avoided whilst simultaneously maintaining a suitably high level of content was designed for the *George Square*. It is discussed further in section 3.2.3.3.

3.1.4 Conclusions

Whilst the *Lighthouse* was undoubtedly a success, since it demonstrated a unique co-visiting system that delivers a range of heterogeneous data to users during their visit, attempting to convert it into a mobile system reveals many problems. Analysis of these problems has led to guidelines for the future design and development of mobile systems, the identification of certain infrastructure that is lacking for mobile systems and, in some cases, simply the highlighting of existing problems. Thus, analysis of the *Lighthouse* system has proved vital to this thesis as it has identified problem areas in mobile systems and focused the remainder of the work in the thesis.

From the investigation of the *Lighthouse* some new mobile design guidelines have been exposed. Although each of these has been discussed in the chapter, they are further clarified here.

Support users changing roles

That users in collaborative systems change roles is not novel, and the fact that users' roles are fluid in desktop applications was highlighted by Dourish and Bellotti as early as 1992 [52]. Smith et al. point out that roles, and the change of roles, is a natural concept in humans [156]:

> *People in a group play various roles. Manager, intern, department chair, guest speaker, the woman who knows how the fax machine works, the guy who got the pizza, the person speaking now, the person capturing this on the whiteboard; roles can be formal or informal, long-lived or ephemeral.*

This highlights that roles may be assumed over long periods of time, perhaps permanently, but that they can also be fleeting, momentarily assumed and discarded. Smith et al. go on to discuss how roles can be applied in CSCW [156]:

> *When it comes to computer supported cooperative work, it is perhaps natural to try to be helpful by supplying roles directly. Group dynamics can be difficult to proscribe, and individual activities can vary from group to group, from one context to the next. Consequently, most workers in this area realize that a system that prevents a group from modifying their roles can hinder rather than help. But roles can evolve in unexpected ways, even in the midst of a collaboration. More recently there has emerged a recognition for the importance of tightness and flexibility in the midst of a session.*

It is clear that Smith et al. believe that a lack of support for flexible roles can be detrimental in CSCW. Muller et al. seem to concur, and propose a hybrid system and demonstrate a prototype that supports user collaboration in the short-term, mid-term and long-term on desktop machines [119]. They make clear that the short-term support is actually support for ad hoc collaboration that is previously unplanned and occurs spontaneously. Again, support for such spontaneous collaboration as well as longer-term collaboration is an important issue in the mobile environment. Just as mobile systems offer a wider range of contextual information, so they open up a greater number of possible roles—and greater support for fluid, rapid change within those roles is vital for smooth collaboration to occur.

Users of collaborative mobile systems are continually supporting one another in differing ways. Each user's unique context gives them unique advantages or access to information that they often utilise to aid other users. Alternatively, in non-collaborative mobile systems, such as games, players often wish to assume certain roles within the game and experiment with the possibilities of each. Smooth exchange of roles can be supported in at least two ways.

Maintain a consistent user interface between roles: By maintaining a consistent interface for different roles it is easier for users to assume new roles. The *Lighthouse* system fails to do this and as a result it was seen that changing roles within it was extremely difficult. As will be seen, *George Square*, maintains a consistent user interface for all its users and thus enables the rapid exchange of roles.

Information about the context of other users should be shared and made visible: In mobile systems as much information as possible about the context of other users should be shared and made visible by the system. Exposing the context of other users allows users to understand better their role and the roles of others, and to assume rapidly new roles when and if required.

Support fluid group dynamics

In mobile, peer-to-peer systems it is normal that the group of users is in continual flux with members joining and leaving unexpectedly. This guideline is distinct from the previous one as it addresses the issues of devices leaving and joining a group of peer devices whilst the previous guideline recommends support for role change within existing groups. Fluid group dynamics can be supported in two ways.

Allow and support late joiners: In mobile systems it is common for groups of users, or their devices, to work together to enrich the experience the system provides. The system should support new users joining already formed groups to allow them to share in the benefits the group provides. Information should be recorded on devices even when isolated or working normally within a group in order to deliver this information and allow new users to "catch-up" with the group when they arrive. A network technology and topology should be selected that allows new devices to join automatically and integrate with existing groups of peer devices.

Support the single-user experience: One important configuration of users that is often neglected in mobile systems is the isolated user. Mobile systems can support peer-to-peer users working in large groups but this does not mean the single-user experience is of no value. Not only should a system continue to operate and provide functionality for the isolated user's benefit, it should also attempt to make use of the actions and information isolated users generate. For example, by recording the choices made when a user is isolated from a group of peer devices the information can later be shared when peer devices are encountered in the future, for the benefit of the whole community.

Avoid reliance on pre-authored content

The design and implementation of the *Lighthouse* revealed that one of the most time-consuming and costly parts of creating the system was finding, recording and inputting the vast amount of content that was required for the digital versions of the museum. Mobile systems can, by definition, roam to any location and it is simply not practical that content can be collected and authored for every possible location. Therefore, pre-authored content must not be necessary to mobile systems if they are to be truly free to roam to any location. Instead, automatic content-generation methods should be employed where possible. Avoidance of reliance on pre-authored content can be achieved through the selection of self-generating or reliance-free content type architectures, which are discussed in section 3.2.6.

As a group, after reviewing the *Lighthouse* system, a deliberate decision was made at Glasgow to create a new tourist co-visiting system that directly addressed the mobility problems experienced in the *Lighthouse*. This new system, *George Square*, adheres to the three design guidelines identified here. It is described in the following section.

In addition to the guidelines discovered, the analysis of the *Lighthouse* also revealed two extremely important pieces of infrastructure that are currently not available for mobile devices. Firstly, it is apparent that a robust network communication method is needed that supports the dynamic groups of mobile systems. The communication method should therefore allow devices to smoothly and intelligently join and leave the peer groups that often spontaneously form in mobile, peer-to-peer systems. As part of this process, it is important that the system is capable of reliably and quickly discovering peer devices that are already present on networks. Furthermore, in contrast to the Lighthouse, it should continue to allow users to maintain their normal network functions (such as

connection to the Internet) whenever possible. The design and implementation of a network driver that conforms to these requirements is described in Chapter 4.

The second crucial piece of infrastructure required is a positioning system that works both indoors and outdoors and that has high availability. It is clear that in the *Lighthouse* the positioning technology used limited the system to a single room within a single building. A mobile positioning system should have the functionality to locate a user wherever they may be. Perhaps more importantly, the system should have little or no setup cost to allow users to roam to new locations without requiring additional infrastructure to be installed at the location beforehand. The design and implementation of a positioning system that meets these constraints are described in Chapter 5.

In conclusion, the analysis of the *Lighthouse* system was crucial to the work in this thesis. It identified important guidelines that can, and should, be applied to mobile systems in general, and led to the design and implementation of two vital pieces of infrastructure that were previously unavailable for mobile systems.

3.2 George Square

The *George Square* system, named after the location it was initially trialled in, attempts to address directly the many mobility problems faced in the *Lighthouse*. *George Square* is a co-visiting system intended to free users from location and mobility constraints experienced previously and allow them to utilise the system throughout an entire city. Whilst the main effort of the *George Square* system implementation went into overcoming the mobility problems identified in the *Lighthouse*, many of the underlying research goals remained similar to those of the *Lighthouse* – for example, examining the viability of manipulating and delivering a heterogeneous range of information to users. The system was implemented over a half-year period from January 2003 through to the summer when it was trialled in *George Square* in the centre of Glasgow. Work on *George Square* has since been published in [21] and [9].

The other main aims of *George Square* were to explore how collaborative ubicomp can work over an entire city space rather than a single confined location, and to encourage users to look beyond their own use of information and consider how their accessing the information may be perceived or utilised by other co-present or remote users. A secondary aim of the system was to explore how the concept of web-logging could be implemented into an existing infrastructure. Over the last couple of years web-logging, or blogging, has experienced an explosion in popularity but, despite their vast number, most bloggers edit their blog using standardised and rather plain tools thus resulting in the majority of blogs having an extremely similar appearance. *George Square* permitted an ongoing integration with a blog-like web site that allowed a user's location and the information they viewed to be continually logged and uploaded in order to generate a novel blog site.

In addition to these aims, *George Square* utilised a surprisingly large amount of what was, at the time, novel technology and so proved an excellent opportunity to investigate how this technology performed in real-world applications and how it could be effectively deployed.

As will be made apparent, *George Square* adheres to the design guidelines for mobile systems identified in the previous chapter and was used as a testbed to validate that these are indeed beneficial.

George Square was designed at Glasgow University, primarily by Matthew Chalmers, Barry Brown, Ian MacColl and myself. The majority of the code for the system was implemented by Ian MacColl and myself, although Barry Brown did also contribute slightly to the coding effort.

3.2.1 System overview

George Square allows visitors to a city to share their visit with other visitors in the city or with remote visitors via the Internet. Tourists in the city carry a tablet PC running the *George Square* software and augmented with a GPS receiver and an USB-stick camera (Figure 5).

Figure 5: *George Square* user with tablet PC.

Remote users can share a visit with those in the city or, as the system delivers a wealth of interesting information about the city, conduct a completely independent remote visit entirely over the Internet. Although a visitor to the city is free to conduct his or her visit as they traditionally would, the system provides several useful features to guide and advise the user through the city, enrich the visit with additional information and enable the visitor to closely share his or her experience with others travelling with them or with remote users.

The majority of the *George Square* interface (Figure 6) shows a large map revealing the user's own location, the location of co-visitors and recommendations for locations, web pages and photographs.

As the user conducts his or her visit through the city his or her location is continually tracked using the GPS unit and displayed on the map to allow the user to navigate easily (mark 1 in Figure 6). If the strength of the GPS signal is too low to get an accurate fix, or if they want to fake their position, the user may switch to "manual" positioning where they simply click on the map the location they wish to appear at. The locations of any co-visitors, who may be automatically or manually located, are also shown.

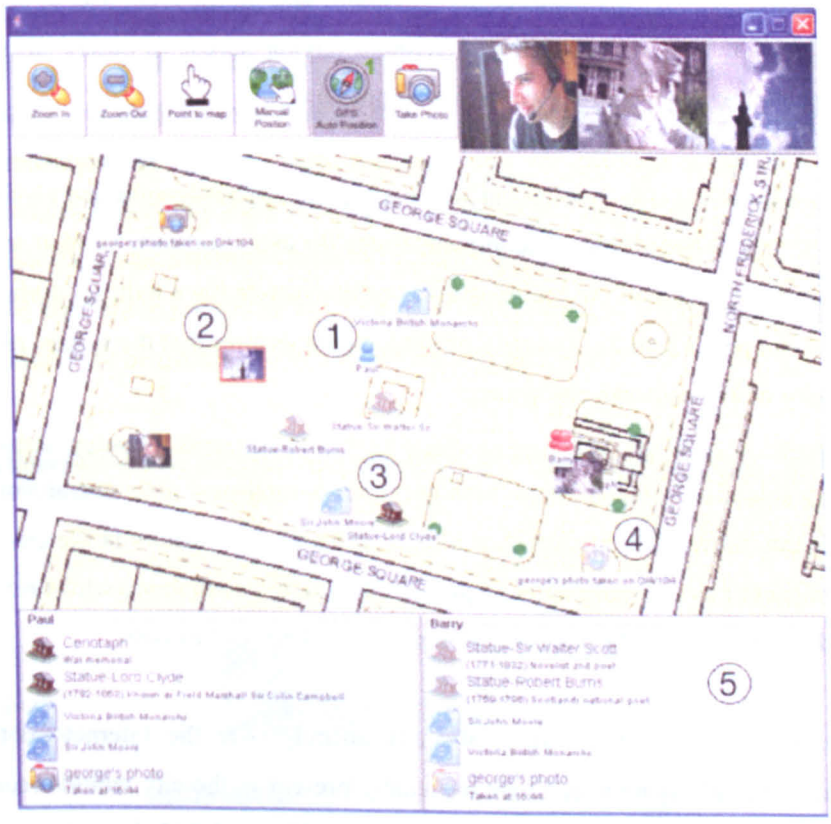

Figure 6: *George Square* interface showing user's location (1), photographs (2), locations (3), photograph links (4) and recommendation lists (5)

Visitors may use the USB camera at any time to take photographs that are automatically geo-referenced. Thumbnails of any photographs a user takes are subsequently placed onto the map in the interface. Clicking on the thumbnail on the map opens the full-size image. Photographs are shared with co-visitors so that others in remote locations (in the city or on the Internet) are able to view and discuss the images as they appear. In addition to appearing on the map, the last three images taken are displayed on a filmstrip at the top-right of the interface. This allows a visitor to quickly see the latest image taken by another without having to scroll their own map to where their co-visitor is currently located. Again, clicking the images on the filmstrip opens the full-size image.

As the user's icon on the map moves around, either through being tracked using the GPS unit or through the user clicking to manually locate his or her avatar, the system generates recommendations for locations he or she may wish to visit, web pages that may be relevant to items nearby and photographs for pictures taken by previous visitors. These recommendations are generated using the

Recer [27] algorithm and are thus relevant to the user's current context[5]. The recommendations are displayed at the appropriate locations on the map and in a list at the bottom of the screen. Clicking a recommendation will open the relevant web page, photograph or highlight the location on the map. In addition to their own recommendations, users see the recommendations currently being shown to other visitors as ghosted icons on the map and in a list, and one may click and open these recommendations just as with one's own. As had been found in the *Lighthouse*, having access to what another sees is vital in allowing for smooth discussion. For example, in *George Square,* it is typical for one user to suggest something like, "let's try my third recommendation" and for the other user to perceive instantly the recommendation that is being referred to. Without having access to others' lists, such smooth discussion and immediate understanding would not be possible.

As the user moves around the city—looking at web pages, taking photographs and visiting locations—all these actions are logged and stored. Recommendations for users currently visiting are subsequently generated from the history logs all the previous users of the system have built up during past visits to the area. This constant and automatic updating of the log data ensures that the system remains relevant and adapts as the city or events in the city change.

In addition to being able to see other visitors' locations, photographs and recommendations, visitors are also able to collaborate through discussion over a VoIP connection. Visitors in the city used the built in tablet microphone and pair of headphones to provide their audio connection whilst remote users had the option to use their machines' inbuilt microphones and speakers, or headsets.

Remote users are able to conduct a visit to the city entirely over the Internet—with the system providing them the same information as a visitor actually present in the city would receive from their tablet PC. By using manual positioning, and clicking where they wished their avatars to be placed, remote visitors can view recommendations of photographs, web pages and locations as they would if they used the system whilst visiting the physical locations. In this way, the remote visitor can view images of the city and learn about it through the web pages that are recommended. Obviously, the remote visitor is able to hold a discussion with, and view pictures taken by, any user who is currently using the system to physically visit the city.

Using the remote interface, a tourist who is planning a visit to the city can conduct a pre-visit—essentially using the remote interface to learn about the city and plan their trip. Once in the city, he or she can again use the same interface, which he or she is now familiar with, whilst conducting the actual physical visit.

3.2.2 Blog system overview

In addition to *George Square's* visiting interface, a secondary blogging interface is automatically made available after a remote or physical visit is conducted. Although this section of the *George Square*

[5] Further details of Recer can be found in section 6.3.1.

system was not part of the main trials of the system, some users were questioned about their use of this part of the interface and access logs of the site were examined. As the user travels around the city his or her actions are recorded and stored, over time creating a history of his or her use. Subsequently, a web interface to this stored information is made available (Figure 7). This interface provides several sections of information. Firstly, a map with the path the user walked overlaid in red is shown and any photographs he or she took or web pages visited during the visit are presented as icons on the map. Again, these icons can be clicked to view the full photograph or visit the web page.

Controls for filtering the data based on a certain time period, area of the city, web page or recommendation are provided, in order that a user who has conducted a long or complex visit can view a more manageable subset of his or her data. Separate controls allow for comments to be appended to entries – for example, a user may wish to leave a comment about a statue he or she visited or explain why a certain web page was relevant whilst he or she was at a specific location. The user may also choose to remove entries for privacy reasons or simply because he or she may anticipate a section of the visit which would not be exciting or interesting for others to view.

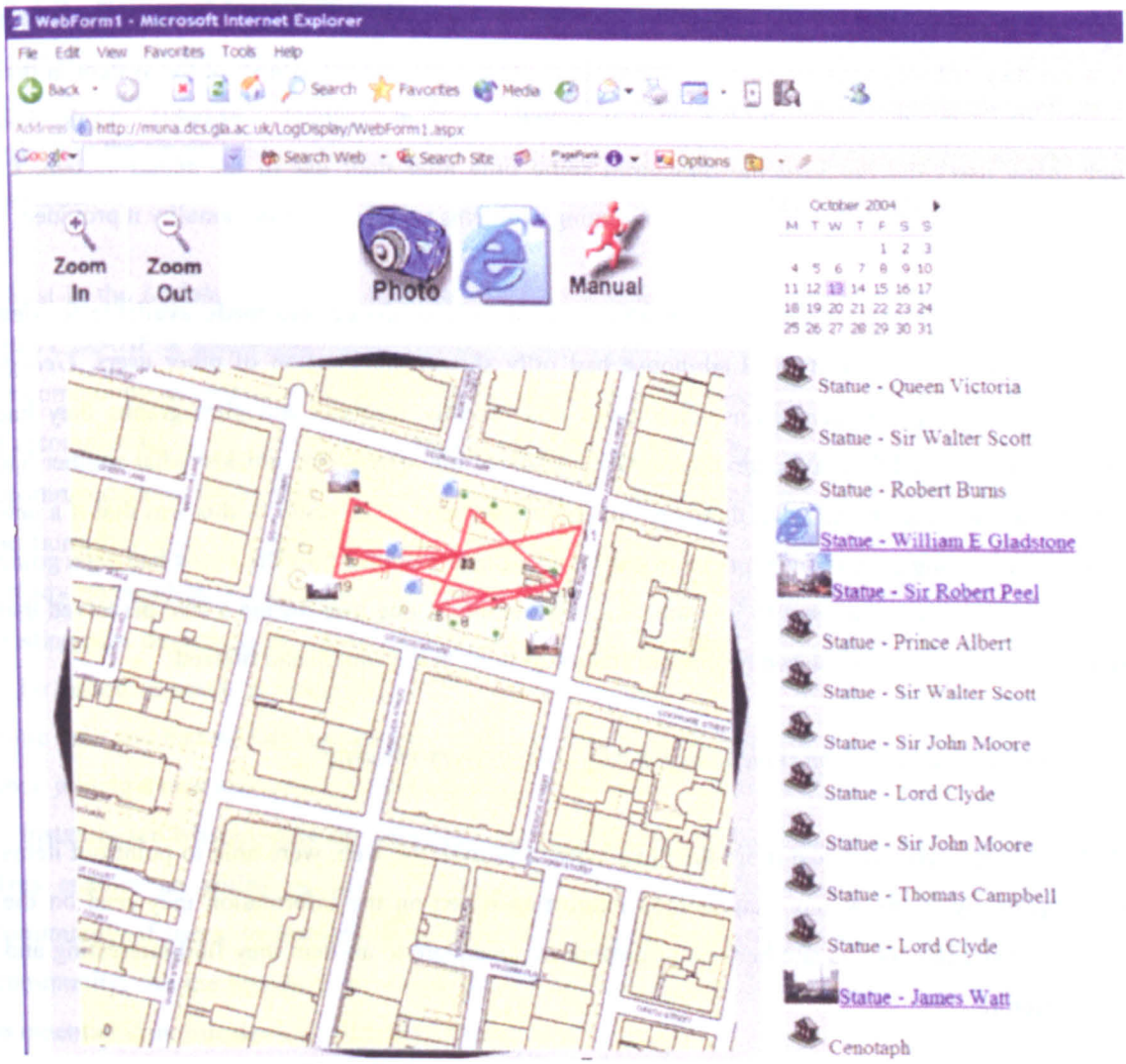

Figure 7: The *George Square* post-visit blog interface

Another section, below the map, provides the same information in a list format. A user who has visited the city can use the map and the list to relive or remind themselves of their visit. Alternatively, the website, possibly after editing, can be used as a blog and shown to friends or family. As a password is required before editing is permitted, these external visitors may only view the site rather than edit it.

3.2.3 Improvements from the *Lighthouse*

As stated, *George Square's* main goals included directly addressing the problems for mobility faced in the *Lighthouse*. The following three sections describe how the three design guidelines identified from the analysis of the *Lighthouse* were beneficial to *George Square*. The fourth section describes how modular design was beneficial to the trial designers in *George Square* and theorises that the benefits of modular design could be passed on to the end user.

3.2.3.1 Support for changing roles

George Square adhered to the guideline that mobile systems should be designed to support users changing roles, or establishing their own roles, in several ways. Firstly, the user interface for co-visitors was identical regardless of whether the users were physically present at the location or conducting their visit remotely. This proved extremely helpful as in several of the trials users expressed a desire to change from indoors to outdoors or vice-versa after using the system for a while and, when they did so, experienced no problems in continuing to use any aspect of the system in their new role. Similarly, the post-visit blog site was designed to be as identical as possible. Again, even though many users did not visit the blog until some time after their use of the actual system, the familiarity of the interface resulted in no users having problems utilising the functionality it provided.

Secondly, a large amount of context information on users was shared and made available to view through the interface. Whilst the Lighthouse had only shown the location of other users, *George Square* showed the recommendations other users had recently received and photographs they had recently taken. This additional context information allowed one user to see quickly what another had recently been interested in and what they were likely to view next. The result of this was that if a new user joined an existing group of visitors he or she could quickly perceive how their visit had been going and smoothly integrate with them to join the visit. For example, one user during a trial perceived that the remote visitor's icon was placed near some statues of lions in the square and offered:

> *"I'll take a picture of the stone lions for you. They're very majestic".*

Similarly, remote users, who had more time available to browse the web, were able to point out items to those physically in the square that seemed interesting based on the information they read on the Web. One remote user points another visitor present in the square to an item they find interesting and requests a picture:

"Or see if you can get that picture of that human rights plaque that's just near you just now. There's a human rights plaque that's just near you. I've circled it, don't know if you [can see]"

This smooth collaboration was made possible by following the design guideline discovered from the *Lighthouse* – by making context information about other users available it is clear that users better understood the current, and possible future, actions of others. Not only did this benefit current collaboration, it also greatly reduces the effort and time a user assuming a new role must observe others before he or she can offer similar help. This was repeatedly observed in trials in which the user physically present in the square often started exploring the area before the remote user's system connected. When the new remote user connected, he or she was instantly able to see what the user in the square had already looked at and what recommendations had been made. Therefore, it was extremely common for the first action upon perceiving this to be an offer of help to find more information on either what the user in the square had just seen or on one of the items that had been recommended, and so was likely to be viewed next.

3.2.3.2 Support for fluid group dynamics

One of the guidelines discovered from the *Lighthouse* was that mobile systems should support fluid group dynamics. Specifically, they should support late-joiners to an experience as well as not neglecting the single-user experience. *George Square* exemplifies a system that closely adheres to this guideline.

Whilst in the *Lighthouse* EQUIP was used only as a synchronous communication system, in *George Square* EQUIP is additionally employed as a type of database which logs the events passing through the community as well as the states of all the devices within the community. For example, when a set of recommendations is delivered to a device, the event of delivering recommendations, and a list of the recommendations themselves, are logged by EQUIP. Similarly, any photographs users take or any locations they visit are stored by EQUIP. Subsequently, if a new device joins a group of peer devices already engaged in a visit, this information about recent activity is passed to them by one of the existing peer devices in the community. Indeed, it is possible for every action since the system began to be passed to newly arriving peers. Thus, even if a group of users had been utilising the system for some time and had already moved around the city taking photographs and receiving recommendations, upon joining a new user would immediately be delivered all this information. This allows the new user to instantly see where others are, catch up with photographs and browse the recent recommendations. This enables an ability in newly connecting visitors to integrate rapidly and smoothly with a community of users, to learn the context of the community and to behave appropriately within that community. As the literature review earlier pointed out, support for correct and appropriate behaviour is essential if mobile devices are to be accepted by their users.

The ability to catch quickly up by examining events missed whilst a device was not a member of a community, either because it had never joined or because it had temporarily left, has the additional

advantage of supporting mobile devices which may face the generally less reliable network infrastructure of a mobile environment. For example, if a mobile device experiences disconnection from a peer community due to network problems, or simply because the owner turns it off to conserve battery power, this ability can ensure it can smoothly reintegrate when a connection is again available without adverse affects – any information that had passed in the interim would be delivered, in order, as soon as the machine rejoined the system. In the *George Square* trials, this was experienced in virtually every individual trial. The north-east edge of the square was rather distant from both the remote visitor's device and the infrastructure access points we had set up in the area. Thus, when a user entered this area of the square, neither a direct connection to a remote visitor's device or a connection through the infrastructure access point was available. However, their system continued to operate, although it did not provide live updates from co-visitors. Once it re-entered the region where coverage was available, the first peer it encountered would automatically update it on the events and information it had missed. Similarly, the device would do the same for the actions the user had performed whilst outside network coverage, propagating his or her actions to others.

The advantage of this configuration was that set, static peer communities were not necessary. Individual users could break off from the main peer community and work on their own for a period of time – rejoining in their own time without losing any information or functionality. Furthermore, the system actively supported the individual user working on his or her own and recorded information for later sharing with the community. Unlike in the *Lighthouse*, where users who became disconnected would experience system crashes, the *George Square* system allowed users to continue using the system whilst isolated from the larger community of users. New recommendations on locations to visit would still be generated and delivered, and any locations or websites the user visited or photographs he or she captured would be stored and shared to peers if the isolated device became reconnected in the future.

This behaviour is vital in a mobile community, as there are often large periods of time when devices are isolated from either infrastructure or a peer community. The *George Square* trials revealed that in virtually every trial at least one device operated in an isolated fashion for some period of time. As periods of isolation are normal for mobile systems it is important that functionality continues in these situations and ongoing information is recorded, in order that the community may benefit from the data collected.

By following the guideline of supporting fluid group dynamics, the *George Square* system smoothly handles disconnections and reconnections of peer devices within the community and strongly supports isolated users. Furthermore, it actively logs information isolated users generate and ensures it is shared with the larger community of users when peer devices are again encountered.

In addition to the actual experience of visiting a tourist location, research highlights the importance tourists give to both planning a trip and to reviewing or reliving a trip once it is completed [19]. Whilst

the *Lighthouse* failed to support pre-visit and post-visit activities, *George Square* aided in both tasks – thus supporting isolated usage in more scenarios and further increasing the availability of the system. Before a visit, the system could be used to experiment with paths through the city. By using the 'manual locate' function to position ones avatar, one was able to experiment with various different actions and histories—viewing the recommendations for nearby attractions that would be delivered as one did so. Furthermore, if the user then subsequently did join a group of others to conduct a visit, the information from this pre-visit would be shared with the peer devices that are newly encountered. This effectively allows the peers to learn from the plan the user created and generate recommendations more in accord with this pre-visit.

Finally, due to the consistent interface for different roles and the implementation of smooth integrations for new devices to a community, *George Square* did not suffer the same problem of requiring a set number of users as had previously been experienced in the *Lighthouse*. In the *Lighthouse*, the strict requirement for an exact number of users, and for each user to fulfil an exact role, greatly restricted the opportunities people had to use the system. Two users who wished to use the system while visiting the Mackintosh room would be unable to do so unless a third person could be found, simply because each part of the system had been designed to look for connections to two other peers, and not to begin unless they were found. Furthermore, each of the three users would then have to assume a specific role of physical, web or VR visitor. This was normally impractical as the most common types of visit to the museum were pairs who wished to physically visit the museum together. Many visitors voiced a desire to conduct a purely remote visit with other distant friends – no members being physically present in the museum itself. The trialled scenario, one physical visitors and two different remote visitors, was not particularly desired by most groups and even felt quite forced during the trials.

George Square made no such assumptions as to the number of users or the roles they should fulfil when using the system. It supports a variable number of users—from a single user on his or her own to a large group. Not only can the number of users vary, the types of users are not important for the smooth operation of the system. A single user may be physically in the city or conducting the visit over the Internet. Similarly, a group of users may consist purely of tourists in the city, purely of remote visitors or a mixture of the two.

The substantially increased freedom of *George Square* was a far better fit for a mobile community where group members and their machines are transient, and where the number and type of users is rarely static.

3.2.3.3 Avoiding pre-authored content

Possibly the largest and most significant problem with the *Lighthouse* was that it required a considerable and extensive amount of content to be authored for whatever location it was to run at. This involved finding a professional knowledgeable in the area, who was prepared to give their time, and liaising with them to transpose their knowledge into digital format. This process represented perhaps the largest percentage of time in implementing the system. It also meant that the entire system

was locked to locations where such content had been generated. Moving the system not only involved transporting and setting up infrastructure but also meant having to find a knowledgeable expert in the target area. This problem alone was more than enough to halt any mobility the system may have otherwise had.

The issue with relying on pre-authored content is succinctly stated by Hansen at al [83]:

> *While location-based information systems for, e.g., tourists, are well known, these systems usually rely on predefining all available information. This reliance on special purpose authored content is unfortunate, as it puts the onus of creating content on the maintainers of the system rather than on its users or the Web in general. Furthermore, users should certainly be allowed to add their own material and observations to a system.*

Hansen et al. realise that content creation is often the primary task in when introducing a new system and also hint that users of a system may not only aid in finding official content to add to the system but that their own input is of a unique and valuable kind.

Harter et al. demonstrated that systems that do not require any pre-authored content are certainly plausible. Indeed the earlier *Campus Aware* application [25], which extends the concept of *GeoNotes* [135], demonstrates a working mobile system in which no content need exist when the system is first started. *Campus Aware* is described by Burrell et al. [25]:

> *This tour guide allows members of the campus community to annotate physical space with knowledge and opinions. Prospective students can also annotate space with questions and thoughts as well as read comments made by those who are knowledgeable about the campus.*

Subsequent evaluation of *Campus Aware* reveals that content generated by users themselves is not necessarily of lower quality than content authored through more traditional methods by system designers or professionals of a subject area [25]:

> *Opening up a system to user contributions holds the promise of content that is much more informal, opinionated, and even more subversive than content provided by an official source... Notes contributed by an unofficial source such as students or other insiders were valued more than the official factual notes that were posted. The content became qualitatively unique and was well-received when it was created by other users. This evidence provides justification of opening system to this type of user feedback.*

Campus Aware demonstrates one possible method for allowing users to generate and contribute their own content to a mobile, peer-to-peer system. However, as will be discussed, there are several ways to generate content without relying on direct input by a designer and 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.

The problems with gathering and inputting content experienced in the *Lighthouse* resulted in it being an essential priority in *George Square* to avoid any set up which required content authoring. Instead *George Square* follows the guideline that, wherever possible, pre-authored content should be avoided in mobile systems. To achieve this *George Square* was implemented in a manner that allowed it to gather information as it ran in order to build up information about the area in which it was being used. As users conducted their visits, their locations, web pages accessed and photographs taken were all logged to the server. It was recommendations of this previously logged data that then provided content for subsequent visitors – both content generated purely by the user (for example, photographs) and content which was essentially recommended by users (for example, web pages). Furthermore, as the log of user activity was continually being extended with data through use, the information about the area remained relatively current as long as the system continued to be in use.

By creating a system that relied on content automatically generated through the use of the system itself, no authoring of any data was required by the system designers. Indeed, as the data was not delivered directly to users but rather filtered through a recommendation system, the quality of the information delivered to the users was of a generally high quality. Acceptances of recommendations (that is, visiting a recommended location or viewing a photograph) provided positive feedback to the recommendation algorithm, thus continually increasing the relevance of the delivered data. As *George Square* required no content authoring it could literally be deployed at any location and over time would begin to gather and deliver useful information to its users.

Although most of the visitors received a wealth of information from the system, the initial users would receive no recommendations, as there would be no logged data for the area. Whilst this did present a small problem it was far less significant than the problem of authoring a static set of content from the outset. The system did provide some value to initial users, as they were still able to share photographs, keep track of each other, discuss their visit over the VoIP connection and view Web pages. Thus, *George Square* provides functionality to initial users and this functionality, and quality of delivered content, grows over time or with the size of the community of users.

3.2.3.4 Modular design

The realisation that designing a system in which roles could easily be moved from one machine to another provided such a great benefit during later deployment and configuration that it led to the experimentation with a more modular and general componentised design process. Although this is discussed in far greater detail later in the thesis (Chapter 7), a brief introduction is warranted at this point.

EQUIP supports a powerful feature in that its tuple-spaces are capable of discovering and accessing other EQUIP tuple-spaces on the same network. Indeed, if security settings permit, any discovered tuple-spaces may be accessed using exactly the same protocols as if it were located on the local machine. The result of this is that the location of a tuple-space is completely abstracted away from an application to the point that, from the application's perspective, it is transparent whether the tuple-space is located locally or on another machine on the network.

It was decided early in the implementation of *George Square* that each client would run its own EQUIP server which would maintain two tuple-spaces – one public and one private. Rather than each system component being coded to directly communicate with other components, any data that was to be read or written would be passed through the tuple-spaces. For example, the component that is responsible for interacting with the USB camera and generating JPEG images writes the images into both the private and public tuple-spaces when the user clicks to take a photograph. Both the map component and the image-strip have hooks into the private tuple-space and monitor for any tuple being entered that matches the type JPEG. When this type is detected both these components read the tuple out and use it to display thumbnails on the interface. Similarly, when peers detect and connect to the public tuple-space they check for instances of JPEG images and read them out in order to display them on their local device as images from a co-visitor. Two tuple-spaces, a public one and a private one, were used simply to clearly separate information that may leave the device and information that would not. However, this was done purely to ease the design process and there is no technical reason that all information could not have been placed in a single tuple space on each device.

Although this technique of mediating messages through EQUIP tuple-spaces required a little extra work compared to directly passing information from one component to another, it provided the substantial benefit of encouraging and supporting components to be completely decoupled from one another. This, in turn, allowed for components to be easily moved, turned off or updated whilst the system was running. Furthermore, as the location of tuple-spaces is transparent to EQUIP applications, components could be moved between machines with no interruption to services. For example, during testing it was common to turn off a recommendation server on one machine and activate it on another to determine which provided the best performance. Similarly, providing the feature to switch user positioning between the component which read location from the GPS unit and the one which read it from manual clicking on the map as it was simply a case of deactivating one and activating the other. Finally, supporting multiple clients became a trivial task as, for example, displaying locations of multiple users on the map was handled automatically as EQUIP detected new user's public tuple-spaces and automatically read the user's locations out of them.

This ability to activate, deactivate and move components from one machine to another clearly maps directly to the concept of users temporarily assuming roles and changing roles in mobile systems. In a sense, the extremely late changes to the *George Square* configuration and setup that EQUIP facilitated

were design decisions to alter the roles users in the trials would assume. It is clear that if users themselves are to assume new roles then a system that supports different roles, and has an interface and tools that can adapt and change based on those roles, could be extremely beneficial.

Although EQUIP provides substantial support for allowing users in a peer-to-peer community to temporarily assume roles it does not provide a comprehensive solution. One of the main problems with EQUIP is that it does not permit a device to supply functionality to support a new role it has not previously been programmed with. Components can be activated or deactivated but there is no method for teaching a device to assume a new role once the system is active and there is no way of transferring a new component to support a new role to a device. Furthermore, EQUIP provides no method for determining which client devices are most suited for assuming a particular role themselves, or supporting a user in a particular role, or when they should do so. Instead, when using EQUIP it is the user that must decide when and which roles to assume. This can be a substantial difficulty, particularly if there are many roles available, as the user is unlikely to have knowledge of, or be proficient in, each and every component available.

These findings and the experience with EQUIP feed directly into the *Domino* system described in Chapter 7. *Domino* aims to address these issues by automatically detecting the actions a user takes and providing support to transfer and install new functionality that can support roles devices – facilitating role change at the system level.

3.2.4 New issues

Whilst it is clear that *George Square* offers significant improvements in mobility, primarily due to following the three guidelines identified from the *Lighthouse*, it still exhibits some severe problems. These are discussed here.

3.2.4.1 Sharing data generated during isolated use

Clients in *George Square* continued to operate while isolated from any other peer devices. Whilst isolated they continued to monitor the websites and locations the user visited, essentially identifying new information and creating new data logs that could potentially be of relevance and use to other users within the community. Although *George Square* allowed this behaviour and supported a smooth reconnection to a single peer or community of peers it failed to adequately spread the information generated back into the community.

When a device that had been isolated for a period of time encountered a peer it would transfer its logs to that peer, allowing the latter to benefit from the data gathered while the former had been isolated. However, the peer device receiving the updates would not store the information again in its own log. Thus, regardless of the number of peers who received the updates from the device that had been isolated, only that single device would ever contain, and therefore be able to share, that information. This has the result that unique information gathered when a peer is isolated is only available to peers if they can currently connect to that peer. This is illustrated in the diagram below.

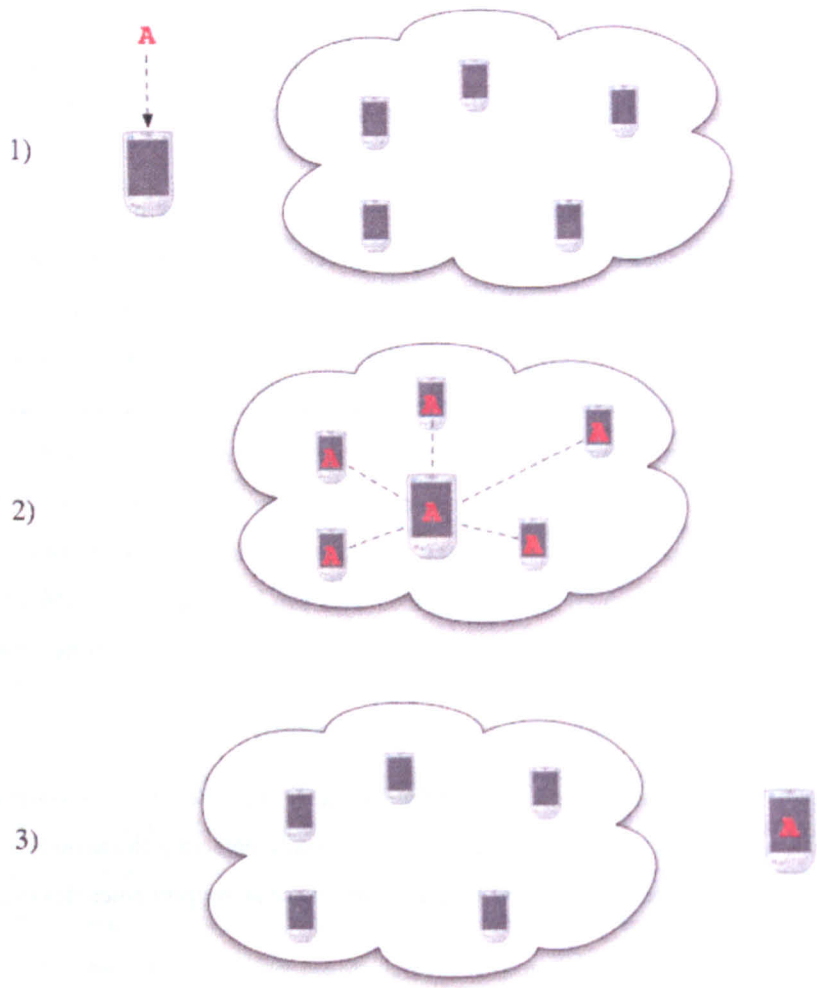

Figure 8: Diagram of one of the problems experienced in *George Square*. (1) While a device is separated from a community of peer devices it may still gather or generate some data *A*. (2) When the device rejoins the community the data or functionality *A* provides is available to all the peer devices. (3) However, when the device again leaves the community a copy or copies of *A* are not left in the community. Thus, the information and value of *A* is lost to the community.

This problem is likely unique to peer-to-peer communities and did not become apparent in the *Lighthouse* as that system was so static and pre-configured that it could not really be classed as a peer-to-peer system. Every device in the *Lighthouse* always viewed all the information within the system as they were always connected to the server.

However, this problem must be addressed and can be overcome by introducing another design guideline for mobile, peer-to-peer systems:

Allow information to be shared and stored by multiple peers

By sharing information with peers that then copy that information and store the copy themselves it can be made available to a greater number of carriers and spread far more rapidly throughout a community. In this way, *all* information be copied to *all* devices. Whilst that is one strategy for achieving rapid distribution it is often not efficient and can prove problematic. The issue is discussed further in Chapter 6. This guideline is later employed by the *Samara* system which implements a distributed

database for peer-to-peer devices. Its implementation is described in Chapter 6 and is itself subsequently used to drive the Castles game described in Chapter 7.

3.2.5 Re-experienced issues

The design and implementation of *George Square* began shortly after the trials of the *Lighthouse* had been completed. Thus, although some lessons were learned and improvements included in *George Square*, there was not enough time to address every issue. Three problems were, unfortunately, re-experienced – these are detailed here.

3.2.5.1 Centralisation

Whereas *Lighthouse* clients would completely fail if their connection to the server were interrupted, in *George Square* each client was able to run independently. Clients relied on querying a server to gain recommendations and to log their own data whilst all other functions were carried out locally or through direct communication with peers. For example, if a client lost its connection to the server but could still communicate with a peer, it would still be able to receive the location of the peer, see the recommendations the other user had displayed and view any photographs that were taken. However, no new recommendations for their own client would be received until access to the server was regained.

The problems of peer-to-peer systems relying on centralised architecture are obvious within the *George Square* system. Although the system does not crash and continues to provide some functionality whilst unable to connect to the server, the system is severely reduced and cannot usefully operate in this mode for any length of time. The information about recommendations and peer locations rapidly becomes stale and it was clear from the trials in which users left network range that the system quickly became of little value without the server. Users were essentially forced to re-enter network range to connect to the server in order to continue with their visit. This issue does not affect the smooth rejoining and integration back into the peer community of clients that have been isolated – it simply results in a frequent need to do so.

The failures of both the *Lighthouse* and *George Square* due to their centralised nature clearly draw attention to the fact that centralised architectures are not suitable for peer-to-peer systems.

3.2.5.2 Network

As discussed previously, the *Lighthouse* required an extremely specialised network environment. Relying on an isolated network not only required a considerable cost – both monetary and in set up time – but also resulted in users having no access to the Internet while using the system. *George Square* largely resolved most of the network requirements. By the time *George Square* was implemented, EQUIP's network protocols were far more robust and, coupled with the previously discussed ability for clients to disconnect and reconnect easily to the system, intermittent network connections or dropped packets were no longer problematic. Due to this, *George Square* did not require an isolated network and was able to run over standard networks connected to the Internet.

The tablet PCs carried by users in the city were capable of connecting to any wireless access point. This allowed them to communicate with peers and to freely access the Internet. Indeed, the system made use of how users browsed the Internet by tracking the web pages they visited and later recommending them to other visitors. Remote users' computers required a broadband Internet connection but there was no requirement for this to be provided over wired Ethernet or to be isolated from other network traffic.

In *George Square,* the type of network was significantly different from the *Lighthouse.* Whereas the *Lighthouse* system was in its own isolated subnet, the clients in *George Square* had full access to the Internet and the wealth of data it provided. Despite this significant change, neither setup proved particularly suitable for the mobile environment. In *George Square* the reliance on infrastructure networks resulted in peers that were not in range of any infrastructure, but were in range of one another, being unable to detect each other. This is extremely wasteful as peers encountering one another in the wild in this manner may benefit from the exchange of information, and such techniques have been successfully employed in many previous systems [62], [173], [7], [146]. Mobile devices experience a large number of networks and types of network, and to utilise these various types they must be able to switch intelligently between them—to use both infrastructure and ad hoc networks where appropriate.

One severe problem that remained was the need to pre-program each client with the IP address of other clients. Despite the fact that many of the clients operated on the same subnet, they were still incapable of detecting one another automatically and so had to be programmed with each other's IP addresses. This again draws attention to the need for an automated configuration process for the network setup as well as the need for a reliable method of discovering peers.

The experience of further network failings in *George Square* simply strengthens the need for a technique for intelligently and smoothly switching between networks. As has been previously stated, the design and implementation of a system that performs such a function is detailed in Chapter 4.

3.2.5.3 Location

At the time *George Square* was implemented there was only one viable positioning system to cover a city, GPS. Although GPS is designed for outdoor use and is aimed to have global coverage we found that it performed extremely poorly in the centre of Glasgow. When users started the system, the GPS often took a long period of time (in some cases as much as five minutes) to find enough satellites to locate the user. When a position was delivered it was often found to be inaccurate. Furthermore, users experienced frequent dropouts when the GPS failed to provide a position and there were certain regions of the square in which, throughout all the trials, a GPS location was simply never available.

The problems we experienced are, in fact, common to the use of GPS within cities and almost identical problems were experienced during GUIDE [36]. GPS performs poorly in any locations where a GPS

unit does not have a clear, unobscured line-of-sight to the sky (and hence the satellites that are in orbit). This problem is often compounded by multipath reflections, caused when satellite signals arrive later than expected, at multiple times and/or weaker, due to being reflected off buildings.

In our trials it was not uncommon for users to attempt to rely on the GPS position but to quickly abandon it in favour of the fall-back self-positioning we provided, which simply involved them clicking on the map to inform the system of their current location. This self-reporting of position is akin to that described by Benford et al, discussed earlier in the literature review [13]. Again, the failure of GPS, a system designed for outdoor use, highlights the finding from the *Lighthouse* and the literature review—that a single positioning technique is not suitable for mobile systems and instead a hybrid position technique should be employed.

As previously stated, a hybrid positioning system developed as part of the work for this thesis is described in Chapter 5.

3.2.6 Categorising mobile, peer-to-peer systems

In the literature review it was pointed out that Triantafillou et al. split peer-to-peer systems into three categories: centralised, hybrid and pure. Their work implies that pure peer-to-peer systems are ultimately the most mobile whilst centralised are the most static. It should be noted that a pure peer-to-peer architecture does not exclude the use of the traditional server-client model for transfers. As long as both peers can simultaneously act as servers and clients to each other then one cannot be said to be assuming a role the other is incapable of.

For example, in many recent mobile music applications, such as that described in [173], music exchange may occur simultaneously in both directions through traditional methods such as the FTP protocol. Here, peer *A* may be downloading a song from the FTP server running on peer *B* whilst *B* simultaneously downloads a song from *A*'s FTP server. As is clear, in this instance a server-client communication model is used but only from peer-to-peer; there is no centralised server for the client community in general. In short, a pure peer-to-peer architecture does not preclude the use of server-based services on individual clients.

Attempting to categorise both the *Lighthouse* and *George Square* according to these definitions proves interesting. The *Lighthouse* is quite clearly a centralised system. A peer in the *Lighthouse* has no direct communication with others and instead uploads messages and other information to a server, which peers then read from. If the server is unavailable then the majority of the functionality of the clients is lost and, in most cases, the client completely fails to respond to any user input. This may be one of the reasons that the *Lighthouse* proved to be quite inflexible in its mobility.

The *George Square* system proves harder to categorise, as it does not fit smoothly into one of these three definitions. Although every other service in *George Square* performed in a pure peer-to-peer manner, the substantial one of generating recommendations was not. The fact that a machine was

required to perform this specialised function clearly prohibits *George Square* from being classed as pure peer-to-peer. However, as this service could run on any of the participating peers it could be said that this role was temporarily assumed and this would suggest that the system was a hybrid one. Unfortunately, in order to ensure that the maximum amount of log data would be captured from the trials, the recommendation server was locked to one specific client machine that was always used by a remote visitor. Thus, although *George Square* is indeed a hybrid-capable peer-to-peer system, it was never actually trialled as one. Despite this, the benefits experienced from the fact that the majority of communication was achieved in a pure peer-to-peer manner and the fact that the recommendation server was implemented in a hybrid manner were apparent both in the end system and in designing the trials.

As with many large trials, during the latter stages of the design process many last minute changes were made. These frequently involved changing the number of participants or the type of machines that would be used. Due to the fact that the recommendation process was implemented in a hybrid manner there were no problems when moving this task from one machine to another. In a traditional centralised, server based model reassigning the roles the server fulfilled to another machine would often involve installing many support applications and require a substantial amount of time. However, as hybrid systems are designed to have services and roles continually migrating from one machine to another, it was a simple and almost instantaneous process in *George Square*.

After categorising both the systems using Triantfillou et al.'s categories, it is clear that they do not account for one of the most important features identified in this thesis – that of content authoring. As the *Lighthouse* demonstrated, authoring content for a system can be one of the most time consuming and expensive aspects of creating a mobile system. Furthermore, if the system is to remain up-to-date then content authoring can prove to be an ongoing commitment even after the system software is complete.

Surprisingly, although content (or the lack of content) can greatly impact a mobile system there is a distinct lack of literature on the subject. In order to advance the discussion it seems necessary to build on Triantafillou et al.'s categorisations and define some of the possible types of systems based on how they gather content. Therefore, four categorisations for defining how content is used in a mobile system are proposed here: *static*, *external*, *self-generating* and *reliance-free*.

A ***static*** content system is one in which content must be gathered and input into the system before it is operational. Once the content is input and the system started, the content cannot be changed and no new content can be inserted unless the system is shut down and rebuilt with new data. In short, all content must be input by one of the system's designers whilst the system is offline. This is the category the *Lighthouse* would fall into. In the *Lighthouse*, all the content that drove the experience was meticulously found, converted, aligned with existing data and input into the system by the researchers working on the system. Inputting the data was a complicated process understood by only a

few people and it was simply impossible even for them to alter any of the data the system used whilst it was in operation. Clearly, the fact that data must be gathered and input by a professional before a static content system can be used means it is possibly the worst choice for a mobile, peer-to-peer systems, as these can unexpectedly roam to any location.

An ***external*** content system is one in which content may continually change but in which a third party is responsible for updating or changing this information. Indeed, an external content system is itself not capable of altering the data at all. Instead, the system links to an external source from which it reads data. For example, an application that reads weather from a website such as weather.com and simply parses and displays it would be an external content system. In such an application, the client application itself has no way of changing the data and the developers of the client need not necessarily have generated, maintained or owned the original data. As any mobile system relying on external content would at least periodically require a connection to the external server holding the data, it is a poor choice for mobile systems.

In a ***self-generating*** content system information is entirely generated by the client applications themselves. This information may be shared to the community either through being uploaded to a server that peers then read from or spread directly through the community from one peer to another. *George Square* is a typical system that falls into this category. Recommendations are generated from the log of the system 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 systems as they can be configured to require absolutely no support, and connections to, any devices or servers outside the community.

Finally, a ***reliance-free*** content system has no dependency on any permanent store of content. A typical system that would fall into this category would be an instant messaging system. In this all data passed from one peer to another is typically generated and used once, and the delay between creation and consumption is minimal. In short, a *reliance-free* system has no requirement for a permanent store for any of the content it uses or generates. Reliance-free content systems should have no problems working in a mobile, peer-to-peer environment as they are likely to only require connections to peers that are currently in range.

Clearly, in order to avoid the considerable task of authoring content, either a self-generating or reliance-free content system must be employed. Static content systems require a committed effort from application developers during implementation and often require continued maintenance by them after the system is deployed. Whilst external content systems remove this responsibility from the developers of the application, it is simply moved into the hands of a third party. Obviously, having some of the content outside local control is often not desirable – particularly in critical applications.

Both self-generating and reliance-free 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. However, it may not prove feasible to create systems that fall into these categories in every domain. For example, any system that relies on officially sanctioned data, such as stock market prices, could not be self-generated on end-user devices.

Whilst it is impossible to create self-generating or reliance-free systems for all possible application domains, designers of mobile systems should aim to create such systems when feasible in order to negate the substantial cost and time involved in authoring content. Such systems also allow designers to avoid the requirement of a central server which often leads to the problems found in the *Lighthouse* and *George Square*. The diagram below (Figure 9) demonstrates where each of the categories described fall in relation to requiring a central server and requiring content to be authored.

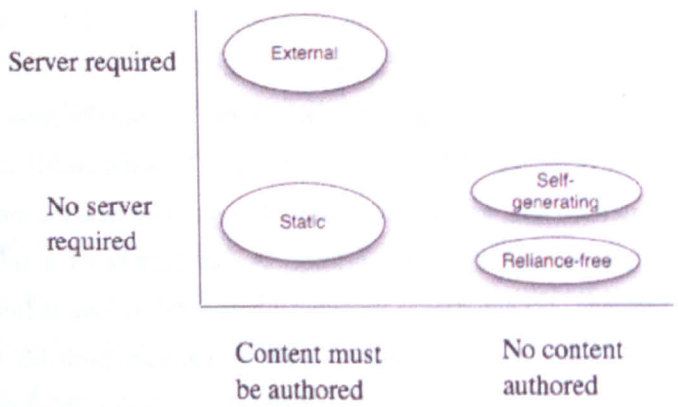

Figure 9: Overview of the design space relating to content types of *pure* peer-to-peer systems in the mobile environment.

As will be seen throughout the remainder of this thesis, the systems implemented from this point forward, with the exception of *Treasure*, attempt to avoid the requirements for content authoring and centralisation, and are either self-generating or reliance-free content systems.

By re-examining Triantafillou et al's categories alongside the content system categories, we can identify where each type combination may store the data they create and, subsequently, the types of network each may rely on.

Centralised peer-to-peer systems with a reliance on either static, external or self-generated content can easily rely on a central server to store and access data. As all clients must have access to this central server it is from here that data can be spread throughout a community.

Hybrid or pure peer-to-peer systems within reliance-free content systems do not require a permanent store and so need no server or external source to operate from. In short, systems falling into these categories do not generate any data that must be spread and stored throughout the entire community of users.

However, pure, and some hybrid, peer-to-peer systems which utilise self-generated content may need to spread this content throughout the community. This presents a problem, as there is no central store that all clients can access in order to share content through. Therefore, an alternative, reliable method of distributing content in such systems must be found. To spread their data, these systems may rely on epidemic network algorithms. These are discussed further in Chapter 6.

Figure 9 demonstrates that servers are not required for anything other than external content systems. However, this is only true if pure or hybrid architectures are followed. Table 1 provides a more complete overview of the design-space, comparing the new content types proposed here to Triantafillou et al's. peer-to-peer types.

	Static	External	Self-generating	Reliance-free
Pure	N	-	N	N
Hybrid	N	S	N	N
Centralised	S	S	S	-

Table 1: Possible peer-to-peer and content combinations. Items marked '-' are not valid combinations. Combinations marked 'S' require a server to operate whilst those marked 'N' do not require a server.

It is clear from Table 1 that pure and hybrid peer-to-peer systems have the advantage of negating the need for a server for the majority of content types. In contrast, centralised peer-to-peer systems can introduce the need for a server where one may not be required (static and self-generating). Therefore, with the exception of external content systems, all content types are achievable without the use of a central server. This leads to a final guideline extracted from the work on the *Lighthouse* and *George Square*.

Avoid centralised peer-to-peer architectures whenever possible

George Square, a hybrid peer-to-peer and self-generating content system, proved far more flexible and mobile than the *Lighthouse*, a static centralised system, yet both provide similar functionality (both are tourist co-visiting systems). It is clear from both the *Lighthouse* and *George Square* that whenever centralisation was used it repeatedly proved detrimental to the mobility and flexibility of both systems. The categorisation of content types and the combination with peer-to-peer types reveals that for most mobile systems centralisation can easily be avoided through a prudent selection of content-type.

In designing a mobile system, developers should seek to rely on static, self-generating and reliance-free content types in order to avoid centralisation. When utilising static or self-generating content architectures, designers should avoid the use of centralised peer-to-peer designs, as such systems can

be achieved through pure or hybrid architectures. The only mobile systems that absolutely require centralisation are those that rely on external content. Designers of such systems should strongly consider alternative content types where possible to avoid the problems centralisation often introduces.

3.3 Conclusions

Analysis of both the *Lighthouse* and *George Square* systems has proved extremely beneficial in revealing problems with existing mobile systems, developing guidelines for the future design of mobile systems and identifying infrastructure that is simply lacking in current mobile systems.

From the investigation five new and important guidelines for developing mobile systems have been identified:

- Support users changing roles
- Support fluid group dynamics
- Avoid reliance on pre-authored content
- Allow information to be shared and stored by multiple peers
- Avoid centralised peer-to-peer architectures whenever possible

For each of these guidelines, information on how they can be applied has been provided and the first three have been demonstrated to be useful as part of the *George Square* system. All of these guidelines are followed by the systems subsequently described in this thesis and their utility is reinforced by their successful application within these systems.

In addition, the importance of avoiding pre-authored content in mobile systems has been highlighted and categorisations of the possible types of content system have been given. Again, the *George Square* system demonstrates how a complex and successful system that avoids a reliance on pre-authored content can be implemented.

Four important pieces of infrastructure missing in mobile, peer-to-peer development have been identified:

- A mechanism for intelligently selecting which networks to use and for reliably discovering peers on these networks
- A hybrid positioning system that requires no initial setup and yet has high availability in providing location both indoors and outdoors
- A method for providing and distributing data within a peer-to-peer community
- A method for adapting the system itself around user's activities

Whilst the remainder of the work in this thesis does follow the guidelines laid out in this chapter, the primary aim from this point on is the investigation and implementation of the four missing pieces of

infrastructure identified as important to mobile systems. Each one is subsequently demonstrated in a successful mobile system. It is hoped that the identification of the guidelines and the implementation of the missing infrastructure will advance the mobile field by facilitating the future design and development of mobile applications.

4 A NETWORK DRIVER FOR MOBILE, PEER-TO-PEER SYSTEMS

During the literature review it became clear that there was a need for a network driver that can fulfil the specific needs of mobile, peer-to-peer applications. Specifically, the requirement is to allow the discovery and communication with the highest number of peers possible. This need is reinforced by both the *Lighthouse* and *George Square* trials that suffered different but severe problems with the network technologies they employed.

Current network drivers and technologies frequently fail when used in mobile, peer-to-peer environments. As has been experienced, they often require configuration by trial designers or by users before they can be used in a particular network setup. They often disconnect and fail to rejoin networks without input from the user. In short, they fail to be autonomous enough to intelligently join and configure devices for use in networks without requiring user input. As they fail to join networks they in turn fail to locate and identify peers. This is a serious problem as it has been seen that the discovery of peers is crucial in driving any peer-to-peer system.

With this in mind, several goals for a mobile, peer-to-peer network driver can be stated:

- Require no input from the user
- Automatically select and join networks where peers are most likely to be found
- Automatically configure the device for use after a network is joined
- Continue to allow users to perform standard tasks (e.g. check email, browse the Web)
- Advertise the device's own existence on a network to allow peers to find it
- Discover peers once on a network

To address these issues, the development of a new network driver was undertaken and this work is detailed in the following sections. Firstly, an overview of the current communication technologies available on mobile devices is presented; this includes considerations of legal concerns and a trial aimed at discovering which technology best supports peer-to-peer discovery and communication. Following this, one of the technologies is selected and a driver is then implemented. The driver consists of two main segments and each is tested within a mobile system. The implementation of each segment and the subsequent trial within a system are described.

During the course of discussing the design and implementation of the wireless driver, the concept of *Seamful Design* is also investigated. Experimentation with the use of seamful design in the mobile systems trialled in this section leads to a final design guideline for mobile, peer-to-peer systems. This is discussed further in section 4.2.2 of this chapter.

4.1 Selection of an underlying technology

In creating mobile systems there are many constraints, such as legal issues, which are outside a developer's control and cannot be directly overcome through software design. Although developers often try to mitigate such problems as best they can, there are frequently instances when the solution is simply too costly or risky, and functionality must be dropped. A particularly relevant issue for mobile applications is the use of wireless communication, as this is generally required if any communication is to occur between multiple users' devices in peer-to-peer environments. The choice of which form of wireless communication to use in a mobile application can be of the utmost importance to its subsequent performance, as it can greatly affect the number of peers that can potentially be encountered, the rate data can spread within a peer-to-peer community, and the types of information that can be transferred and accessed. In addition, legal issues can simply exclude some forms of wireless from being used in certain communications. For example, one of the major issues in the *Yoshii* game (discussed later in section 4.3.3) was that it was thought too risky, from a legal standpoint, to use *any* discovered access points to transfer data over in order to synchronise scores with an Internet-based server.

Although there are new wireless protocols emerging, such as Wireless USB and WiMAX, there are currently only three that are popular on mobile devices; 802.11, GPRS/GSM and Bluetooth. This section examines these three different wireless technologies and presents results from trials in order to compare them.

4.1.1 802.11

802.11 wireless is probably the most popular choice for communication in peer-to-peer applications on PDAs for several reasons. It has relatively high bandwidth, a large range, little protocol overhead in terms of setup, and is built-in on nearly all modern PDAs. 802.11 wireless cards are also beginning to appear built-in on phones. Although there are currently three widely used 802.11 protocols (a, b and g), mobile devices predominately use only 802.11b, which provides a maximum transfer rate of 11Mbps. As this is the only form of 802.11 available for most mobile devices, it is the only one included in the trials and comparisons are described later in this section.

Wireless card power consumption is not normally detailed in the technical specification of most mobile devices but will normally range from 1-4W during normal use. This is relatively high compared to both Bluetooth and GPRS, and despite recent attempts to improve the power efficiency [51], is unlikely to change in the near future.

Despite its many advantages, use of 802.11 can still prove problematic due to the many legal concerns surrounding its use. The primary problem with 802.11 is not that there are specific laws limiting its use. Rather, the problem is that there are too few laws (at least in the UK and the USA) directly addressing issues with 802.11. This results in overly cautious design in many applications utilising 802.11, as developers are simply unaware of how they may and may not use 802.11. Inevitably,

particularly in commercial applications, they opt to err on the side of caution and avoid 802.11 whenever possible.

There are few or no legal issues concerning ad hoc use of 802.11, in which devices communicate directly to one another rather than involving a third-party. Similarly, as discovery of 802.11 access points is a completely passive process, there are no known legal concerns here either. Furthermore, as the vast majority of access points are owned and operated by individuals rather than commercial or institutional bodies, there are no concerns that mapping the location of access points in any way constitutes IP theft or a security risk. However, despite the fact that 802.11 access points are now extremely common—it is typical for hundreds to be in range in streets at the centre of large western cities—the use of access points that are not one's own remains a legal grey area. Although the work in this thesis leads to advocating peer-to-peer communication for mobile applications in general, the ability to use discovered access points is important for many travellers as they provide a convenient, and often free, connection to the Internet. Given that users desire to connect to access points, be they their own or others', it would be advantageous to allow peer-to-peer applications to continue to discover and communicate with peers whilst connected to an infrastructure mode network. Indeed, this is one of the aims of the network driver described here—to allow mobile devices to continue to run peer-to-peer applications regardless of the type of network they are using.

Obviously, to use 802.11 access points that are randomly discovered while a device moves within a city the access points themselves must be left open. Although the American federal government at one point believed open access points to be a threat to national security [17] and some American counties are considering legally forcing businesses to secure their access points [4], there is still a substantial number of open access points to be found. Indeed, the majority, 67%, of access points still remain open and this is likely to increase to 80% by 2007 [79].

Most open access points are operated by home users who simply have no concern about leaving it open, do not have the knowledge to enable security on them (nearly all access points have security disabled by default) or deliberately leave them open as they do indeed intend for their connection to be publicly shared. Deliberately sharing a connection through an access point is not necessarily illegal, nor is it necessarily a security risk if sensible measures are taken to encrypt or secure data transmitted inside the network. Indeed, in choosing between securing network traffic and securing an access point it may be wiser to do the former as both WEP and WPA, the most common forms of 802.11 security, are extremely easy to crack – there are even applications which automate the process[6]. Although many ISPs have clauses in their contracts that prohibit anyone but the contract holder from using the connection, others actively encourage it. For example, SpeakEasy state in their policy document[7]:

[6] http://airsnort.shmoo.com

[7] http://www.speakeasy.net/netshare/terms/#Wi-Fipolicy

> *Wireless networking and publicly shared wireless networks present exciting new opportunities to share information and connectivity resources with one another - we encourage you to explore it!*

Despite the fact that many access point owners do desire their connection to be publicly available, there is no concrete way to differentiate these from owners who do not wish their access points to be used other than asking the owner him or herself. Similarly, although many cities are starting to provide free wireless access, there is no definitive technique for automating the process of determining which access points belong to these networks, keeping any list of open access points up-to-date, or filtering out access points which by coincidence have the same SSIDs as ones which are provided freely.

Given that there are so many open access points, and that their use could be beneficial for mobile applications that rely on external data, there is a strong inclination to write applications that can make use of them. However, the main legal concern in using any of the open access points that are discovered is that doing so may constitute theft if the owner does not wish to share his or her connection. By connecting to the Internet through an open access point, one increases the bandwidth used by whoever owns the access point. If their ISP charges them on a usage basis then clearly the owner of the access point may incur additional fees. To date, only one legal case has arisen in which a man using an open access point was charged with theft, and no legal precedent has been set[8]. Hale states [79]:

> *This panoply of case law provides fairly broad (and potentially confusing) latitude to courts in determining whether unauthorized access has occurred in the case where defendant piggy-backs off of another's WLAN.*

Despite this apparent legal confusion, the general consensus among users seems to be one which Hale also states, but one which he does not highlight as legally proven. That is [79]:

> *...[the] absence of password protection, or a similar failure to take reasonable safeguards against unauthorized use, such as encryption, may rebut the view that any outside access to a private WLAN constitutes unauthorized access.*

Kerr also arrives at this conclusion stating that [95]:

> *Here the code-based restriction presumably includes the network's encryption scheme. Under this approach, access is without authorization only if the user bypasses the wireless network's encryption scheme. If the hospital left the network open and unencrypted, however, use would not be circumventing a code-based restriction and could not be without authorization.*

[8] http://www.msnbc.msn.com/id/8489534/

Although this may be strong enough support to convince individuals to connect to and use open access points to access the Internet, the fact that the legality is not proven means that it is not yet advisable for developers to create applications which will automatically implement this behaviour. Whilst individual use of single open access point is likely not to be noticed and even less likely to appear in court, an application which automated the connection to an access point and then utilised it to connect to the Internet could potentially be used by many – thus making it easily noticeable and potentially a target for legal attack. Instead, it may be safer to allow detection, connection and non-Internet use of open access points; such as accessing local clients, printers, or other services on the local subnet. This avoids many of the legal issues concerning theft since, as no paid service is utilised, the owner incurs no financial loss. Hale concludes similarly, stating [79]:

> *If a Wi-Fi interloper must continue, he or she should avoid heavy downloading activity (music, games, movies, etc.) that has a tendency to overburden the network and may amount to recoverable damages.*

All mobile developers should have knowledge of these legal concerns and should be fully aware of how far their applications push them. Whilst there are no insurmountable hardware or software problems with using 802.11 for direct peer-to-peer communication or detection of wireless access points, developers must be conscious that connecting to and using Internet connections through open access points remains a legal grey area. Connecting to and using open access points as a conduit for peer-to-peer communication is likely to be legal, and therefore may be considered as acceptable to implement within applications. Developers should certainly not avoid 802.11 completely, as 802.11 is perhaps the most suitable communication platform for peer-to-peer applications.

4.1.2 Bluetooth

Bluetooth is the most common form of wireless communication technology on mobile devices and is found on both modern phones and PDAs (for example, the iMate SP5 phone and the iPAQ 4150 PDA). Although it is possible to use Bluetooth to connect to the Internet by connecting through another device, such as a mobile phone using GSM or a laptop computer using 802.11, this is extremely rare. There are no Bluetooth infrastructure points that provide Internet access and thus in practice Bluetooth is primarily used only for connecting devices directly to others in a peer-to-peer fashion. Clearly, this leads to an immediate constraint, as Bluetooth is simply not an option in cases where mobile devices, which do not have another wireless communication technology, require Internet access.

Bluetooth is capable of speeds up to 2.1Mbps, and uses between 1 to 100mW depending on the class of Bluetooth used. There are three classes of Bluetooth the ranges and power consumption rates listen in Table 2.

Class	Power Consumption (mW)	Range (m)
1	100	100
2	2.5	10
3	1	1

Table 2: Bluetooth classes, power consumption and range.

As the overwhelming majority of Bluetooth devices are sold with Class 2 devices built-in—indeed, the author has been unable to locate any PDAs or phones with Class 1 or 3 Bluetooth currently available—we shall concentrate only on this one class. External cards providing Class 1 Bluetooth are available for many PDAs but, as Class 2 is sufficient for most users' range requirements when using most commercially available Bluetooth devices, and as the addition of an external card can make a device's form unwieldy, the use of external cards remains extremely rare.

4.1.3 GSM/GPRS

GPRS (General Packet Radio Service) is a data service which runs on top of GSM (Global System for Mobile communications) and is available on virtually all modern phones. Most mobile phone network providers make GPRS available for data transfers to customers. When actively using GPRS the phone network assigns the phone an IP address and allows full Internet access. Standard GPRS is capable of data rates up to 160kbps although EGPRS (Enhanced GPRS) increases this to 236.8kbps. In practice, the data rate achieved is generally much lower as bandwidth has to be shared with other users in the same phone cell. Also, interference from other phones, even those not transferring data, can significantly impact reception. In addition, GPRS rates are affected by the range from the device to the cell tower and, as average users are not aware and have no means of discovering where a cell tower is, they have no information available on how to decrease this range in order to achieve higher data rates.

The main disadvantage of GPRS for use in mobile, peer-to-peer applications is that the connection method, a direct connection to the Internet, is detrimental to the behaviour of peer-to-peer applications. As there is no local subnet, an application relying on GPRS is unable to discover nearby peers, even those within the same phone cell, and thus is unable to utilise any pure peer-to-peer techniques. In short, devices relying solely on GPRS have no choice but to rely on a central Internet server if they wish to discover peers. Other notable disadvantages are high cost (in Britain charges average approximately £3 per megabyte of data transferred) and relatively slow connection rate.

In addition to the technical disadvantages of GPRS on the GSM network, the mobile phone operators, who pay for the construction and placement of the cell towers, are extremely protective of any potential profit that may be made through the use of mobile phones or cell towers. If the locations of cell towers are known it is possible to utilise the information about what cell towers are in range and the signal strengths to each to provide a form of positioning through triangulation. As the mobile phone network

providers wish to profit from this information themselves, they actively seek to quash attempts to use information about cell tower locations to locate users. They do this in two main ways. Firstly, by encouraging mobile phone manufacturers to block access to the information about cell towers that a phone detects. For example, mobile phones must be constantly aware of what cell towers are in range and the signal strengths to each in order to handle hand-offs from one cell to another adequately and for simpler tasks such as displaying the current signal strength to the user. However, the required API information is not made publicly available so as developers might use it in their own applications, and thus laborious work probing memory must be completed for each phone operating system and driver in order to determine where this information is stored and how it can be interpreted.

Secondly, mobile phone network providers claim the location, layout and identification numbers of their cell towers are copyrighted information. Recently, in Austria, an individual was prosecuted for writing an application that used this information [98]. By writing an application that detected cells and recorded their unique identifiers and locations he was able to record cell towers and their locations as he drove through large sections of Austria. Subsequently, he was able to write another application that then used this information to find a user's location and provide some basic location-based services. Unfortunately, the network provider whose cell tower information he was using, T-Mobile, decided to prosecute for this use and successfully managed to stop the application from being used.

For the mobile developer not working for the phone network companies, the fact that cell tower location information is aggressively guarded by these companies greatly reduces the range of mobile applications that can be made on phones. For example, any applications that reveal information about cell towers or their IDs are simply not permissible. This means that applications we have worked on, such as *Treasure* and *Feeding Yoshi* would not be possible using GSM networks. This situation is unfortunate as many of the users of these applications, as well as other researchers, have specifically shown an interest in a version which would work on phones. Similarly, games such as Node Runner[9] may not be possible if the nodes being sought were cell towers instead of 802.11 access points.

As GPRS on GSM networks is incapable of supporting pure peer-to-peer applications and since the phone networks discourage use of cell information, GPRS is not a candidate for most mobile applications. Whilst the network characteristics may be of interest for centralised applications, they are not relevant for this thesis and, in any event, are widely published already. Therefore, GPRS is not included in the comparison of possible peer-to-peer wireless technologies and instead only 802.11 and Bluetooth are examined.

4.1.4 Trial

In order to investigate which transmission techniques are most suitable for pure peer-to-peer mobile, two specific aspects must be considered:

[9] http://www.noderunner.org

- the time it takes to discover peers
- the amount of data that may be transferred during peer encounters

Firstly, consider the time taken to discover peers and the reliability of doing so. This is obviously important for pure peer-to-peer applications in which maximising the number of peers encountered is critical to maintaining both enough information to drive the application and to keeping the information up-to-date. Secondly, consider the amount of data that can be transferred during encounters at different distances. Again, this is critical for similar reasons—the amount of data transferred during encounters can directly affect the performance of any mobile application that relies solely on peer-to-peer communication. Furthermore, the distances are important as when a device is in a mobile environment randomly encountering peers, encounters will occur with a different proximity and this information can be vital to discovering if an application is viable. For example, if a mobile application were to be used to transfer files of 500kB between peers wandering around a museum it may not be a feasible one if the average time and distance between devices during encounters resulted in less than 500kB being transferred.

As stated, out of the three wireless technologies common on modern devices, only 802.11 and Bluetooth were considered suitable for use in pure peer-to-peer mobile environments. Furthermore, only one specific version of each was examined – 802.11b and Class 2 Bluetooth. Two separate trials were conducted to record results of the two topics of interest – the time for peer discovery to occur and transfer rates over successful connections.

In the first trial, examining peer discovery and reliability, both 802.11 and Bluetooth were tested at different ranges: 1, 5, 10, 20, 50 and 100 metres. The Class 2 Bluetooth being used has a range of 10 metres so was not expected to work for any distances above this but was tested for completeness. 802.11 devices were tested both in ad hoc and infrastructure mode. The primary interest is in ad hoc use as this is the most likely form used in peer-to-peer mobile applications but again, for completeness, tests using an infrastructure access point were conducted. For the Bluetooth and ad hoc 802.11 tests, the devices were simply moved the required distance from each other before the tests were run (top of Figure 10). In the infrastructure 802.11 test, the relevant distance was from the access point so the two devices were moved the required distance in opposing directions from the access point (bottom of Figure 10). For all tests care was taken to ensure that there were no obstructions between the devices. This includes the bodies of the organisers conducting the tests, the organisers were careful to ensure they were all behind the devices when trials were being run.

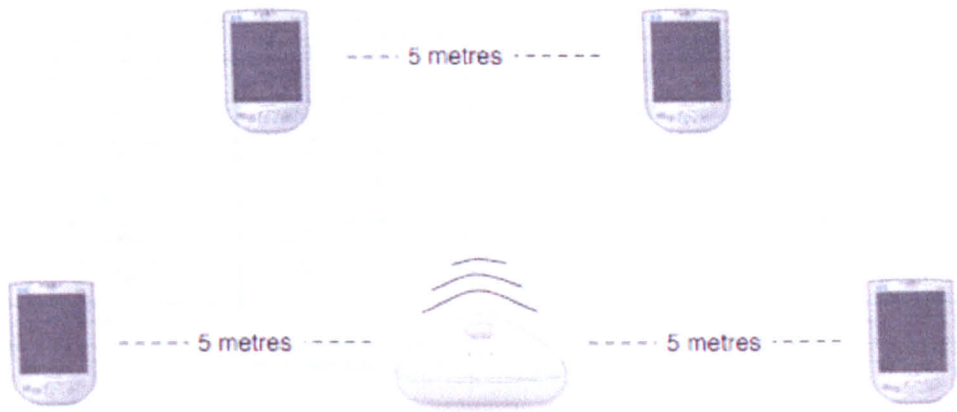

Figure 10: 802.11 and Bluetooth trial device setup. In both the ad hoc 802.11 and Bluetooth tests, the devices were connected directly at the required distance apart with a clear line of sight to each other. For the infrastructure 802.11 tests, the devices were connected to an 802.11 access point, the required distance from the access point and with a clear line of sight to the access point.

In order to conduct and record the necessary timings, applications were written for both 802.11 and Bluetooth that continually seek out peers. In the case of Bluetooth this simply involved standard calls to the Bluetooth stack that already includes discovery methods. However, for 802.11 SDS (Self Discovering Spaces, explained in more detail in section 4.3.2) was used as there is no discovery mechanism included in its network stack. In order to improve accuracy, the SDS broadcast rate was increased to 10ms rather than the default of 1s. The default is set at this length so as not to flood the network under normal conditions but this was not an issue for the tests as they were conducted on isolated networks. Thus, the 10ms rate obviously allows for results to be accurate to within that interval. The 802.11 network mode, IP address and SSID were locked throughout the trials and not allowed to jump from one network type or SSID to another, so as to eliminate the underlying network setup times from interfering with the trial results.

At each of the 6 distances, the peer discovery was attempted 20 times and the time to discover peers, in cases where it was successful, was noted. If devices failed to discover one another within 60 seconds, the attempt was recorded as unsuccessful. Whilst results will vary slightly depending on many external factors we did not measure, such as moisture in the air or air pressure, it is hoped the recorded results are representative of the technologies' general profiles. The number of individual broadcasts and discovery attempts made was also ignored. This was done because it is not deemed worthwhile to compare these values as 802.11 and Bluetooth discovery protocols are substantially different. For example, such a comparison attempted between SDS's 10ms broadcast rate and Bluetooth's long scan time, which can be longer than 10 seconds, would undoubtedly lead to an extremely low discovery *ratio* for 802.11, which does not accurately reflect the overall detection rate during peer encounters. The results from the first trial are shown in Table 3.

	Bluetooth		Ad hoc 802.11		Infrastructure 802.11	
Range (m)	Avg time (ms)	% succ	Avg time (ms)	% succ	Avg time (ms)	% succ
1	4256.7	80	9.9	100	52.2	100
5	4247.3	100	9.0	100	45.1	100
10	15081.6	55	9.1	100	51.2	100
20	N/A	0	10.1	100	66.8	100
50	N/A	0	11.2	100	167.9	100
100	N/A	0	14.85	100	N/A	0

Table 3: Peer discovery times and success rates

It is clear from the results that both ad hoc and infrastructure 802.11 are far superior to Bluetooth when it comes to peer discovery. Even within the 10 metres that the Bluetooth protocol specifies that Class 2 devices should operate, there was a clear sensitivity to range. It is commonly known that Bluetooth devices are not reliable when they are either too close or too far, and this is borne out in the results of the trial which show that discovery was only 80% reliable at 1 metre distance and 55% reliable at 10 metres. Furthermore, the time for peers to discover one another was substantially higher between 5 and 10 metres, more than tripling from just over 4 seconds to over 15. This is a serious defect for peer discovery in mobile environments as devices are rarely within 10 metres of one another for 15 seconds unless owners are travelling as a group or are stationary.

802.11, however, provides extremely reliable peer discovery from 1 to 100 metres. In ad hoc mode, range within the 100 metre limit seems to affect discovery rates little, and all averages are below 50 milliseconds. However, in infrastructure mode the range seems to have a stronger affect and it is clear at 50 metres there is a marked increase in the time taken for discovery to occur and at the 100 metre limit there are no successful discoveries whatsoever. This difference from ad hoc mode may be due to the aerial on the access point itself being less powerful, or more obscured, than on the devices. An Apple AirPort was used as the infrastructure access point and the casing that surrounds the internal aerial may have interfered with the signal. Tests with another type of access point, preferably with an external aerial, would have to be conducted to confirm this theory. Despite this effect experienced at the maximum range in infrastructure mode, 802.11 proved far superior to Bluetooth. Even at its maximum 100 metre range, discovery took place in the worst case in under 1.5 seconds compared to over 4 seconds in the best instance for Bluetooth at 5 metres (the distance it performed most efficiently).

The second trial was conducted using a similar method as the first. The devices were again separated at the same ranges, but in this trial the time taken to transfer 1MB of data between the devices *after* they had successfully detected one another was recorded. This was repeated five times at each of the ranges and the average transfer rate calculated. Obviously, despite many attempts to connect devices outside the maximum ranges specified, connections were not possible at some of the higher distances. In these cases results could not be recorded. The results from the second trial are presented in Table 4.

Range (m)	Bluetooth avg time (ms)	Ad hoc 802.11 avg time (ms)	Infrastructure 802.11 avg time (ms)
1	19063.4	2589.7	4260.2
5	20459.2	2606.6	4233.8
10	67024.1	2637.3	4332.8
25	N/A	2769.1	4303.9
50	N/A	2757.1	4571.0
100	N/A	4526.7	N/A

Table 4: Transfer times between peers exchanging 1MB of data

At 25 metres and over, Bluetooth was unable to complete any transfers whatsoever. Similarly, when in infrastructure mode 802.11, it was unable to complete any transfers at 100 metres. Again, both ad hoc and infrastructure 802.11 prove vastly superior to Bluetooth. Bluetooth experienced slow transfers of around 20 seconds at both 1 and 5 metres and a remarkably poor transfer rate, taking over a minute to transfer 1MB, at 10 metres. This is particularly surprising as 10 metres is within the operating range specified for Bluetooth[10]:

> *Class 2 radios – most commonly found in mobile devices – have a range of 10 metres or 30 feet*

Therefore, such a substantial drop-off in transfer rate was simply not expected within the operating range. 802.11 performs far better in transfer rates with 1MB being sent in under 3 seconds for distances of 50 metres and below in ad hoc mode. In infrastructure mode, transmission times are just slightly under double the corresponding ad hoc times. This is most likely not because the total travel distance is doubled (the devices are in opposite directions of the access points) but due to packet collisions and subsequent traffic management in the access point firmware. The access point must receive data from one device and retransmit it in order to reach the destination device. As all wireless traffic is broadcast, this results in a busy network environment in which the access point is receiving data and sending acknowledgements to the sending device and sending data and receiving acknowledgements from the receiver device. Again, despite this it is clear that 802.11 outperforms Bluetooth both within its operating range and far beyond.

Complete results from the trials are available in the appendix.

4.1.5 Conclusions

Whilst neither Bluetooth or 802.11 can be said to be the most appropriate technology for all mobile peer-to-peer communication, it does appear that 802.11 is generally more effective than Bluetooth. This is mainly because 802.11 works over a larger range, and is substantially quicker and more reliable in the very thing that is fundamental to all peer-to-peer applications, that of detecting peers.

[10] http://www.bluetooth.com/Bluetooth/Learn/Basics

Although the use of 802.11 does evoke the legal problems discussed, none of these in any way affect peer-to-peer communication itself. Thus, developers using 802.11 can design for standard peer-to-peer ad hoc use without any concerns and, if they desire, additionally allow for peer discovery over infrastructure networks. It is only if the application requires Internet access that the developer should be hesitant about using randomly discovered open access points. As one of the findings thus far has been that peer-to-peer mobile applications should generally avoid reliance on a central server, this should not be an issue for the majority of such applications.

802.11 has clear advantages over Bluetooth for peer-to-peer mobile applications. Its extremely short discovery time, greater operating range and relatively rapid transfer rate adequately meet the need to identify the largest number of possible peers and quickly spread information throughout a community of users.

Given that 802.11 does appear to provide so many benefits for mobile applications, it can appear surprising that a large number of applications do employ Bluetooth rather than 802.11 on devices where both are available. The most likely reason for this is that Bluetooth does have a single but significant advantage over 802.11 in its power consumption that can literally be a thousand times lower (2.5mW compared to 1-4W). However, this advantage may soon be inconsequential as modern devices, such as the i-Mate SP5, are now capable of running for an entire day whilst using inbuilt 802.11. Traditionally, the lack of a device lasting an entire day has been the barrier inhibiting its wider use.

Future versions of Bluetooth are likely to use UWB (ultra-wideband), which may provide greatly increased bandwidth, significantly reducing peer discovery times and increasing bandwidth. However, this is unlikely to be implemented in the near future and will not significantly affect the operating range of Bluetooth devices. As higher operating ranges greatly increase the chances of discovering and communicating with peers, it is possible that 802.11 will remain a preferable choice even when faster UWB Bluetooth becomes available.

4.2 The wireless driver

The need for the network driver and the decision to create it using 802.11 directly resulted from the work in this thesis – specifically as a result of the problems encountered in *George Square*. However, the decision to implement it was made in conjunction with Malcolm Hall, and both he and myself worked equally on the implementation of the first part of the driver described in this section.

The implemented driver runs on PocketPC and Windows Mobile devices. The first part of the driver essentially allows the default driver on the mobile device to be bypassed, and for our driver to scan for and select which networks to join. The standard drivers provided as part of the PocketPC and Windows Mobile operating systems are not suitable for a number of reasons. Firstly, they continually request the operating system to ask the user about network connection decisions through a pop-up

balloon. This is not only annoying for users but also requires them to have knowledge of various connection issues such as IP addresses and wireless security. Furthermore, as there is no automatic connection to networks, there can be substantial periods when the device simply remains disconnected from all networks whilst it waits on user input. If a device is carried in a user's pocket then these network notifications will not be seen and, as a result, the device may remain disconnected for the entire duration of a user's travels. This is a clear waste of battery power and of opportunities to meet peers—the network card is on and there are networks it could connect to in order to search for peers yet instead the card remains idle. Finally, the standard PPC/WM wireless driver cannot easily be controlled programmatically, meaning that these continual interruptions to the user and the lack of automated connection cannot be overcome using this driver.

Most of the work on the driver is implemented in C#. Indeed, as will become apparent, most of the mobile systems from Glasgow that are discussed in this thesis have been implemented in C#. Throughout work on many of the systems discussed in this thesis there has been a gradual switch from using Java to C# for implementing mobile applications. Although there are many reasons for this, the main ones are: C# IDE tools are more tightly integrated with development on the main mobile operating systems (PocketPC and Windows Mobile); performance of compiled C# executables is substantially faster than Java executables, particularly in thread allocation and graphics performance; and native API functionality, which is commonly required on mobile platforms to access their hardware, is easier to invoke from C#. Janecek and Hlavacs arrive at similar conclusions in their study comparing both languages for use in mobile development [92].

The driver we created is mainly implemented in C# but certain parts of code which interface with the NDIS[11] APIs on the device are written in C++. The driver deactivates the standard PPC or WM driver and also, optionally, disables the network notification bubbles that can prove distracting to the user. Once the default driver is disabled, our driver allows card power to be turned on and off, scans for 802.11 networks to be conducted, connection mode to be switched between ad hoc and infrastructure, networks to be joined and IP addresses to be automatically configured. In short, full control of the wireless functions on the device are made available by our driver through a simple set of API methods.

This bypassing of the default driver to allow full control of wireless functionality is the first part of the wireless driver. The second part, which adds some default logic to control which networks are joined and the ability to discover peers, is described in the subsequent section in this chapter. However, the first part was tested within the *Treasure* game, which is described in the next section.

4.2.1 Treasure

The *Treasure* game was created for two reasons: to test the wireless driver and to investigate Chalmers' concept of *Seamful Design*. This concept relates back to the findings of Flintham et al. described in the literature review – by revealing details of infrastructure to users they can be empowered to find novel

[11] Network Driver Interface Specification (see http://www.ndis.com/ for more details)

ways to use the infrastructure and to overcome problems better when the infrastructure fails. As the concept of *Seamful Design* employed in *Treasure* turns out to be highly applicable to mobile systems more generally, it is discussed after the description of *Treasure* and before further discussion of the wireless driver.

The idea and game concept behind *Treasure* was my own—inspired by Matthew Chalmer's desire for a Seamful Design testbed. The overwhelming majority of the implementation was done by Malcolm Hall and myself, although several undergraduate students did contribute small pieces of code to the game client as part of a summer research project.

To take part in the game, players are split into two teams and carry a PDA (iPAQ 5550) augmented with GPS capability. The interface shows a map of the area onto which coins periodically appear (Figure 11). Players can collect these randomly appearing coins by physically moving over the apparent location of the coin. When users feel they have collected enough coins, they can exchange them for points by clicking a button marked 'upload'. The coins that appear are placed randomly in the game area and so can be *outside* areas of 802.11 coverage, but to exchange the coins for points the players must be *inside* network coverage. If they are not in network coverage, or if they are in an area of particularly poor network coverage, the coins will simply be dropped and no points gained when the upload button is clicked. Furthermore, new coins only appear on a player's map whilst the player is within network coverage. This concept of having apparently physically located objects that are only viewable on digital devices whilst not viewable in the physical world is common to mobile games, and appears in *Six in the City* [118] and Benford et al.'s *Unearthing Virtual History* system [12].

As the game progresses, the map the players view is continually updated with an overlay of the network strength in the area – sampled by players' devices. In addition, the current 802.11 signal strength is always displayed as a bar graph at the top of the screen. Thus, as the game progresses, players learn where areas of strong and poor 802.11 signal strength are, and learn that to collect points reliably they must be in an area of high signal strength. Players are also shown information about the GPS infrastructure. Their own, and other players' locations are always displayed on the map, and the GPS accuracy can be inferred by a coloured number that shows the quality of the GPS fix and the number of satellites currently in view. The user interface for the game, on which the overlay of signal strength readings can be seen, is shown in the figure below.

Figure 11: The *Treasure* interface. The map overlay of the 802.11 wireless signal is shown in red, yellow and green in the interface. In this particular screenshot, taken shortly after a game started, no low signal strength (red) areas have yet been recorded.

In addition to collecting coins to gain points, players can also 'steal' coins from others. To do so both players must be in network range and within ten metres of one another. Players may then click the 'pick-pocket' button to attempt to steal all the coins a competitor is currently holding. Alternatively, if a player suspects another may be chasing him or her in an attempt to pick-pocket, he or she may activate a temporary shield that protects from pick-pocket attempts for a short period of time (20 seconds). If a player is lucky enough to be in range of multiple opponents, a successful pick-pocket can gain the coins from two or more opponents simultaneously. As pick-pocketing can only be achieved in areas of network coverage, there are advantages in being both in network coverage and in being outside network coverage. Whilst in network coverage, players learn about new coins they can collect, and may exchange coins they are carrying for points. However, they are susceptible to pick-pocketing and their location is made available to competitors and displayed on their maps. Outside network coverage players are safe from pick-pocket attempts and their locations are not shown to competitors. They may collect coins in relative safety without being observed on the screens of opponents' devices. However, they do not learn about new coins and are unable to exchange coins for points. Thus, to play the game successfully, players must learn the advantages of being both in network coverage and being outside network coverage, and balance their time appropriately in each. By taking part in the game players come to learn the characteristics of the network coverage in the area and discover how it can be both advantageous and problematic.

This situation of experiencing advantages and disadvantages in both being connected to a network and being isolated from it can be seen as a cipher for more general Internet privacy and awareness issues. When connected to the Internet, there is the potential for exposure to viruses, hacking or being spied upon—although often one must use the Internet, and the services available on it, to conduct parts of one's work or leisure. When disconnected from the Internet, a user's privacy and their machine's health are generally safer from these threats, and may still conduct many local network tasks within an office or home intranet. The design of *Treasure* represents in the small issues that are a common concern in the large. *Treasure* demonstrates that by drawing attention to these issues—forcing users to be consciously aware of them—users can gather deeper understanding of a system.

Figure 12: Player running through trees and bushes whilst 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, and at each location a large number of people signed up to play the game. Nearly all players reported that the game was extremely fun to play and this was substantiated by the observation that 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 (Figure 12). Research on *Treasure* was published at UbiComp 2005 in [6].

4.2.2 Seamful Design

Seamful Design opposes the traditional view of mobile design for ubicomp expressed by Mainwaring et al. [112]:

> *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.*

Seamful Design suggests that by inverting this traditional view, in which seams are hidden from the user, substantial benefit can be found in empowering users with the greater understanding observable and obvious seams facilitate.

Treasure exemplifies *Seamful Design* as it reveals the underlying 802.11 infrastructure to the players as part of an enjoyable game. Through playing the game, players who had no previous knowledge of either 802.11 and GPS were able to learn of their characteristics in a practical way, and use them to their advantage. As coins were only reliably uploaded in areas of good signal strength, players attempted to learn where the 802.11 access points were and upload when they were particularly close to them. For example, one player reported that they always attempted to upload "near the green node in the window". In actuality, the "green node" they identified was a brightly coloured first aid kit, and the access point they were uploading through was located in the office above, but was not clearly visible from outside. However, this rather amusing example does clearly demonstrate that players did develop an understanding of the 802.11 coverage and actively attempted to locate the nodes. Indeed, in the example, the actual access point was more or less directly above the first aid kit and the player did extremely well locating it as closely as he did.

Players also learned and used the characteristics of the GPS infrastructure to their advantage. We observed one player who placed his magnetic GPS antennae onto a metal fence, which allowed him to gain a more accurate GPS fix rapidly and collect a nearby coin he had previously had difficulty collecting. Similar to players in *Can You See Me Now*, we also observed that players currently carrying many coins and deep in an area of network coverage would attempt to 'hide' near walls or in trees where they suspected GPS quality to be poor. They realised that if they were in an area of poor GPS coverage their locations would be obscured from competitors who would then experience difficulty in getting their positions close enough to the 'hidden' player's for the pick-pocket function to work.

These examples demonstrate that by designing to expose infrastructure and information about the individual components that make up a system, and how they work together to form that system, users gain vital knowledge and understanding of the system. This, in turn, allows users to take advantage of the system at a level they would otherwise not be capable of and to deal better with breakdowns when they occur.

Treasure demonstrates that *Seamful Design*—design that attempts to make apparent the characteristics of the underlying infrastructure—can be of great benefit in the mobile environment. In *Seamful Design*, instead of attempting to patch over or hide problems in services from the user, such problems

are designed in a way as to make them obvious and clear to the user. In a system that fully reveals and provides descriptions of the seams in infrastructure, users are able, through understanding the reasons for disconnections or breaks in services, to mitigate problems themselves. In this way, a system that employs *Seamful Design* can provide more functionality and uptime than a system that attempts to patch over and hide problems. A common example of this, used in the first paper on *Seamful Design* [29], is a cell phone which hides the signal strength and currently connected cell information it may leave callers bewildered when they are unable to make calls and their phones do not respond. However, if users are shown the seams of the system, and told which cell they are connected to and how strong the signal strength is, then over time they will learn that connection to a cell and a high signal strength allow for high audio quality and low dropouts during calls. Therefore, when they are unable to make calls, they are more likely to understand why and seek out an area in which they can gain higher signal strength. This is echoed by the behaviour of players in *Treasure* who attempt to seek out areas of high 802.11 coverage to be sure of reliable network connection before uploading their coins for points. It is clear that players are using their knowledge of infrastructure to their advantage as they seek out 802.11 coverage when required and avoid coverage when they observe competitors following them.

The behaviour of players observed in *Treasure* as a result of *Seamful Design* leads to the final design guideline of this thesis:

Expose characteristics of underlying infrastructure where appropriate

It is clear that users are more adaptable than many designers expect, and by taking advantage of their intelligence, through *Seamful Design* and the revealing of seams this can be leveraged to overcome unexpected breakdowns in the infrastructure. *Seamful Design* is far more applicable in the mobile environment than the standard desktop environment simply because the infrastructure is currently less reliable. This is exemplified by the typical and commonly experienced fact that network disconnections and reconnections are normal and expected in the mobile environment whereas they are rare and surprising in the office environment. Thus, *Seamful Design* should be a vital component of every mobile system.

In order to understand how to implement *Seamful Design* in a mobile system, it is beneficial to examine a subset of Gaver et al.'s guidelines for designing for ambiguity [68] as their inversion from looking at ambiguity as a negative feature to perceiving it as a positive one in design echoes the inversion the concept *Seamful Design* applies to the underlying infrastructure in a system. Whilst Gaver et al. define many rules, only two seem relevant to *Seamful Design*. This is unsurprising, as although the two concepts of *Ambiguity as a Resource for Design* and *Seamful Design* share the stated inversion of negative to positive, they remain distinct concepts. The two methods are:

- *Expose inconsistencies to create a space for interpretation*
- *Cast doubt on sources to provoke independent assessment*

By exposing inconsistencies early on, before a system breakdown, the user has more time to learn to understand the nature of the inconsistencies, identify when they are within limits and when they are unacceptable, and to use the knowledge later to attempt to overcome the inconsistencies when necessary, or even find ways to take advantage of them. By casting doubt on sources, the user is also encouraged to seek actively to understand better the sources and the information provided. Again, the increased knowledge a user can gain from exploration initially caused by doubt can greatly aid in repairing breakdowns when they subsequently occur. However, one key difference between *Seamful Design*'s use of doubt and Gaver et al.'s use of doubt is that in *Seamful Design* doubt should only be encouraged or evoked when the user is likely to learn practically from the experience—whereas Gaver et al. often seem to support its use for artistic effect.

Although these ambiguity methods are two that are applicable to designers who wish to utilise *Seamful Design* in their own application, it should be noted that the over-arching goal is simply to expose the underlying components and infrastructure of both software and hardware (both internal and external) to users in order to provide them with increased knowledge which may provide novel benefits or aid in overcoming system breakdowns or problems. As stated, this is particularly important in mobile, peer-to-peer environments where the level of infrastructure systems rely on is generally far higher, and more transient, than in desktop systems. However, designers should be careful not to attempt simply to expose *every* aspect of infrastructure they can identify. Rather, *Seamful Design* should only be applied to those aspects of infrastructure that are potentially problematic, frequently causing breakdowns, or those that are potentially useful if people can identify ways to appropriate them. Such features may be identified early in the design process of a system through observational studies or through designers' prior experience and intuition in an area. By applying *Seamful Design* selectively in this way, it may achieve greater impact in highlighting a few crucial aspects of the system to the user.

4.2.3 Wireless driver in *Treasure*

The wireless driver we created proved to be necessary to the *Treasure* game. The game involved players frequently leaving and re-entering network coverage, and the frequent sampling of the networks SSIDs in range and their strength. The default wireless driver simply would not have supported this behaviour as it does not robustly and automatically rejoin networks, requires frequent user input when it does, and does not update signal strength (or expose it programmatically) frequently enough to give adequate feedback to players. Furthermore, the default network driver also commonly disconnects from a working network when it detects a new or slightly stronger network. This is often a disastrous decision as the newly detected network may prove to be unusable due to encryption, MAC security or simply because the detected access point does not connect to any usable network.

By 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 scan automatically for network coverage, detect the network used for the game and instantly connect, configure and, importantly, stick

to that particular network for as long as it was available. From the goals stated at the start of this chapter, the driver at this point fulfils two of the original goals:

- Require no input from the user
- Automatically configure the device for use after a network is joined

The implementation of the driver used for *Treasure* was a significant feat. The code required to implement it is extremely complex and low-level and, although it is clear that many others have tried to implement similar drivers, none have managed to do so with the combination of compatibility this one provides. This is exemplified by the fact that the mobile wireless scanner application subsequently developed using this driver, WiFiFoFum[12], is compatible with more 802.11 devices and cards than any other mobile wireless scanner. However, the driver used for *Treasure* is only part of the solution required for supporting mobile, peer-to-peer applications. It is 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* (section 4.3.3).

4.3 Switching networks and discovering peers

As discussed previously, the ideal network situation for most mobile applications is that they be on the appropriate network to discover peers as often as possible yet still allow users to connect normally to infrastructure access points which permit connection to the Internet. To support this behaviour two improvements were made to the wireless driver used in *Treasure*. Firstly, default logic was added to the driver code to allow it to continually select and join the network where peers are currently most likely to be found. Secondly, a peer discovery mechanism was implemented. Both the improvement to the original wireless driver and the creation of the discovery mechanism were designed and implemented entirely by myself.

4.3.1 Improvements to the wireless driver to support peer-to-peer

Although the driver used in *Treasure* was a significant step forward for use in mobile applications, it did not adequately support mobile, *peer-to-peer* usage. It has been seen that mobile, peer-to-peer applications must allow users to continue to perform the tasks they normally perform such as checking email and browsing the Web whilst also maximising their opportunities for encountering peers. In order to achieve this over 802.11 wireless, mobile peer-to-peer systems must therefore allow the normal connection to 802.11 infrastructure access points when they are available but find a way to continue to discover peers when no infrastructure access points are available, as mobile devices frequently encounter areas of infrastructure coverage and areas of no coverage.

To implement this behaviour, additional logic was added on top of the existing wireless driver. The driver is used to scan continually for infrastructure access points and if any are in range the driver is

[12] Malcolm Hall primarily developed WiFiFoFum, with some sections of code being contributed by myself. It is available for download at http://www.aspecto-software.com/WiFiFoFum

used to connect automatically to the infrastructure network and an attempt is made to use DHCP to assign an IP address for the device. If an IP address can be assigned in this manner, a check is conducted to verify if a connection to the Internet can be made. If an IP address cannot be assigned and there are more infrastructure networks in range, the process is repeated, testing the next network in the queue, which is arranged in order of signal strength. If an IP address cannot be assigned using DHCP but there are no other networks in range (or all networks have been tested) then the device stays connected to the current network and instead uses a self-assigned IP address. However, if there are multiple networks in range, none of which have a connection to the Internet, then the driver will automatically switch between them if no peers have been detected after a predefined period of time.

This behaviour automates the process of connecting to infrastructure networks and maximises the opportunities for encountering peers when near infrastructure access points. If any of the infrastructure access points provide a connection to the Internet then all devices using this driver will connect to it. This allows mobile peer-to-peer applications to continue to allow normal use of the Internet to the user whenever it is available. Indeed, since this process is automated it generally makes the Internet easier to access than would normally be available on the device. If none of the access points provide a connection to the Internet then devices will still connect and use self-assigned IP addresses in the same subnet range. Furthermore, if there are multiple infrastructure networks, devices will switch between them until a peer device is encountered. Once a peer is encountered, both will stay on the same network. Thus, even if there are multiple access points and networks available, devices using this driver will attempt to cluster around one particular network.

The behaviour of the enhanced driver when it is near infrastructure access points is described by the following pseudo-code:

```
1 Scan for infrastructure networks
if (already connected) {
        if (Internet is not available) {
                if (new networks found since last scan) {
                        disconnect from current network;
                } else if (predefined time passed without encountering peers AND other networks available) {
                        disconnect from current network;
                        connect to next network and use self-assigned IP;
                }
        }
        goto 1;
}
else if (networks available) {
        sort networks by signal strength;
        for each (network in list) {
                attempt to connect, assign IP address and check for Internet connection;
                if (connection is not suitable AND more networks in list)
                        try next network;
                else
                        use self-assigned IP address;
        }
}
goto 1;
```

Connection to particular access points and networks is achieved by setting the SSID and associating the card with a particular BSID. The code shown provides a comprehensive solution for ensuring mobile devices use the connection with Internet availability if there is one among multiple networks in range. Whether an Internet connection is available or not, the enhanced driver ensures that peer devices will cluster around the same network and thus maximise their opportunities for encountering one another. However, it does not allow peers to discover one another when infrastructure access points are not in range. Luckily, there is a simple solution to this problem.

The driver behaves as described whenever a scan is conducted and infrastructure access points are available. However, when the device is not connected to any network and a scan reveals no infrastructure access points in range, the enhanced driver simply switches the network device to ad hoc mode, joins a pre-defined ad hoc network and uses a self-assigned IP address. Thus, two devices in range of one another but not within range of any infrastructure access points ensure that they are on the same ad hoc network and are able to discover one another and communicate through the ad hoc network. This enhanced driver essentially ensures that mobile devices are able to locate and find an Internet connection whenever one is available, find any peers in the area when they are near infrastructure access points and find any peers in the area when they are in the wild (out of range of infrastructure access points).

If two devices running the enhanced driver are brought near an infrastructure access point they will both automatically connect to the infrastructure access point and be able to discover and communicate with one another through it. Even if multiple infrastructure networks are available they will eventually meet on the same network and stay connected to it (Figure 13). If the same two devices encounter one another in the wild they will both be in ad hoc mode on the same network SSID and be able to discover and communicate with another directly.

Figure 13: Diagram of device behaviour when multiple access points are in range. The wireless driver ensures that, even when there are multiple infrastructure access points in range, devices will cluster around one access point to ensure they may communicate with one another.

If the same two devices are in range of multiple infrastructure access points and one allows a connection to the Internet, the two devices will select this network above the rest and connect to it. They will then be able to discover and communicate with one another through the infrastructure network as well as providing a standard Internet connection to the user (Figure 14).

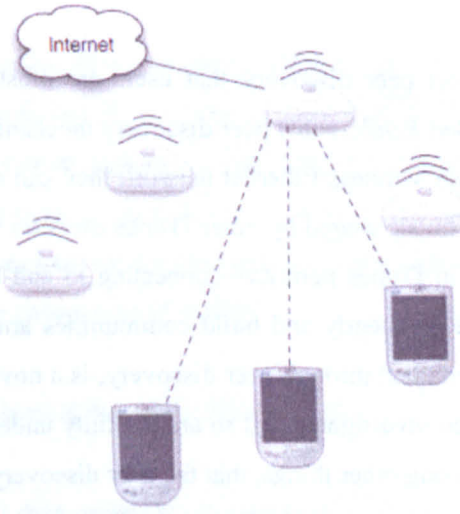

Figure 14: Diagram of device behaviour when an Internet connection is available through an access point which is in range. When there are multiple access points in range the driver ensures the one with a connection to the Internet is selected if one is available.

Allowing automatic connection to infrastructure networks and falling back to a predefined ad hoc network when there are no infrastructure access points in the area means that devices can locate peers

regardless of where they are. It also means that they are able to connect to the Internet normally through infrastructure access points. This allows the user to both use the mobile application and continue to use their normal Internet applications without being bothered with pop-ups from the operating system.

The enhanced driver fulfils another two of the goals set out at the start of the chapter:

- Automatically select and join networks where peers are most likely to be found
- Continue to allow users to perform standard tasks (e.g. check email, browse the Web)

Throughout this section there have been many mentions of a device connecting to a network and "discovering" peer devices. However, the discovery of peers is not a simple matter for mobile devices and is implemented as part of this thesis, and described in the next section. In addition, situations in which the algorithm implemented may potentially fail are also discussed.

4.3.2 A peer discovery mechanism

Discovery of peer machines is often taken for granted in desktop and laptop systems as it has been available for some time. Perhaps the most widely used and earliest peer communities were used solely for the purposes of sharing files over the Internet. Most of the peer-to-peer sharing in these older systems was achieved using pseudo peer-to-peer discovery in which the IP address of a server is required to bootstrap the process. Machines register themselves with a central server or supernode and, rather than discovering other peers directly, simply retrieve the list of peers stored by the server. These configurations are still common today and both the Gnutella [86] and eDonkey [82] networks work in this manner with no direct peer-to-peer discovery but rather pseudo peer discovery achieved through the use of a server.

The most common form of direct peer discovery that users are most likely familiar with today is Apple's *Bonjour* (previously called *Rendezvous*) peer discovery mechanism used in iTunes. Whenever iTunes users connect to a wired or wireless Ethernet network they can elect to share their music with others and can, in turn, view the music shared by other iTunes users on the network. The simplicity of sharing that the use of *Bonjour* in iTunes permits—connecting to and browsing others' music – has allowed users to use this service frequently and build communities around sharing. Newman et al. realise that this type of sharing, enabled through peer discovery, is a novel type of sharing – the social aspects of which have not yet been investigated and so are not fully understood [123]. They conduct a study of iTunes users and find, among other things, that the peer discovery alone allows complex social interpretations to be made about when a user is off work, at lunch or has left for the day. It is clear from such research that users find significant value in creating and using peer communities that develop in this manner – both from technical and social standpoints.

Implementations of *Bonjour* are available for both Microsoft Windows and Mac OS X. Both rely on mDNS to achieve their peer-discovery functionality, and both are essentially implementations of the

ZeroConf standard (RFC 3927[13]). Microsoft also has its own version of ZeroConf that it calls SSDP[14] (Simple Service Discovery Protocol), and it is included in modern versions of Windows. The various versions of ZeroConf available on each desktop platform can provide novel functionality to desktop machines and improve ease of use in existing systems such as instant messaging[15], sharing of clipboard contents[16] and discovery of printers [2]. Despite the clear advantages of peer discovery exhibited on standard machines, there is no implementation of *Bonjour*, ZeroConf or the mDNS protocols available for mobile devices.

The lack of a mobile version of any of these discovery techniques is surprising as peer discovery is more applicable, and important, in the mobile environment. Certainly, a reliable peer discovery mechanism is vital to much of the subsequent systems described in this thesis.

As there was no existing peer discovery mechanism available for mobile devices, one was implemented as part of this research. The idea, design and implementation are all my own work and the concept is the direct result of the findings described previously in this thesis.

The peer discovery mechanism is titled SDS (abbreviated from 'Self Discovering Spaces') and provides essentially the same functionality to mobile devices that *Bonjour* does for desktop devices. It exposes a simple API that allows devices to broadcast their existence, and to listen for and be notified of any other peer devices that are broadcasting their existence. Additionally, meta-information such as the device's IP address, owner's name, services it is providing and ports through which those services can be accessed may be included in the broadcast messages themselves. By default, peers using SDS broadcast their existence every 1 second, but this period can be customised to any desired value using the API.

In addition to the meta-information about the client contained in every broadcast, a further set of 20 strings referred to as TextRecords can be encoded in the broadcast messages. These can be used by applications to embed any additional information that they may require. Peer devices using SDS for discovery can hook into events to be notified of newly discovered peers and when a known peer alters its broadcast message. This may happen if a peer adds a service to the list of services it is advertising, alters one of its TextRecords or changes its IP address.

SDS fulfils the last two goals listed at the start of this chapter:

- Advertise the device's own existence on a network to allow peers to find it
- Discover peers once on a network

[13] http://www.ietf.org/rfc/rfc3927.txt
[14] http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt
[15] http://www.apple.com/macosx/features/ichat/
[16] http://www.porchdogsoft.com/products/spike/

It should be noted that the idea of peer discovery is not novel or unique to this thesis. This is clearly apparent from the existing implementations available for desktop machines, Apple's *Bonjour* and Microsoft's *ZeroConf*. However, the implementation for mobile devices is new and the combination of the enhanced wireless driver and SDS to provide a comprehensive peer discovery mechanism for use in all mobile network scenarios is new.

The code for SDS has been made publicly available as part of the Equator Software Archive and can be downloaded from the Equator website (http://www.equator.ac.uk).

Although the combination of enhanced wireless driver and SDS provides an efficient and robust method for peer discovery on mobile devices, it is not a perfect solution. Two situations, both involving infrastructure access points, have been identified in which peer devices within range will fail to discover one another.

In Figure 14 it was shown that multiple devices encountering several access points in an area will gradually settle on a single wireless network that provides a connection to the Internet. However, if multiple access points provide an Internet connection and devices initially connect to different ones, the devices will elect to stick to these different networks, unaware that a peer device is nearby and connected to another available network. One possible solution considered during the design of SDS is to continue to switch to and test the remaining networks in range even when it has been detected that one, or more, of those networks provide an Internet connection. However, it was thought that such behaviour would be frustrating, as users would experience intermittent Internet availability even while remaining in a single area, as the driver continually switched to and from networks that had Internet connectivity. As one of the initial goals was to allow the use of standard Internet connections and functionality when available, this solution was subsequently considered unsuitable.

The second situation in which discovery may fail is if four or more devices simultaneously begin sensing and attempting to connect to infrastructure networks. Figure 13 shows that when there are infrastructure access points available, none of which have an Internet connection, devices will continually switch from one network to another in order to detect peers which are in range but on another network. If four or more devices begin this process at approximately the same time, there is a chance that two will encounter one another on one network whilst the other two discover one another on a second network. As, after discovering a peer, devices then stay on the same network the devices would discover at least one other device, but fail to discover the second set of devices connected to the other network. Although the likelihood of four or more devices conducting this process simultaneously is low, and attempts are made to lessen its occurrence by searching networks in the order of signal strength, it is still possible for peer devices to fail to discover one another in this way.

It may be possible to overcome this second problem by slightly altering the algorithm to permit peer devices that have identified one another on a network to agree to jump simultaneously to another

network. In this way, groups of peer devices could continue to search for other devices whilst ensuring they switch to networks at the same time as the devices in their group. However, the implementation and exploration of the viability of this solution is left to future work as it may prove to be a complex task. It is likely that such a solution would introduce a large amount of additional network traffic, and complexity, as devices exchanged information about which networks they were in range of, and negotiated which one to switch to next, with the devices in their group.

A simpler solution applicable to both the problems identified would be available if the hardware were capable of connecting to multiple networks. If connection to multiple networks were possible, other networks could be searched for peers or Internet connectivity without interruption to current Internet connectivity or connections to peers. Whilst this may currently be achieved through the use of two physical 802.11 cards, this is a poor solution as extremely few users, or devices, have two 802.11 cards. A better solution may be found in Microsoft's *Virtual WiFi*[17], which allows a single network device to switch rapidly between multiple networks and network configurations—essentially allowing a single network device to behave as two independent devices. Currently, *Virtual WiFi* is only available for desktop versions of Microsoft Windows and only supports wired connections. However, if this technology were ported to mobile versions of Windows in the future, it may provide a suitable solution for use with the enhanced wireless driver and SDS.

Whilst the solution to peer discovery presented here does have the two problems identified, these are unlikely to occur commonly, and the solution provides an extremely robust and reliable technique for the discovery of peer devices in many varied mobile environments in general.

4.3.3 Enhanced wireless driver and SDS in a mobile, peer-to-peer game

The enhanced wireless driver and SDS were used within a second mobile game, titled *Feeding Yoshi.* The *Feeding Yoshi* game descends from a game concept originally conceived by Malcolm Hall, Scott Sherwood and myself. This original idea was adapted by all the Equator researchers at Glasgow University and several at the University of Nottingham to create *Feeding Yoshi,* and the game was subsequently implemented by Malcolm Hall, Scott Sherwood and myself. Whilst the main research interest relevant to this thesis in the game was testing the enhanced driver combined with SDS, to verify they were of benefit at both the system and user level, the game was also used to research how mobile games could be designed and implemented to be played over a relatively long period of time (a theme which it has in common with games such as FIASCO [32]) and to further investigate *Seamful Design.* Research on *Feeding Yoshi* was published at CHI in [10].

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

[17] http://research.microsoft.com/netres/projects/virtualwifi/software.htm

seeds from the Yoshis themselves—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, e.g. feeding all five desired fruits at once scores 150 points. Feeding a Yoshi a fruit it does not want results in the player losing 10 points. 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 attend to the PDA screen.

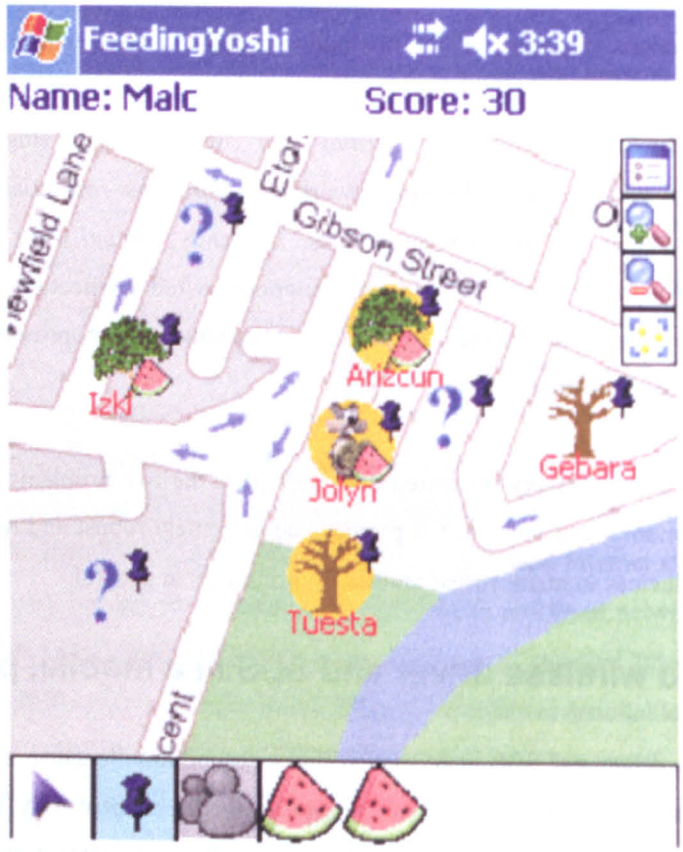

Figure 15: Map screen of Feeding Yoshi game. In the map screen, Yoshis and plantations are shown as icons, and navigation controls are on the right. Near the bottom is a row containing (from left to right) a button for selecting icons, pinning an icon onto the map, initiating a swap with another player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

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. On the right 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 done by selecting a seed in the basket then clicking a Seed button.

Figure 16: Yoshi screen of Feeding Yoshi game. 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.

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. This is useful if the Yoshis in the areas that a player knows want fruit that do not grow there. By swapping with team-mates 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 the 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 players move around the city, their PDAs continually scan for the presence of wireless networks. Secured wireless networks become Yoshis and open networks become plantations. While it would be an easy and, perhaps, obvious solution to communicate with the *Feeding Yoshi* game server via the open access points that are discovered (e.g., to upload scores automatically), as has been previously pointed out at the start of this chapter, 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 [76]). 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 the 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 scores were behind another team or player's score. Swapping fruit between players is achieved through the use of the enhanced wireless driver and SDS. Each game client continually broadcasts its own existence using SDS, and uses the enhanced driver to swap appropriately through infrastructure and ad hoc networks. When another PDA is detected and one of the players wishes to initiate trading, that player's PDA stops scanning and sends a message requesting the other PDA to cease scanning too; important as the constant scanning is a relatively heavyweight task for the 802.11 equipment on standard PDAs. The exchange itself is done through traditional TCP socket connections. During the game trials it was found that most participants did not use the map the game provided, and instead primarily relied on the alternative list interface that showed the same Yoshiis and plantations. Therefore, the map was removed in a subsequent version of the game that was made available for public download at www.yoshigame.com. As of July 2006 the game has been downloaded over 3500 times.

The success of the enhanced wireless driver and SDS was apparent in the trials of *Feeding Yoshi*. As players carried the devices with them over the course of the week, many became familiar with their general use and began to utilise the Internet and mail capabilities it had. Indeed, as part of the game required users to "claim" their points by entering a code on a website, many users reported that they simply used the PDA itself to do so when a connection was available. Without the enhanced driver, players would not have been able to both use the Internet when it was available and run the game simultaneously.

The combination of enhanced wireless driver and SDS also allowed users to work together in various scenarios. Users in the same team reported collaboratively working together. In one instance, four members of one team arranged to meet during their lunch break. By travelling as a group of four they were able to fulfil all of a Yoshi's five desires more often (and therefore gain the maximum number of points). Each of the four users deliberately tried to carry different fruits. When a Yoshi was encountered, the team swapped the required fruits to a single user's basket until that user had the required five fruits the Yoshi desired. The smooth interaction between players both when away from Yoshis (access points) and when close to them would again not have been possible without the intelligent swapping of networks and network modes, and a reliable peer discovery mechanism.

The fact that peer devices maximised their opportunities for discovery is best exemplified in what might previously have been considered a rather unlikely meeting of players from different teams. Each of the two teams consisted of four players, most of whom worked in Glasgow, but none had a workplace within two miles of those of the players in the opposing team. Surprisingly, despite the extremely large play area, two players from opposing teams met by chance through the game during the one week they played it for. One of the players describe the event herself:

> *"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 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."*

Whilst this encounter was indeed an unlikely occurrence within our trial setup it would simply not have been possible at all without the enhanced driver and SDS. When the players encountered one another they were simply playing the game as normal, gathering fruit and feeding Yoshis on their own. In these situations, the enhanced driver is intelligently switching between ad hoc and infrastructure mode. Without this capability, and the driver's ability to select the network which has the highest chance of containing peers, it is extremely unlikely that both devices would have been on the same network at the same time. The enhanced driver ensures they are on the same network when they are co-located, and SDS provides an extremely fast and highly reliable discovery mechanism once they are.

4.4 Conclusion

The combination of wireless driver and SDS fulfils the first of the four pieces of infrastructure identified as required for mobile, peer-to-peer systems in Chapter 3:

- A mechanism for intelligently selecting which networks to use and for reliably discovering peers on these networks

The earlier work in this thesis led to the identification of the clear need for a peer-to-peer discovery solution that facilitates the discovery of peers regardless of the location or connection type of the mobile device, whilst still permitting the user to take advantage of a standard Internet connection whenever one is available.

An investigation into underlying communication technologies identified that 802.11 was more suitable for use in peer-to-peer applications than either GSM or Bluetooth. By combining an enhanced wireless driver with SDS, a comprehensive solution running on 802.11 was found which provides extremely rapid yet reliable peer discovery for a large number of mobile phones and PDAs. In addition, the solution not only permits users to continue to utilise Internet connections when they are available but

also actively seeks such connections and automatically configures devices to use them whenever available, all the while permitting peer discovery to occur.

The wireless driver was used in *Treasure* and the combination of enhanced wireless driver and SDS was demonstrated in *Feeding Yoshi*. This combination is again used in systems implemented and described later in this thesis.

The combination of enhanced wireless driver and SDS provides automated peer discovery to mobile devices at a level that was not possible previously. Devices can reliably discover one another regardless of whether they are close to or distant from infrastructure access points without requiring that the device's network card be tied to any single configuration that would prohibit use of the Internet or limit connections to newly discovered networks. This is an extremely important contribution, as the mobile application market is rapidly growing and this solution allows peer-to-peer applications to run without interfering with the normal use of the device for Internet access. With a solution that does not compromise existing uses of a mobile device, the uptake of new mobile and mobile peer-to-peer applications can proceed more rapidly.

5 A HYBRID POSITIONING SYSTEM FOR MOBILE, PEER-TO-PEER APPLICATIONS

During the examination of the many existing mobile, peer-to-peer applications as part of the literature review, it became apparent that no single positioning system is suitable for general use in such applications. As was made clear, several researchers have realised that hybrid positioning systems may provide a more suitable solution, and at least one basic attempt to create such a system was made by Baus et al. [8]. However, this basic attempt required considerable setup, was specific to one application and was in no way a lightweight or general solution.

Analysis of both the *Lighthouse* and *George Square* systems confirmed that current indoor and outdoor positioning systems are indeed not a suitably comprehensive solution and that a hybrid solution is likely to provide greater performance. The required solution was identified at the end of Chapter 3 as:

- A hybrid positioning system that requires no initial setup and yet has high availability in providing location both indoors and outdoors

This can be broken down further for the sake of clarity as:

- Able to locate users or devices both indoors and outdoors
- Little or no initial setup cost
- High availability (able to provide a location as often as possible)

One system that comes close to fulfilling these goals is Intel's *Place Lab*. However, as will be seen, *Place Lab* fails to offer a general solution for mobile, peer-to-peer applications in several crucial areas.

This chapter begins with a brief review of previous positioning technology, and how it has been utilised in the systems created and discussed earlier in the thesis, in order to clarify further the problems with existing systems. It then examines the most advanced existing hybrid positioning system currently available, *Place Lab*, and identifies several drawbacks relating to the four goals listed above. Finally, a new hybrid positioning system created as part of this thesis, *Navizon*, is introduced and discussed.

5.1 Review of positioning systems

Although an investigation of existing positioning systems was presented earlier in the literature review, this section provides a brief review concentrating on the particular shortcomings and successes of such systems, and how attempted upgrades or augmentations often simply improve one aspect of a positioning system at the expense of another.

In all the systems discussed previously, both in the literature review and as parts of this thesis, accurate and continually available positioning of the user has remained, to differing extents, a problem. The *Lighthouse* positioning system required an extensive amount of set up, and was tied to a single small, indoor area. Whilst the GPS positioning utilised in *George Square, Treasure* and *Feeding Yoshi* allowed a far greater area to be covered, it was less accurate and still failed to cover substantial areas (such as narrow streets and, of course, any indoor location). As an extremely large number of mobile systems rely on position information to provide functionality, it is vital that a positioning system with extremely high availability is found.

A positioning system's value may be judged in three main categories: accuracy, potential area of use and availability (or uptime). Unfortunately, none of the traditional positioning systems available today performs well in all of these categories. As has been seen, high accuracy has been achievable in many indoor systems using ultrasonic and RF techniques such as the positioning system developed at Bristol and used in the *Lighthouse* [20], and the Bat system [85].

The Bat system is important as it built on the seminal work carried out in the Active Badge system – one of the first accurate and convenient indoor positioning systems that was actually deployed and in use for a long period of time [84]. Most indoor positioning systems require substantial setup and the Bat system is typical with a high installation cost. It requires a matrix of receivers to be installed on the ceiling of the area where positioning is required. Each receiver must be connected by wire to a central computer that coordinates both the transmission of identification requests to clients and the calculation of the position when the receivers detect a beacon. The transmitting devices, which are the items that are tracked, do not periodically transmit a beacon as multiple devices may be co-present and doing so may cause interference. Instead, the transmitters also have a receiver that detects incoming requests and responds by broadcasting a pulse when a request corresponding to its ID is received. Thus, the computer controlling the system must be programmed with the IDs of any devices that are to be tracked prior to their entry into the environment—further complicating the setup process. It then polls each device it is aware of in turn, asking each to broadcast a pulse. Clearly, such a system involves a tremendous commitment in installation and maintenance. However, although now many years old, *Bat* remains one of the most accurate positioning technologies; able to position devices to within a few centimetres accuracy.

Whilst *Bat* performs excellently in accuracy, it has neither a high area of use or high availability—it is only available in a small area after a costly setup is conducted. Other ultrasonic and RF systems such as Walrus [16] and Landmarc [124] generally have similar profiles of high accuracy but a significant set up time and small operative area.

Many alternative techniques to ultrasonic and RF positioning in indoor environments exist but these have similar problems in requiring extensive setup. For example, CarpetLAN [66] is a novel technique in which a user can be located through the use of a high speed switching voltage which travels from a

mobile device through the body itself to the carpet tile the user is standing upon. Each carpet tile is a node which can, to an extent, automatically configure itself when placed adjacent to existing tiles. As each tile is aware of its own dimensions, it can calculate its location and location extents from information gathered from neighbouring tiles. Whilst this aids in simplifying the extension of an existing CarpetLAN positioning system, the initial cost and setup remains substantial – as does the financial cost of each tile.

Some indoor positioning techniques attempt to greatly increase availability by trading away accuracy. For example, Bluetooth is now available on the vast majority of modern mobile devices and consumes a relatively low amount of power. Thus, by relying on Bluetooth, an indoor positioning system may hope to become available to a far greater audience. Hallberg et al. investigate the plausibility of Bluetooth positioning, and manage an accuracy of around 1.7 metres [80]. However, due to limitations with the Bluetooth technology, particularly the length of time it takes one device to pair with another and the transmission range of Bluetooth, this accuracy level was only achieved with severe restrictions. For example, a user travelling at normal walking speed would likely be too fast for their device to create the necessary connections to receive position information, and so in the trials the devices were either stationary or moved very slowly. Furthermore, the setup required Bluetooth nodes to be installed throughout the building. Thus, higher availability is not really achieved as nodes must be installed and setup cost remains high.

Most indoor positioning systems rely on a form of triangulation from beacon signals. By discarding triangulation, some positioning systems may attempt to increase availability, again at the cost of accuracy. In the MobiTip system, Bluetooth kiosks were used to identify the general region a device was in—such as a shopping centre or a university building [146]. Whilst this technique allowed enough time for the Bluetooth connections to be created and thus was useable as part of a prototype system, the accuracy was obviously very low—at the building level. There is a clear pattern in many current positioning systems, in that they attempt to improve on a certain feature at the cost of another. In the case of MobiTip, availability is improved but with a substantial cost to accuracy.

One common theme in all these position systems is that they rely on a centralised machine for coordination purposes. The setup of the central machine and the connections to it are often the main causes of high setup and maintenance costs, and result in only small areas being covered and general low availability of the system. The Cricket system attempts to overcome this by offering a decentralised solution [139]. Indeed, the authors state that one of the goals of Cricket is "widespread building-wide deployment" and that they believe "that it is not possible to deploy and administer a system in a scalable way when all control and management functions are centralized". Such a statement appears to be supported by the fact that few of the previously discussed positioning systems have ever been deployed throughout an entire building or used for any length of time. Indeed, it would appear that the only indoor positioning research systems that were deployed and actively used throughout a building over a period of time were Want's original Active Badge system and Olivetti's

Bats system. Cricket's decentralized architecture relies on occupants of a space being responsible for installing beacons if they wish the positioning system to be available in that region. Clients listen for beacons and attempt to infer their position from the list of visible beacons and, if required, a map server can be consulted. Decentralisation is achieved in the Cricket system by moving a significant amount of data and calculation to the client. Clients must maintain a list of beacon locations or, if possible, consult an external database to look up the beacons it can see. Essentially, the beacons are dumb nodes that do nothing but transmit an identifier. Whilst this does allow beacons to be easily placed, it presents many problems for the client as it introduces a number of requirements on the client side.

Overall, it is clear that none of the existing indoor location techniques provides a comprehensive solution to locating an indoor user accurately at all times. Indeed, the authors of Cricket identify this themselves, and determine that neither RF nor ultrasound on their own are adequate, and so determine to use both to gather a more accurate reading (RF to receive an initial broadcast signal and do some initial calibration and then ultrasound to receive a final set of pulses and triangulate from these). Whilst most indoor systems are accurate once their infrastructure is in place, the prohibitive cost, in both time and effort, to install them results in few areas being covered. Similarly, the extra equipment that must be appended to the mobile devices for most indoor systems to operate is awkward and can consume a large amount of power – making them undesirable and useable for only short periods of time. For these reasons it is rare to find a continuously available indoor system, and there are no known commercial applications available to the general public that make use of indoor location information.

Outdoor location systems often have an inverse profile to that of indoor ones, in that they are less accurate but require minimal setup and are available over far greater areas. The most widely known and used outdoor positioning system is undoubtedly the Global Positioning System, which gives varying results depending on the user's location and the weather. Typically, a user's position can be determined to within a few metres under good conditions. As GPS is so widely known and has already been discussed in many systems earlier in the thesis, it will not be discussed further here.

Despite the prevalence of GPS for outdoor positioning there are several other techniques available. These normally take advantage of the existing technologies already available on the mobile device and the existing infrastructure in the environment. For example, the commercially available Ekahau system relies on a device's 802.11 wireless capability to detect wireless access points in the area and triangulate a position from them. Ekahau works both indoors (at room level accuracy) and outdoors [59]. However, it requires a considerable amount of setup. Before it can be used, the locations of all the available access points in the target must be measured and then a comprehensive sampling of the area must be conducted in which the signal strengths of visible access points must be recorded four times for every square metre of area to be covered. In areas were the position is found to be weak it is simply recommended that another access point be inserted into the area (even if it is not needed for data transmission purposes).

Unlike indoor positioning systems, outdoor positioning systems are commonly used in commercial applications available to the public. For example, TomTom Navigator[18] is an extremely popular wayfinding application which runs on a variety of mobile devices including phones, PDAs and purpose built TomTom units. In addition to practical applications utilising outdoor positioning systems, games have become more popular. MogiMogi[19] has been a popular location based game in Japan for over a year, and Sony have recently announced that future games for their PSP may utilise a GPS hardware plugin in order to make physical location a gaming element[20].

Despite the fact that applications utilising the user's location are starting to become popular, it remains clear that there is currently no positioning system providing a comprehensive solution to locating a user accurately both indoors and out. Indeed, even if only indoor positioning is required, a single technology is not sufficient and, as in Cricket, multiple technologies must be combined. It is, therefore, advisable to investigate the amalgamation of multiple positioning technologies in order to provide a more comprehensive solution.

5.2 Place Lab

Place Lab is a positioning system that attempts to combine many technologies in the hope that, working in conjunction, the technologies will be able to provide greater uptime and accuracy than any single system could previously achieve. *Place Lab* provides an API that allows modules for different positioning technologies to be utilised with the core *Place Lab* library. However, the majority of the modules currently available with the *Place Lab* software rely on the detection of statically placed beacons and the subsequent triangulation of beacon signal strength in order to calculate the position of a mobile device.

Triangulation is a simple and well-understood technique, and is commonly used in many positioning systems. *Place Lab* continually scans for known beacons and records the signal strengths to them. If only one beacon is detected it simply places the user at the location of the beacon. If two or more beacons are detected it uses the signal strengths to each to estimate a likely position for the device.

In *Place Lab*, the static beacons can be wireless access points, cell towers or fixed Bluetooth beacons. The project is open-source and the code exposes a Java API that can be used to code additional modules to further the number of possible beacon types used for positioning. For example, a module could be added to allow IR beacons to be utilised in a similar way as the fixed Bluetooth devices are. However, to date there have been no publications referring to any but the three included (802.11, GSM and Bluetooth) positioning technologies being used with the *Place Lab* software.

[18] www.tomtom.com
[19] http://www.mogimogi.com/
[20] http://uk.gear.ign.com/articles/729/729703p1.html

Before an area can be used to locate users with *Place Lab*, it must first be sampled to detect where some of the beacons are located. This involves utilising GPS whilst scanning for wireless, cell and/or Bluetooth beacons to discover the positions of as many static beacons as possible. An important failing of *Place Lab* in the discovery of beacons is that the software that must be run to carry out detection is not the same as the software that is run to position users later using the locations of beacons. This introduces complexity and effort to the setup required before *Place Lab* can be used in a location.

Users must first install and execute the beacon finding software and completely cover the area in which they wish positioning to be available in order that enough beacon samples are gathered. They must then either install and execute the positioning part of *Place Lab* on the same device, or copy the database of collected access points to a new device they wish to use. Alternatively, *Place Lab* recommends that users check if beacon data for their area is available from Wigle.net, and download data from there if available. Users are also encouraged to copy manually the logs to a desktop machine and upload them through the *Place Lab* website. Finally, the positioning component of the software can be run in the sampled area in order to locate the user carrying the device. However, no new beacons are detected whilst using this part of the software and this makes a comprehensive sweep of any areas where the positioning software is to be used vital during the initial detection stage. Clearly, this complexity introduces an extremely high setup cost and directly opposes the aim of having *little or no initial setup costs*.

Research has been conducted in using diffusion techniques to calculate the location of newly discovered beacons in order to partly lower the initial setup costs of beacon systems. Spratt explains the concept of diffusion [158]:

> *The Positioning by Diffusion approach is characterized by a seamless inter-working of the positioning algorithm across Access Points, Static Devices as described, and Mobile Devices. The actual movement of Mobile Devices is used to diffuse position information from Access Points – which have had their position programmed into them – to Static Devices which the former passes close to. Subsequently, other Mobile Devices passing close to Static Devices can obtain this stored position information to perform positioning calculations. The position data originally emanating from a small number of Access Points can be combined to perform positioning calculations.*

Diffusion of this kind allows nodes that were not captured during an initial sampling of an area to be later detected, positioned and included as resources for use in a positioning system. However, the locations of nodes detected in this manner are not as accurate as those achieved by traditional sampling. Despite this, LaMarca et al. are able to demonstrate that a sampling of only 20% of the available beacons led to an accuracy of 56 metres with 84% coverage on beacons that had not been captured originally, when diffusion was applied to determine their positions [105].

The use of diffusion techniques allows for the possibility of positioning accuracy and the number of sampled beacons to continue to grow after an area's initial sampling and creation of trace logs. A general overview of the type of diffusion *Place Lab* is capable of utilising is as follows. When a beacon, such as a wireless AP (access point) is sampled, the time is recorded. Subsequently, when a new AP is found the time difference between detecting the known AP and the new AP is calculated. As the majority of mobile devices are carried and enabled when users are walking, an assumption is made that the velocity of the user is no more than a certain value, say five metres per second. This allows a radius around the known AP to be created in which it is possible the new AP is located. If more readings are recorded with times from other known beacons, from which more radii can be calculated, the circles will overlap. As can be seen in Figure 17, the area where all, or the majority of circles overlap, is the likely location of the newly found beacon.

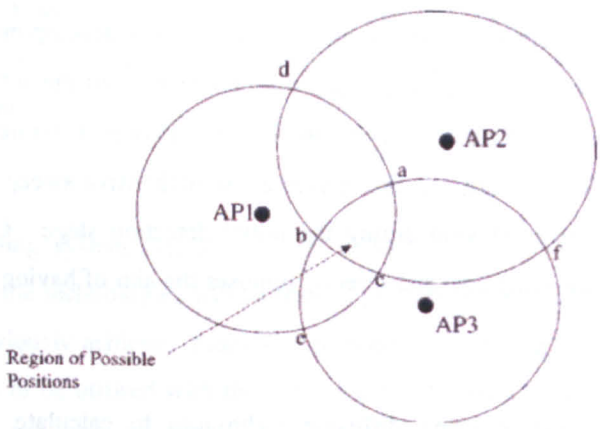

Figure 17: One technique used in positioning by diffusion. Overlapping time circles allow for a newly detected beacon to be placed within the overlaps (taken from [158]).

It should be noted that whilst *Place Lab* has implemented diffusion techniques, they are still experimental and not included as part of the default download package. Thus, *Place Lab* has not yet overcome the problem of incurring an initial high setup cost. Furthermore, even with diffusion techniques, a significant sampling of the area must still be conducted and a generally uniform number of beacons sampled over the area to maximise the chance of overlaps. The use of diffusion techniques to estimate beacons locations (rather than relying on correctly sampled beacon readings) also leads to a gradual decay in quality and accuracy over time and will never be as accurate as raw sampling.

Although *Place Lab* is in theory an extremely comprehensive system for providing location, it does have some further weaknesses that can make it a poor choice for use in a mobile application. One of the primary problems is that not all components of the system run on many mobile devices. Although the 802.11 component which allows detection and triangulation of 802.11 beacons works on a number of PDAs, it is not available on many phones. Worse, the GSM cell tower component appears only to work on a single set of phones, the Nokia N60 series, which do not have 802.11 capabilities. Thus, to use both 802.11 and GSM beacons using *Place Lab*, a Nokia N60 series phone must be paired with a PDA through Bluetooth. The phone then passes the information about detected GSM signals to the

PDA which runs the main *Place Lab* software. This, of course, means that users must both invest in and carry a PDA and phone, linked by Bluetooth, if they wish to use the *Place Lab* functionality fully.

An additional problem with the GSM detection in *Place Lab* is that it is only able to detect one GSM signal at a time. This is described by Otsason et al. [129]:

> *The Place Lab system employed a map built using war-driving software and simple radio model to estimate a cell phone's location with a 100-150 meter accuracy in a city environment. The goal of Place Lab was to provide coarse-grained accuracy with minimal mapping effort. This is different, and complementary to our goal of doing accurate indoor localization given a detailed radio survey. Another distinction is that Place Lab used a cell phone platform that only programmatically exposed the single associated cell tower.*

The relatively low accuracy results, as Otsason et al. state, are based on a reliance on only a single associated tower. Although *Place Lab* relies on 802.11 beacon *triangulation* to achieve high accuracy when using 802.11, it is unable to employ triangulation in GSM as it is only able to detect a single GSM signal at a time. The result is that *Place Lab* is only able to detect the single cell-tower a phone is currently connected to. As phone cells can cover areas of 100m to 3km, this gives GSM positioning which relies on a single cell at best the same accuracy level. Furthermore, as there is no guarantee that the cell tower a phone is currently associated with is the closest, a second level of inaccuracy is introduced.

Place Lab is primarily coded in Java and although Java is intended to be a particularly portable language, the communication that is required between 802.11, GSM and Bluetooth hardware on mobile devices is low-level and cannot usually be achieved through Java alone. Instead, drivers must be coded natively and interprocess communication used to communicate between Java and the native code library. Writing drivers to communicate with every variation of wireless, cell phone or Bluetooth hardware available on the variety of mobile devices is simply not an achievable task and negates the advantages in portability of the sections of code that are in Java. Indeed, *Place Lab* currently only supports Nokia Series 60 cell phones, as these are the only phones for which the phone manufacturers have released a useable API and a driver has been coded. The difficulty of using Java's JNI to perform native process execution is highlighted in [92] and is possibly the primary reason *Place Lab* does not support a substantial amount of mobile hardware.

A final weakness of *Place Lab* is that it does not support a standard protocol through which it may deliver a position to third-party applications. Instead, application developers must design their systems around *Place Lab* from the onset if they desire to utilise it. [104] presents the claim that:

> *Place Lab provides a virtual serial-port interface that can mimic an external GPS unit by emitting NMEA 0183 navigation sentences in the same format generated by real GPS hardware.*

However, this does not appear to be correct. Over a year after this claim was stated the *Place Lab* code does not support the NMEA 0183 protocol. Rather, it is able to parse only two of the five NMEA sentence types (GGA and RMC) and emulate only the single GGA sentence type. Whilst this is enough to supply some web-based mapping applications and extremely basic mobile applications, it is not sufficient to drive the overwhelming majority of existing mobile applications (for example, popular navigation applications such as TomTom and iGo are not supported). As a result, positions calculated by *Place Lab* cannot easily be retroactively fitted to most existing applications as it provides no external delivery to anything other than other Java applications.

These last three points – that *Place Lab* provides poor GSM positioning, does not run on many mobile devices and does not expose the position it calculates to third-party applications – results in *Place Lab* having low availability in many situations. For example, if a user decides to run another process in addition to *Place Lab* on their device, it may consume resources required by *Place Lab* which may result in it no longer being able to deliver positions in a timely fashion. Whilst such problems affect the availability of the system as a whole, the previously discussed fact that an area must be sampled before *Place Lab* can provide any location finding features in the area can hinder its availability in specific areas.

Despite the weaknesses that lead *Place Lab* to be currently impractical as a positioning system for applications on mobile devices, it exemplifies the power of amalgamating different positioning technologies into one system. Furthermore, by providing an implementable interface the system can be quickly adapted to recognise new types of beacon should they become more common in the future. Delivering availability of almost 100% in most areas [104], *Place Lab* far outperforms GPS in outdoor environments whilst continuing to operate quite accurately in most indoor buildings. Although not a suitable solution for most mobile applications for the reasons specified, *Place Lab* does demonstrate that a hybrid positioning system is viable for mobile devices.

5.3 A comprehensive positioning solution for mobile, peer-to-peer devices

Whilst *Place Lab* is a powerful system, it is currently unsuitable for use in applications on mobile devices due to the weaknesses outlined previously. In order to fulfil this need, a new system was designed and implemented at Glasgow by Malcolm Hall and myself.

This system, titled *Navizon*, is similar to *Place Lab* in that at its core is the concept of amalgamating existing positioning technologies, and utilising infrastructure that is already widely deployed in order to calculate positions. It is important to note from the outset that work on Navizon is completely independent from *Place Lab* and that the concepts behind Navizon were arrived at independently. The

initial design and start of implementation of Navizon was conducted before any *Place Lab* research had been published and, although it is impossible to be certain, is likely to have commenced before any work on *Place Lab* had begun. In short, Navizon was developed and implemented at approximately the same time, but completely independently from, *Place Lab*. Malcolm Hall and myself have designed, implemented and maintained Navizon from initial concept through to the commercial application it is today. Even though Navizon is now funded and marketed by Mexens[21], a company based in New York, we remain the sole developers coding, maintaining and augmenting the Navizon system.

The need for an amalgamation of positioning technologies had gradually become clear following the development and evaluation of many systems at Glasgow, and from reading the literature available at that time. The *Lighthouse* had used ultrasonic positioning that was accurate but tied to one location (low availability and high setup cost). *George Square* and *Treasure* utilised GPS that covered large areas but often failed to locate the user for large periods of time during trials if the weather was bad or if users did not stay in open areas (low setup cost but low availability). Our own experiments with the Ekahau system revealed that although it worked indoors, it simply required too much investment in time to carry out the required calibration (high setup cost). Bluetooth positioning was attempted but found to work only on a few devices and be extremely unreliable, due to Bluetooth requiring too long to search for and identify peers in the area. The Bluetooth issues have, of course, been previously discussed in Chapter 4. Through the implementation of previous systems and investigation into alternative positioning technologies it became obvious that only by combining several technologies would a suitably reliable positioning system be achieved.

The primary, overarching goal of Navizon is to provide a positioning system that is available (delivering a location) as often as possible. In order to achieve this, Navizon, like *Place Lab*, relies on several different positioning technologies. Through continual monitoring of each positioning system's performance, Navizon is able to switch to whatever system is likely to give the most accurate position at any given time. Currently, the default Navizon release utilises three positioning technologies: GPS, 802.11 wireless and cell-phone towers. However, the combination of these technologies allows Navizon to achieve accuracy to within a few metres both indoors and outdoors.

Navizon is implemented primarily in C# but some low-level sections that interface with NDIS drivers and read information on cell tower IDs and signal strengths from phones are written in C++. To detect 802.11 access points, Navizon uses the wireless driver discussed in Chapter 4. The driver allows it to scan for and detect every single 802.11 access point that is in range of the device. Navizon vastly improves on *Place Lab* in using GSM cell tower beacon information, as it is able to detect simultaneously and read information about seven cell towers on phones, as opposed to *Place Lab*'s one. As a result, Navizon is far more accurate when relying on cell towers for positioning as it can detect multiple cell beacons from which to triangulate a location, rather than just detecting one and

[21] http://www.mexens.com

basing the position on the tower's location without using triangulation. The technique Navizon employs when using GSM beacons is similar to that described by Otsason et al. [129]. However, Navizon achieves the detection and use of seven GSM signals simultaneously using standard cell phones whilst Otsason et al. relied on an external GSM modem connected to a laptop computer.

Navizon works by continually marking the locations of beacons, currently wireless access points and cell towers, whenever a good GPS signal is available. When a reliable GPS signal is being received the device can estimate the locations of the beacons it detects. The recorded positions of these beacons can later be used to triangulate a client's position when no GPS signal is available, or if it is calculated that a beacon triangulation is currently likely to be more accurate than the one GPS is delivering. A significant difference from *Place Lab* is that the *Navizon* software is a single process that both continually monitors new beacons and provides a location. Thus, there is no need in *Navizon* for separate software to be installed and run, or for an area to be sampled *before* the system can be used.

Navizon also attempts to calculate beacon positions more accurately by employing a simple technique. When a beacon is detected, it is not simply placed at the location of the current GPS signal. Instead, as it is almost impossible that the beacon will be at the current location of the user when it is first detected (it is more likely that this occurs on the edge of the beacon's coverage range), an algorithm is applied in an attempt to position it more accurately. When a beacon is detected, tuples of the current location and the beacon's signal strength are recorded until the beacon is no longer in range. Once the beacon is no longer in range, a log of the path the user took past the beacon and the signal strengths to it will have been created (Figure 18).

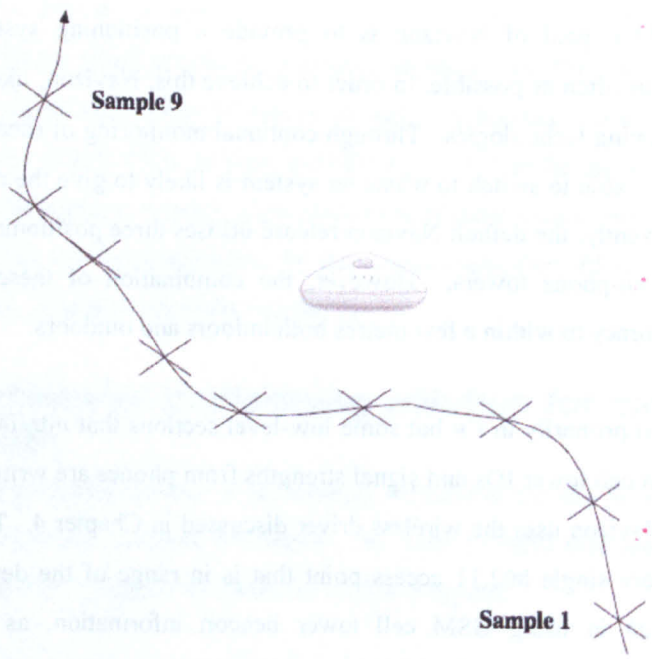

Figure 18: Path of Navizon user passing a beacon. As a Navizon user walks past a beacon, his or her device records many samples of the beacons signal strength. Subsequently, the average of these are calculated to find a location likely to be closest to the beacon. For example, in this case the 9 samples would be averaged leading to a position around the mid-samples being stored. This is far more accurate than relying on any one reading.

Navizon then analyses the log and discards spurious readings, which are mainly those for which GPS accuracy, taken from the HDOP[22] value, is poor. It uses the remaining readings, taking account of the signal strengths, to find the weighted centroid of the samples, which is hopefully close to the actual location of the beacon. In tests this technique has proved to increase accuracy of the discovered beacon by over 4.5 times compared to using the first detected signal and over 2.1 times compared to relying on only the strongest signal (see the appendix for complete trial results). This position is then assigned a quality rating based on the number of samples that were used to calculate the final location and the mean of the signal strengths detected. The database on the local device is then checked, and if less than ten readings already exist for the beacon ID, the new reading is inserted. However, if there are already ten readings in the database then the quality rating of each existing one is considered and, if the new reading is higher, the lowest one of these is replaced.

Whenever a beacon is being used to triangulate the position of the user, the beacon's location is calculated as the weighted average (based on quality) of all the readings currently on the device, according to the following equation. Note that i is the number of readings.

$$location = \frac{\sum_i position_i \times quality_i}{\sum_i quality_i}$$

Later, if the device is synchronised with the server, the same process of removing the reading with the lowest quality rating is repeated if any of the entries on the global server are lower than the new one. The date of a reading is also considered so that an extremely accurate but nevertheless extremely old entry may be replaced by a medium quality reading which is new. This is an important advantage over *Place Lab* as although beacon locations are generally static they may be moved from time to time. *Place Lab* is unable to adapt to such changes, but by using this technique *Navizon* can gradually learn the new locations of any beacons that have been moved.

The process of first calculating the most likely position of the beacon, marking it for quality and only retaining the highest quality readings results in the creation and maintaining of an accurate beacon database. A client using Navizon continually reads from GPS and scans for beacons. If the GPS is working but delivering a poor signal then Navizon switches to the beacon-based positioning that is currently able to deliver the most accurate position. If a user does not have a GPS device, Navizon can still operate by relying purely on the beacon-based location technologies available to it. Similarly, if a

[22] The Horizontal Dilution of Precision is a measurement of the predicted accuracy of a GPS position based on the geometric configuration of GPS satellites currently in view of the GPS receiver. A low HDOP value, such as 2, represents a 'good' reading that was generated from satellites that are distant from one another—spread widely across the sky. A high HDOP value, for example over 20, represents a reading from satellites that are close together. For example, if only 5 satellites were in view and they were all almost directly above the user then a high HDOP value would be calculated. Typically, HDOP values of 6 or lower are said to be 'good', whilst readings of 7 or 8 are 'moderate', and higher values represent 'fair' and 'poor' readings. The maximum allowable HDOP value is 50, and is rarely, if ever, seen in practice.

user owns a PDA that does not have GSM or a phone that does not have 802.11 then Navizon simply deactivates the module for that technology but otherwise continues to work as normal.

The Navizon application maintains a beacon database on the local device but is able to synchronise with a global server if the user desires. If one never chooses to synchronise with the global database then one is only able to utilise beacon data generated by oneself. Thus, such a user would require, at least for a while, a GPS unit that they would use to find beacons. Once this data had been gathered, the system would be able to select from the available beacon systems when GPS was poor or unavailable. The beacon information that Navizon gathers is immediately entered into the database and available for use. So, for example, a new user could travel from home to a local shop using GPS and on the way back remove the GPS unit and rely on 802.11 or GSM positioning for the return journey, as on the outgoing journey Navizon would have mapped and stored the beacon locations. This is significantly different from the *Place Lab* system where one must use a separate program and then upload one's traces to a server before one is able to use the sample data.

When one does synchronise with the server, one can select a region for which one wants to download beacon data by simply drawing a rectangle on a Google Map[23] on the Navizon website. The data is then transferred to one's device and is then immediately available for use. The ability to download this data, which may have been generated by any of Navizon's users, allows a user with no GPS unit at all to use Navizon as a positioning system. If users move to different areas, they are free to simply alter their region and resynchronise their devices to acquire new beacon data for those areas. When users do synchronise with the server, any data they themselves have collected is uploaded and if, as previously described, the readings are of higher quality than existing ones on the global server, then the latter are replaced. The need to synchronise with a central server is, to the author's regret, in opposition to the guidelines stated earlier in this thesis and the author fully believes that a hybrid peer-to-peer and self-generating content architecture would be most suitable for the Navizon system. Unfortunately, however, Navizon is a commercial application and is no longer fully controlled solely by the author. Hence, some aspects of Navizon's use of data adhere to the guidelines—for example, it does use a self-generating content architecture—whilst others, such as the reliance on a central server, do not. Whilst this may affect Navizon's relevance as an example system for the architectural guidelines, it does not affect its technical capabilities as a hybrid positioning system.

Navizon's sharing of gathered data to others creates a community where the majority of users who do not have every supported positioning technology are fed by a small minority who do. Users with GPS units, 802.11 and GSM capabilities map out beacons in an area, and users with only 802.11 or GSM can then use this data through Navizon to position themselves. Navizon supports the ability to use any type of positioning to locate beacons for any other type. For example, a user with 802.11 and GSM may discover a new GSM beacon that is then automatically positioned using the current data from

[23] Google Maps (www.google.com/maps)

802.11 positioning subsystem. This can then be uploaded and shared, allowing users with only GSM to use the beacon when locating themselves.

5.3.1 Using Navizon

Although positioning systems are becoming more common, and devices with inbuilt GPS are starting to emerge, most users are unfamiliar with their use in general. Certainly, the majority of mobile device users experience difficulty when trying to configure a GPS device for the first time. In addition to the positioning technology, many mobile users are lacking fundamental knowledge about their mobile devices, such as how to synchronise them with their computers and how to transfer files to them. Expecting users to be able to comprehend how to download, install, configure and maintain a complicated positioning system themselves is unrealistic. For example, the majority of mobile users lack sufficient knowledge to carry out these steps in order to run the *Place Lab* system.

As there are so many issues with current positioning systems, *Seamful Design* may not be an appropriate solution to the configuration and usage problems experienced. As discussed previously, *Seamful Design* should be selectively applied, highlighting the most important issues to users, rather than applied to every aspect of a system. Such an approach would probably leave the end user overwhelmed, rather than focusing their attention, and drawing out further knowledge, on specific, yet important, issues within the system. Thus, a more suitable approach, in this case, may be to follow traditional techniques, attempting to make the positioning system simply easier to use, and *less* transparent, to users.

This is particularly important in a positioning system, as one that requires extensive configuration, or advanced knowledge, may exhibit reduced availability, as the majority of users may not have the required skills to configure it to perform well. Therefore, to ease the burden on the users and make the system generally more available, one of the goals of Navizon was that it be simple for the end user to install and use.

Navizon is distributed as a single installer package that, when run on a desktop machine, will install to any PocketPC or Windows Mobile device, including PDAs and mobile phones. A version of Navizon is also available for Symbian phone. The single Navizon application performs all of the tasks required to drive the entire Navizon system within a community of users including data gathering, synchronisation and, of course, the positioning of the user. All of this is automated by default although one can, if one desires, opt to turn off the automation (for example, one may choose not to automatically synchronise their data with the server). By default, therefore, a user does not need to concern him or herself with collecting beacon data, as this occurs, when possible, transparently in the background through normal use of the system—regardless of what positioning technologies one's device is equipped with.

Navizon also makes the upload of gathered data and the download of newly available data as simple as possible for the end user. Gathered data is automatically compressed and uploaded to the server either

transparently during normal use through the device's 802.11 capabilities or, if the device has no 802.11 capability, when the user docks their device with their computer. Similarly, as new data becomes available for the area the user is located in or has marked as their predefined area on the website, it is automatically and transparently downloaded and installed during normal use, when a connection is available, in the same manner.

If one wishes to utilise Navizon in a new area, one may easily do so on the Navizon website by dragging a rectangle around the desired area. One's device does not have to be docked during this procedure, as the new data will simply be transmitted to one's device the next time an Internet connection is available. However, Navizon does not use the wireless driver described in the previous chapter to connect automatically to access points. Instead, users must configure the connections they wish to use in the traditional, time consuming, manner. Whilst this is a weakness of Navizon, it is currently thought to be necessary in a commercial product. As previously discussed, the legality of using open access points is questionable, and it would be unwise to build such behaviour into a commercial product that is widely used.

Browsing the entire Navizon location database and marking the area users wish to use Navizon in is provided through a website interface. A map of the entire world is provided through a Google Maps interface, which one can drag and zoom to view any location they desire. Beacons available in the area are overlaid on the map—802.11 and GSM beacons appear with distinctively different icons (Figure 19). This interface allows a new user to determine quickly if an area already has a high coverage (although if it does not they can still use the system, as it will quickly learn the beacons through his or her own use).

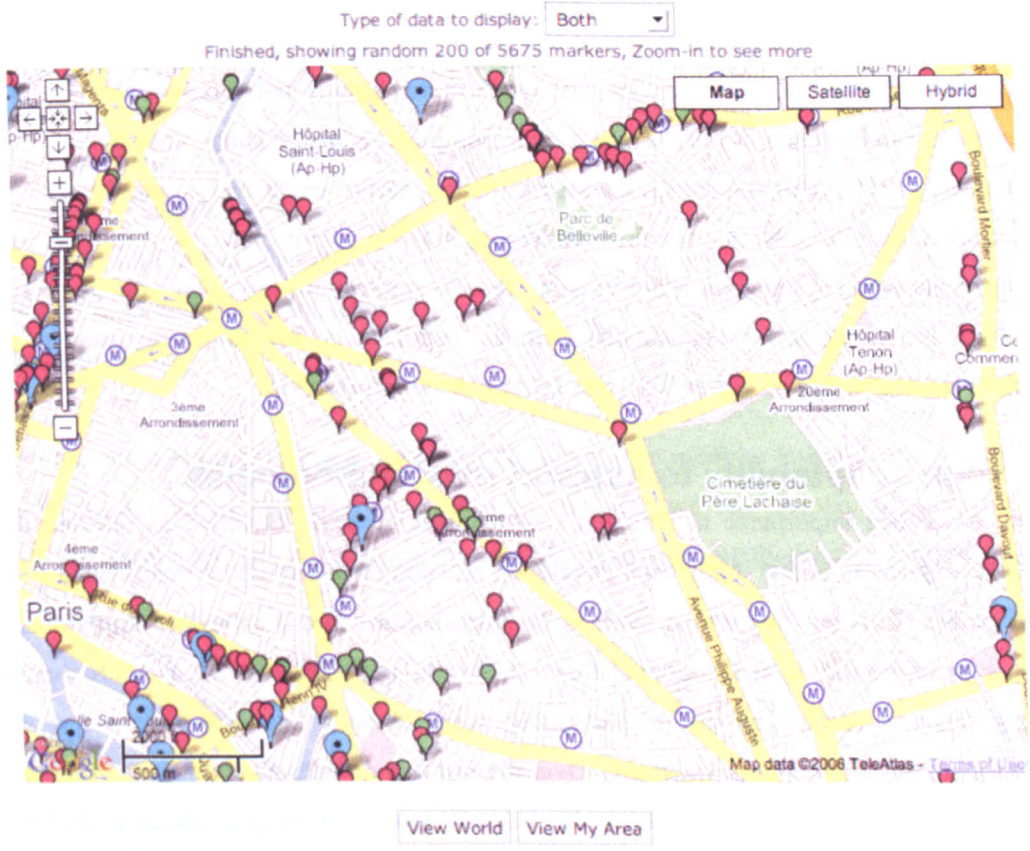

Figure 19: Navizon web interface showing coverage in a small area of Paris. Cell towers are shown as larger blue markers whilst secure and insecure 802.11 access points are shown as smaller green and red markers respectively. Note that for performance reasons in displaying the Google Map only a subset of the true markers are shown. This is clear from the message at the top of the image stating that although there are actually 5675 beacons recorded for the viewable area, only a random 200 of these are shown.

Further information, such as the ID of the beacons (MAC for 802.11 beacons and unique IDs assigned by the mobile phone providers themselves for GSM) are displayed as well as the vendor's ID (the maker of the access point for 802.11 and the network provider for GSM) are available simply by clicking a beacon icon.

Navizon has been designed not as a stand-alone positioning system to run by itself, but as a system that can be used to drive other applications. To achieve this, it is vital that Navizon performs well in two important areas. Firstly, it must not consume too much of the available system resources in order that the majority of resources are instead available to the main application which Navizon is driving. Secondly, as Navizon is designed to replace one positioning system with an amalgamation of many, it must be capable of transparently replacing previous positioning systems. In practice, Navizon consumes approximately 2-10% of the processing time on modern CPUs, and requires only an average of 119 bytes of storage space per beacon. This low resource usage is vital, as it allows sufficient resources for interactive services, such as mapping applications or location-based services, to run, whilst Navizon concurrently runs in the background, delivering the data that drives such services.

In order to allow Navizon to replace transparently existing positioning technologies, it can create a virtual serial port, to which it writes position data correctly formatted to conform to the NMEA-0183

standard. Navizon fully emulates NMEA GGA, GLL, GSV, GSA, VTG and RMC sentences. These sentences contain information on latitude and longitude, time, velocity, orientation, number of satellites in view, signal information from satellites, and much more (a short, but concise overview, of NMEA-0183 sentences can be found in [49], whilst a more complete reference is the NMEAs own publication on the standard [125]). This standard is the most common, almost exclusively used, format for position data on modern mobile devices. By fully emulating NMEA output, Navizon is able to replace the majority of positioning systems on which existing mobile applications rely. For example, Navizon can replace GPS device input for applications such as TomTom or iGo, allowing users to run such applications relying purely on either 802.11 or GSM beacon information.

5.3.2 High adaptability to support new technologies

A feature of critical importance to Navizon's future development, identified as relevant to mobile applications in general, is that it has been developed in a completely modular manner and is therefore easily adaptable. This is of high importance as the infrastructure that is embedded throughout a single building or an entire city is in a continual flux. A decade ago, GPS was not publicly available and positioning systems useable by the general public were virtually non-existent; the few that were available were confined to indoor locations. Over the last two decades an abundance of mobile phone cell towers have been placed throughout the western world, creating a massive change in the technological infrastructure of modern cities. For example, the Navizon database reveals 545 cell antennae in a single square kilometre in the Manhattan area of New York, and this density is typical of most major Western cities around the globe[24]. Similarly, throughout the last decade an even higher number of 802.11 access points have been sited, and it is now almost impossible to find any street in large cities, such as New York, where a signal from at least one wireless access point can not be detected. For example, the same square kilometre in Manhattan contains 14953 access points. More recently, Bluetooth devices have become common for the average user and it is now equally difficult to find any location in a busy pedestrian street where there is an absence of Bluetooth signals.

The almost blanket placement of these devices—the infrastructure which feeds the modern city's data requirements—is what allows Navizon to use beacon detection to locate users accurately. However, as the infrastructure and the type of technology used is in continual flux, it is vital that Navizon be prepared to integrate quickly any new technology should it become popular. For example, Bluetooth has emerged as a common technology on mobile devices themselves but is not yet frequently used on static devices due to the relatively long connection time required for devices to pair. However, if future versions of Bluetooth do not have this limitation, it is likely that static devices utilising it would become common. Therefore, if this occurs it would be prudent to add Bluetooth to the beacon types Navizon can use.

[24] The Navizon database can be browsed to view beacons and check densities at http://my.navizon.com. The area the figures cited here is the square kilometre running from approximately the intersection of 48th Street and 10th Avenue to the intersection of 39th street and 3rd Avenue. Note that mobile phone cell towers typically contain between 4 and 10 antennae, and the actual number of cell *towers* in this area is likely to be 545 divided by this number.

To support rapid adaptation and integration with future technologies, Navizon is designed in a modular manner at all levels. Navizon consists of five main areas:-

- User interface
- Positioning system input
- Location filtering and calculation
- Data storage and synchronisation
- Position output

All of these five main areas are loosely coupled, but each is also modular in itself. For example, each possible position system is implemented in its own module, and any number of modules can be connected to the system and run simultaneously, with Navizon selecting the best performing one at any time (Figure 20).

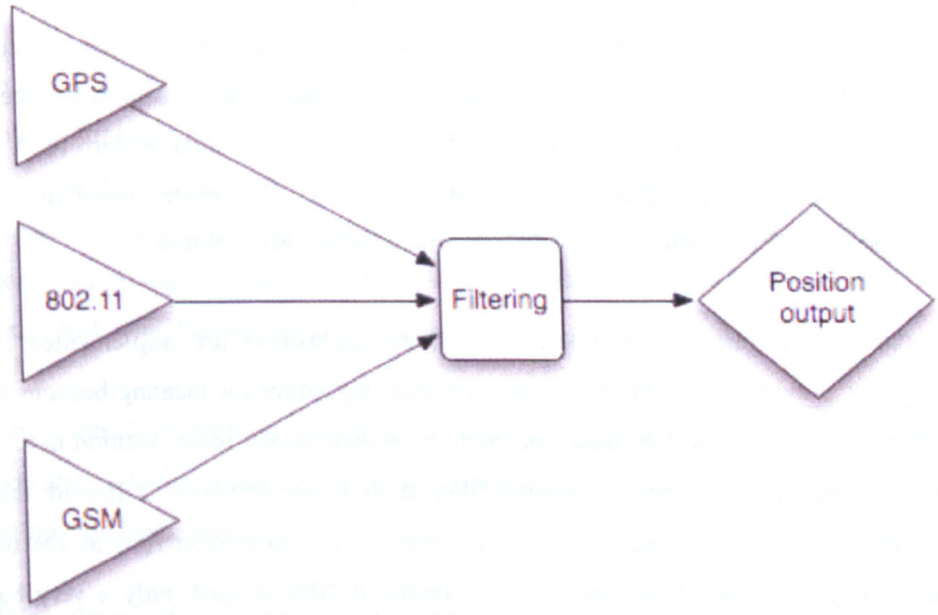

Figure 20: Navizon positioning modules. Only one of the three positioning modules must be active for Navizon to provide a position.

The position output from Navizon currently only comes from one of the three possible positioning modules; GPS, 802.11 or GSM. The decision of which of these to use is based on some simple limits. Assuming all three are available on the device, if the current GPS HDOP value is 6 or lower then GPS will always be used. For HDOP values between 6 and 10, 802.11 will be used if more than three access points are in view, otherwise the GPS position will still be used. If the GPS HDOP value is higher than 10, then any position delivered from 802.11 will override the GPS position. GSM positioning is only used when both GPS and 802.11 are unable to deliver any position (there are not enough GPS satellites to get any position and no known 802.11 access points are detected). Table 5 details where Navizon's final position will come from in each situation.

HDOP / APs	<= 6	7 - 9	>= 10	GPS invalid
0	GPS	GPS	GPS	GSM
1 - 2	GPS	GPS	802.11	802.11
>= 3	GPS	802.11	802.11	802.11

Table 5: Table of which underlying positioning subsystem Navizon uses to calculate a final user position, based on the HDOP value of the GPS and the number of APs currently detected.

The use of limits to determine which underlying technology to use is relatively simple, and it is likely Navizon's final position accuracy could be improved further if sensor fusion techniques were employed—specifically, complimentary sensor fusion of the type Brooks and Iyengar describe seems appropriate [18]. For example, rather than relying solely on GPS in the case where both the GPS and 802.11 subsystems are providing good accuracy, the two readings could be combined to provide greater overall accuracy. However, to date, such algorithms have not been implemented. This is mainly due to the fact that early versions of Navizon were relatively processor intensive, and so during the initial design sensor fusion was dismissed, as it would have added further processing requirements to an already processor-heavy task. However, Navizon has been greatly optimised since its initial versions, to the point that it typically uses less than 5% of the CPU time on a modern PDA such as an iPAQ 2750 (with an Intel PXA270 processor clocked at 624MHz). Therefore, sensor fusion is now a highly suitable candidate for improving Navizon's accuracy further in the future.

Just as with the three positioning modules, the filtering algorithms are implemented in distinct modules. Currently, Navizon relies primarily on a centroid algorithm for locating beacons but this is simply a module, and the raw location data can easily be routed to any other position module, which may implement an alternative algorithm. Several filter modules are available to smooth the location data Navizon generates, and these are hot-swapped—that is, the application runs as the underlying positioning technology being used switches. For example, if GPS is used, only a very basic filter which smoothes large jumps in location is employed, whilst under 802.11 positioning a more complicated filter is switched to, which takes account of the last four positions and calculates an expected movement vector; this is then used to smooth the position Navizon subsequently delivers.

Again, the filtering modules are currently relatively basic—simply predicting future movement based on the current vector, and averaging this calculated vector with the current position whenever the position is substantially different from what the vector would suggest. Just as with sensor fusion, during the design process alternative techniques were ruled out due to excessive processor usage. However, since Navizon now consumes far less processor time, alternative techniques can be considered to improve further the delivered position. One such technique could be the use of Gaussian process models, which are indeed likely to provide more accurate positioning. Chen et al. show that Gaussian processes delivered higher accuracy than both centroid and fingerprinting techniques for GSM positioning in five of the six scenarios they tested [34]. Schwaighofer et al. implement a

Gaussian process-based positioning technique using DECT[25] networks [150]. Whilst they find their implementation achieves an average error of around 10m after calibration with 100 points, their work also demonstrates several reasons why Gaussian models might not yet be suitable for use with Navizon. Firstly, Gaussian models generally rely on information about how radio signals propagate from the base stations, and a model, or calibration data, of the area around each access point. Navizon is used with an extremely large number of PDA and phone models, and the base stations it uses to triangulate data can be any model from any manufacturer. It would be almost impossible to create a database of all possible radio propagations from each aerial of each type of access point that might be encountered. Furthermore, storing such data (propagation models) for such a vast range of PDAs, phones and base stations is simply not a viable option given the limited storage space available on most mobile devices. Secondly, Navizon is a system designed to be used in any area without any specific bootstrapping by the user. This directly conflicts with the calibration data that normally must be collected prior to the use of Gaussian process models. Although Navizon does attempt to locate 802.11 and GSM base stations automatically, it cannot achieve the high accuracy in this process that is required for such techniques to be used reliably.

Each of the five main areas within Navizon's structure contain modules that can easily be swapped in and out allowing for rapid production of new interfaces and inclusion of new technologies. Indeed, the strength of a modular code design in a mobile system has already been confirmed in Navizon's own development cycles, as it has allowed the addition of new filtering modules to be quickly implemented and swapped—improving the quality of the delivered location—and for a mobile phone version to be rapidly created following the initial PDA version.

Finally, to aid other developers in integrating Navizon into their applications, a comprehensive API is publicly available, which includes methods for connecting to the Navizon database, detecting beacons and generating data from them.

5.3.3 Sharing of information between users

The design of Navizon was strongly influenced by many of the findings from the systems discussed previously. Initially, Navizon was designed as a pure peer-to-peer application with a self-generating data model. Although a central dependency on a global server was later added to make Navizon a commercially secure product, the benefits gained from the initial design remain. In particular, the self-generating architecture, in which new data is generated through the normal use of the system, is thoroughly embedded in Navizon.

The initial design of Navizon was as follows. The application itself would generate the entirety of the data that is required to calculate positions through detecting and calculating beacon positions. Although a new beacon can be positioned when the current location itself is being derived from other

[25] Digital Enhanced Cordless Telecommunications are, as the name implies, primarily used to provide networks for cordless telephones. It is now common for large office blocks to have tens or hundreds of DECT base stations spread throughout the building, much in the same way as 802.11 access points. Similar to 802.11 clients detecting 802.11 access points, DECT clients can detect the unique identifiers and signal strengths of DECT base stations currently in range.

beacons (for example, a cell tower beacon can be calculated from positions generated by the 802.11 beacon module), the concept was that a minority of owners with devices containng GPS capabilities would aid in essentially war-driving, or bootstrapping, a much larger community of 802.11 and GSM devices. Areas would be mapped out by GPS users and the data they collected would be automatically distributed to other users through a peer-to-peer network. As no beacon data is ever generated from an external source (beacon data is only created by the application itself) Navizon is a completely self-generating data system. Initial development versions of Navizon were able to propagate the data they generated directly to peers, and had no dependency on any external server. The underlying idea was that as users moved from one location to another, they would encounter peers and continually gain new data on the local area, both by generating it themselves and by receiving it from peer devices.

Clearly, this design is extremely powerful as it allows Navizon to be introduced into a new community of users who all contribute to generating data that is distributed for the benefit of the entire community. However, Navizon was subsequently adapted to a commercial product and, as the new owner[26] desired the maximum level of control over the generated beacon data as it is believed to be of high commercial value, it was decided that it should be controlled centrally. Obviously, as described, the decision to hold the data centrally introduces a number of negative aspects previously outlined.

As it is clear that a decentralised architecture is the most desirable type for mobile systems, attempts were made to make the central server, which was required by the new owner, as transparent to the process as possible. It was hoped that by minimising the role of the central server, the negative effects introduced by its use would also be minimised. When new beacon data is generated, Navizon caches it as it did in the decentralised version. However, instead of automatically sending the data to peers when they are detected, it instead uploads it to the server when a connection to it is available. Unfortunately, this of course means that a connection to the central server is now a requirement if up-to-date data is to be supplied. Although one may achieve this connection through docking with a computer, wireless 802.11 or GPRS, all these have convenience, configuration, legal or cost implications—as has been previously discussed.

This centralised architecture admittedly contradicts the previous findings and guidelines in this thesis, which suggest the initial pure peer-to-peer architecture, or a hybrid one, would be superior. The author does believe that this is true, and that an alternative architecture would be more suitable. Whilst this constraint is unfortunate, it in no way minimises the power that the hybrid *positioning* solution of Navizon provides, and Navizon makes a substantial contribution to the mobile field in providing a positioning system with greater availability and ease of use than any previous technique has managed to achieve.

[26] Whilst Navizon is now funded and marketed by Mexens, Malcolm Hall and myself remain the sole developers of the Navizon application. To date, all code in Navizon has been implemented by Malcolm Hall or myself, and we continue to maintain and develop it.

5.4 Performance

Several tests have been conducted to estimate Navizon's performance when using 802.11 and GSM beacons to locate the device. Firstly, Navizon was run on a device for 10 minutes at three locations, and the accuracy of each of the three positioning types was measured from the readings gathered. Secondly, three walks were conducted, during which a device running Navizon was carried. The data gathered from these walks allows for an overview of how accurately the location data generated by each of the three positioning types aligned with the actual route followed.

As Navizon has been widely used, a substantial amount of data on 802.11 and GSM positions have been collected from the public's use, and most major cities in the USA and the UK have many beacon locations recorded. Therefore, as the majority of the trials were conducted in one of these cities, Glasgow, it was not normally necessary for the area to be sampled (wardriving) prior to the trials. Instead, before the trials were run, the device running Navizon was synchronised with the global Navizon server, which contains the beacon data gathered by many Navizon users. Undoubtedly, a portion of this data would have been contributed by the author through the normal use of Navizon in the area. However, this would have occurred through normal, daily use of Navizon and was not conducted with the trial in mind. The one exception was a walk that was carried out in Erskine, a small suburban area approximately 15 miles west of Glasgow. As it was not believed that this area would have been extensively mapped prior to the trial, a 30 minute war-drive of the area was completed before the walk was commenced. The war-drive was conducted with the use of a car, and involved driving along the roads surrounding the route that was to be walked.

For the first set of trials the accuracy of Navizon's three positioning methods were measured at three locations in the west-end of Glasgow. For each location an iMate SP5 phone running Navizon was placed on the ground and left to log data for 10 minutes. Placing the device on the ground ensured that it was not moved for the duration of the trial. The actual location coordinates (latitude and longitude) the tests were conducted at were determined through the use of a geo-referenced OSGB map. To verify the coordinates gained from the geo-referenced OSGB map they were compared to the coordinates returned from Google Earth for what was believed to be the same location. For all three locations the coordinates were found to have no greater than a 0.00003 difference, which represents approximately 2 metres for the areas tested in, and is believed to be acceptable.

Measuring the accuracy of any positioning system is always challenging, as there is no completely reliable method to provide a true, fully accurate position to compare the system being tested against. Unfortunately, although it is extremely unlikely the error of this method is greater than 5 metres, the exact error cannot be determined. Whilst this introduction of error is unfortunate, this problem of finding a completely reliable measurement of position to compare a system against is commonly experienced by researchers examining positioning systems, and is not unique to this research or to these trials. LaMarca et al. employ a different method when measuring accuracy in PlaceLab. They rely on averaging the positions delivered by two independent D-GPS devices [104]. However, as GPS

is one of the positioning systems being trialled, this was not believed to be a suitable technique for use here, as it would likely weight the results in favour of GPS.

During the trial, the device logged the latitude and longitude coordinates generated by Navizon's GPS, 802.11 and GSM modules, all of which were concurrently active and generating location data throughout the trial. The following three figures (Figure 21, Figure 22, Figure 23) plot the locations generated by each of the three Navizon modules during the trials. In each figure the actual location the device was placed at is marked with a white square. GPS results are shown as smaller blue squares, 802.11 as green, and GSM as red. Note that the maps shown in each figure are shown at different scales—a scale bar is included at the bottom left of each to help gauge distances.

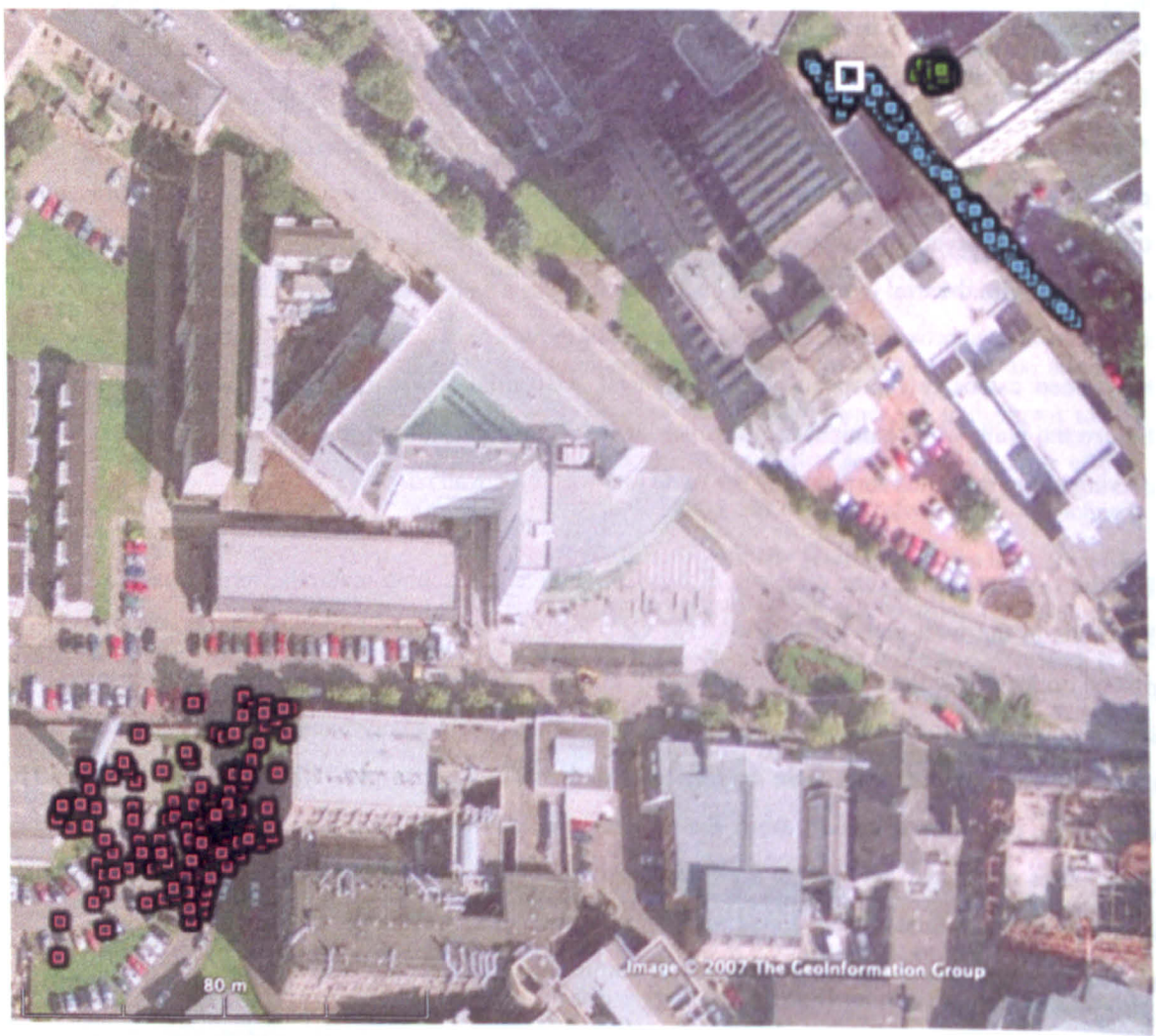

Figure 21: Map of locations determined by Navizon's three positioning techniques (actual location is white square, GPS in blue, 802.11 in green, GSM in red) for location 1 (55.873784, -4.29227)

Figure 22: Map of locations determined by Navizon's three positioning techniques (actual location is white square, GPS in blue, 802.11 in green, GSM in red) for location 2 (55.872978, -4.289858)

Figure 23: Map of locations determined by Navizon's three positioning techniques (actual location is white square, GPS in blue, 802.11 in green, GSM in red) for location 3 (55.87443, -4.288545)

After the results were gathered, the NMEA data generated by both the GPS unit and Navizon was parsed to extract the sexagesimal latitude and longitude values. These values were subsequently converted from sexagesimal values to decimal latitude and longitude. In order to measure the distances between multiple decimal latitude and longitude coordinates, they were further converted to OSGB tranverse mercator coordinates using the Airy Spheroid constants[27]. The code used for completing this conversion is shown in the Appendix. Once in OSGB coordinates, the distance between two sets of coordinates can be easily calculated using the traditional Cartesian distance formula, as OSGB coordinates represent a square grid with each unit being exactly one metre. A simple C# program was implemented to complete the conversions for all the trial results and to calculate the distances of each from the known actual location. Finally, from these distances the maximum and minimum errors, mean, variance and standard deviation from the recorded actual location were calculated. The results are shown in the following tables (Table 6, Table 7, Table 8). Note that all result values have been rounded to two decimal places.

	Minimum error (metres)	Maximum error (metres)	Mean (metres)	Variance (metres)	Standard deviation (metres)
GPS	0.63	64.13	12.83	216.55	14.72
802.11	13.21	20.36	17.75	1.11	1.05
GSM	164.72	231.19	195.00	136.87	11.70

Table 6: Minimum and maximum levels of error, mean, variance and standard deviation, for positions determined by Navizon for location 1.

	Minimum error (metres)	Maximum error (metres)	Mean (metres)	Variance (metres)	Standard deviation (metres)
GPS	2.32	9.23	6.58	3.18	1.78
802.11	3.38	15.08	13.70	4.97	2.23
GSM	205.15	315.23	259.58	220.15	14.84

Table 7: Minimum and maximum levels of error, mean, variance and standard deviation, for positions determined by Navizon for location 2.

	Minimum error (metres)	Maximum error (metres)	Mean (metres)	Variance (metres)	Standard deviation (metres)
GPS	10.23	17.81	11.41	1.33	1.15
802.11	6.36	27.15	21.19	6.18	2.47
GSM	52.09	140.41	87.39	447.08	21.14

Table 8: Minimum and maximum levels of error, mean, variance and standard deviation, for positions determined by Navizon for location 3.

[27] The Airy Spheroid is an approximation of the curvature of the Earth for Great Britain. It was defined in 1830 by the then British Astronomer Royal, George Biddell Airy, and has since become the standard ellipsoid used for Great Britain.

It is clear from the results in the tables that both GPS and 802.11 prove both more accurate and more reliable than GSM. However, surprisingly, it is also clear that 802.11 may often outperform GPS positioning in both accuracy and reliability. The results from location 1 show that although 802.11s minimum error is lower than that of GPS, 802.11 does in general deliver a more accurate and reliable position, and has a substantially lower maximum error. For the two other locations GPS and 802.11 performance is extremely similar. In all trials and measurements GSM's performance was the worst. However, its maximum error in all trials is 315.23 metres, which is still accurate enough for many coarser-grained location-based services. Furthermore, although GSM's standard deviation is higher than GPS and 802.11, it still proves to be of relatively high reliability. As position information from GSM is available almost 100% of the time—far outperforming both GPS and 802.11 in this aspect—it can prove more useful in many scenarios despite its lower accuracy.

The second trial conducted involved following a known route whilst walking with a device running Navizon. All the routes begin and end in the same location, and as viewed from the maps, were traversed in an anti-clockwise direction. The device recorded all the positions calculated by Navizon's GPS, 802.11 and GSM modules throughout the walk. This process was repeated for three routes, two around the University of Glasgow and one in Erskine. As there are many points generated, each individual set of GPS, 802.11 or GSM points are displayed on their own map in the following figures in order to present the findings clearly and avoid confusion. A final figure in each set displays the final locations output from Navizon after it selected which underlying positioning system it calculated most to be performing best at the time. The first set of figures (Figure 24, Figure 25, Figure 26, Figure 27), show the results from the Erskine walk. As with the maps from the first set of trials, note that each map is shown at a different scale, and that scale bars are provided at the bottom-left of each map to aid in gauging distances.

Figure 24: Map showing GPS positions recorded whilst walking route marked yellow in Erskine.

Figure 25: Map showing 802.11 positions recorded whilst walking route marked yellow in Erskine

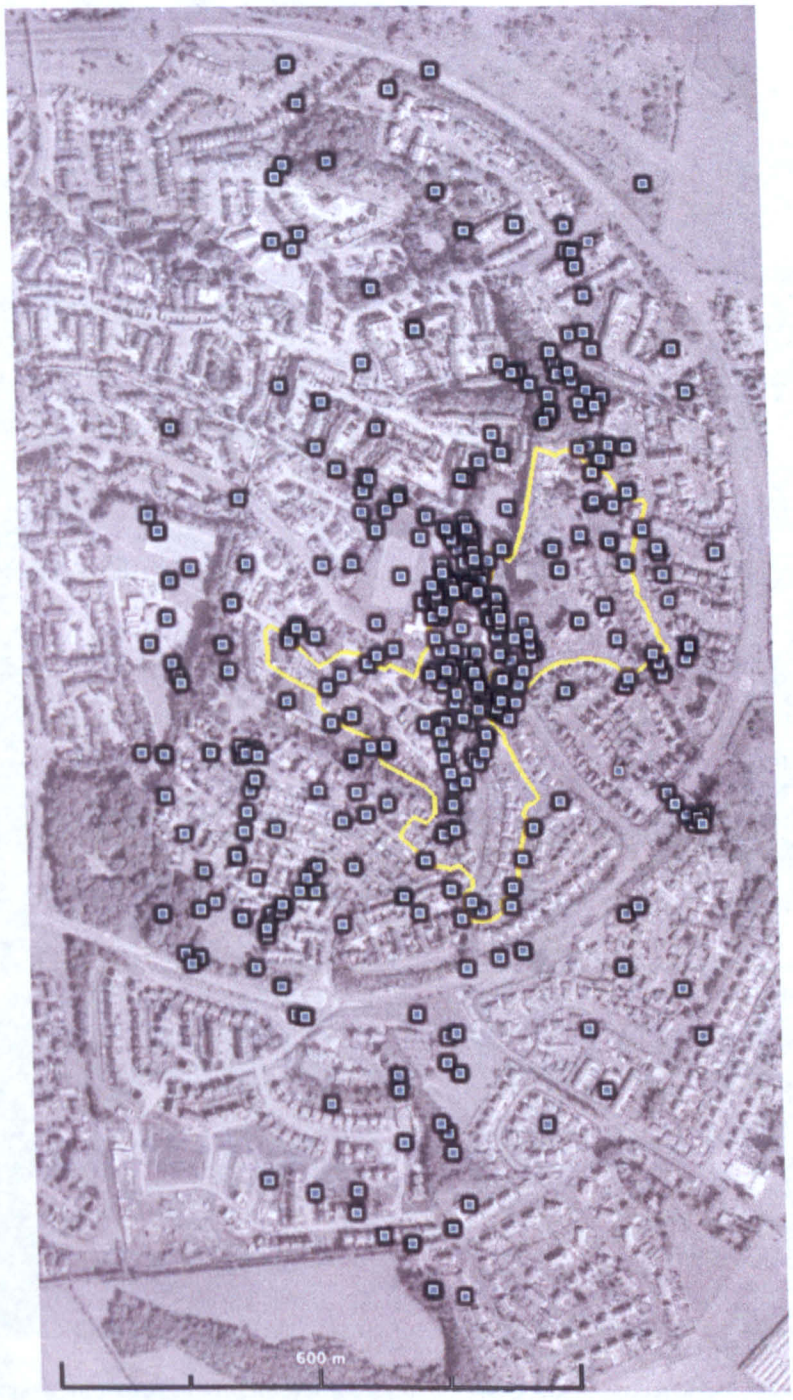

Figure 26: Map showing GSM positions recorded whilst walking route marked yellow in Erskine.

Figure 27: Final locations output from Navizon for route marked in yellow in Erskine.

The next set of three images (Figure 28, Figure 29, Figure 30, Figure 31) show the results from the first route around the University of Glasgow in the west end of Glasgow.

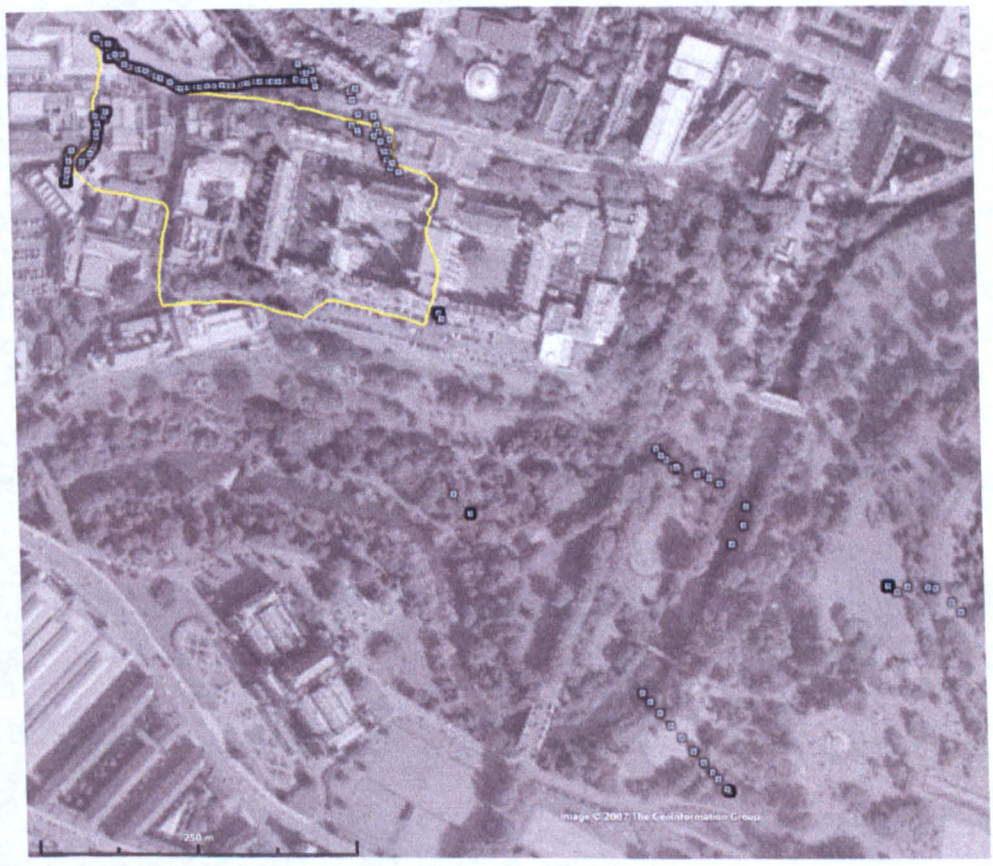

Figure 28: Map showing GPS positions recorded whilst walking the first route in Glasgow (marked in yellow).

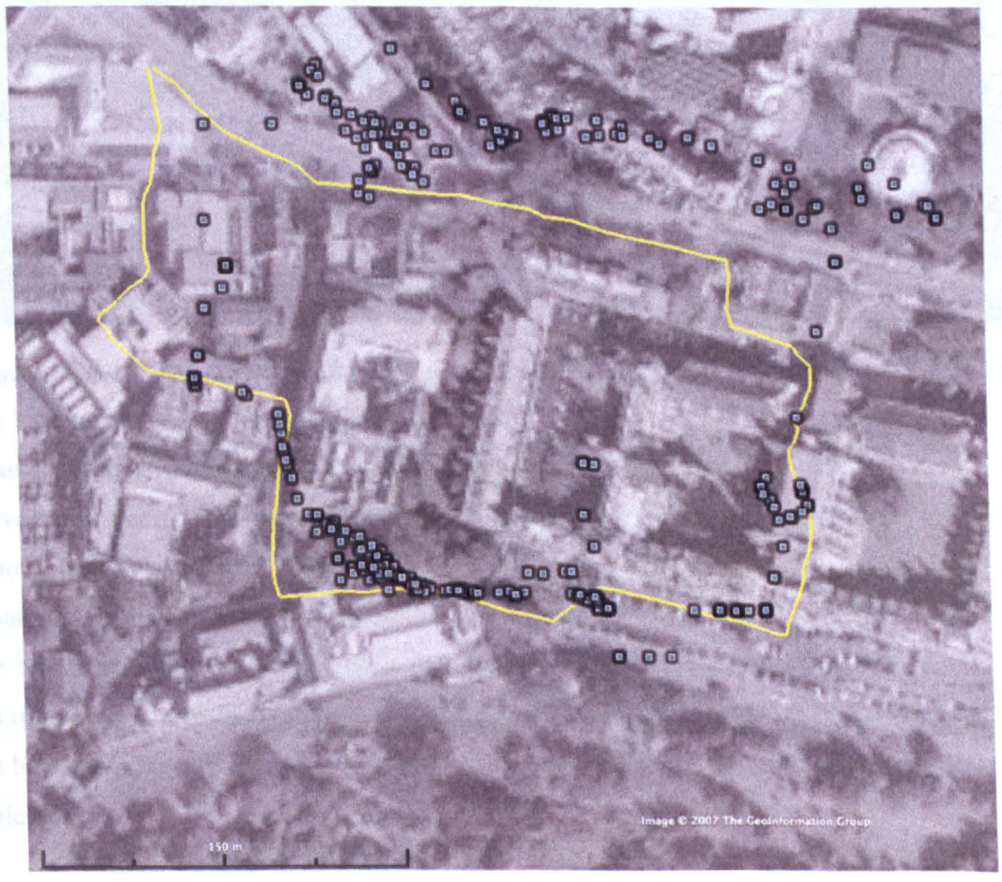

Figure 29: Map showing 802.11 positions recorded whilst walking the first route in Glasgow (marked in yellow).

Figure 30: Map showing GSM positions recorded whilst walking the first route in Glasgow (marked in yellow).

Figure 31: Final locations output from Navizon for first route in Glasgow, marked in yellow.

The results from this first Glasgow walk proved surprising, as the GPS readings were extremely poor. The walk was started at the very upper-left corner of the route and followed in anti-clockwise direction. The walk soon entered a building and, as could be expected, GPS lost the position. However, the journey through the building was short, and the vast majority of the southern edge of the route was outside in a relatively open road surrounded by low buildings or, at many parts, no buildings at all. This segment of the journey took over 5 minutes, and the GPS appears to have performed extremely poorly during this time. For several minutes the GPS device did not provide any position at all, or provided positions that were approximately half a kilometre south-east of the actual location. Ironically, the GPS device appears to have begun providing more accurate positions just moments before the route turns north and reenters another building.

For this particular route, 802.11 positioning appears to have substantially outperformed GPS. This perhaps highlights one of the ways in which 802.11 aids positioning in general, and why a hybrid positioning system is more powerful than systems that rely on only one technique. It is clear that GPS can prove more accurate if users are primarily travelling outside in open locations. However, 802.11 is often more useful if parts of the route are indoors. Not only can 802.11 provide a location indoors, it also continues to be immediately available once a user returns outside, whilst GPS can take many minutes before providing reliable positioning once more. Note that in general 802.11 is quite accurate for this route (the 802.11 map is magnified to a greater level than the others in this walk, and the scale bar can be used to verify that the accuracy is comparable to GPS for the parts of the route where GPS is available).

The final set of map images shown in the following figures (Figure 32, Figure 33, Figure 34, Figure 35) and display results from the second route around the University of Glasgow.

Figure 32: Map showing GPS positions recorded whilst walking the second route in Glasgow (marked in yellow).

Figure 33: Map showing 802.11 positions recorded whilst walking the second route in Glasgow (marked in yellow).

Figure 34: Map showing GSM positions recorded whilst walking the second route in Glasgow (marked in yellow).

Figure 35: Final locations output from Navizon for second route in Glasgow, marked in yellow.

As with the first Glasgow route, there are areas of the route for which GPS did not provide a position as it was indoors, or was a segment following an indoor area during which the GPS device did not yet have a satellite fix. In these areas 802.11 did provide a position, and seems to have been generally as accurate indoors as it was outdoors. For the areas for which GPS and 802.11 provided positions, the accuracy again seems to be generally similar between the two; although it is clear that GPS does achieve a higher maximum accuracy when in optimum conditions. Although showing maximum errors of 350-400 metres from the route, GSM appears to provide approximately block level accuracy or better in general.

One unavoidable limitation of the trials was that they were limited in area, mainly being conducted in Glasgow and the surrounding area. Any positioning technology that relies on beacons is obviously affected by the density of these beacons. The density of 802.11 and GSM beacons varies from one country to another, from city to city, and even from regions within a single city. Again, simple browsing of the Navizon database[28] reveals substantial variations from one area to the next. Furthermore, as Trevisani and Vitaletti point out, there are marked variations between urban and suburban areas in general [164]. Therefore, due to this constraint, the results from the trial are, literally, not globally applicable, and should be viewed, at best, as typical of the performance within a large Western city.

[28] http://my.navizon.com

In addition to these trials a simple, shorter trial was conducted to find the availability of each positioning technology Navizon relies upon during a typical shopping trip in the centre of town. The solution Navizon provides is perhaps most apt for use in dense urban areas, and so the selection of a shopping trip in the centre of a large city was thought appropriate, as outside dense urban areas, such as in suburban or countryside areas, GPS availability is generally extremely high, and a solution such as Navizon is not a necessity if mobile applications are to be used.

An iMate SP5 running a slightly edited version of Navizon[29], and connected via Bluetooth to a small GPS unit, were given to a trial participant to carry with her during a normal shopping trip with her friends. The iMate had both its GSM and 802.11 capabilities activated, and Navizon was able to use all three of the underlying positioning technologies it relies on. The version of Navizon had been slightly edited to log counters of the total time it depended on each of the three positioning technologies, and the time when none of them were producing a result. As with the previous accuracy trials, the same area constraint applies, and again the results can only be viewed as typical of a large Western city, and not applied to other types of area.

The subject spent just under 4 hours on her shopping trip, the majority of it spent in shops in or around Buchannan Street and Argyle Street in Glasgow. The percentage of the trip for which each system was available and producing a valid location is shown in Table 9 (note that timing information from the same trial is also available in the appendix).

Technology	802.11	GSM	GPS	Any
Availability (%)	68.35	99.83	23.20	99.85

Table 9: Availability of each positioning technology Navizon uses during a normal shopping trip in town.

Whilst the positions delivered by GSM are less accurate than GPS, the availability of this system in urban and indoor areas is far higher. Furthemore, 802.11 has been shown to be only slightly less accurate than GPS outdoors, and it is significantly more available. Using Navizon, the trial participant had a valid location throughout almost the entire duration of her trip, compared to only around 23% for GPS.

Whilst the accuracy of Navizon using 802.11—commonly under 20 metres and at worst around 27 metres—is higher than that reported by LaMarca et al.'s *Place Lab*, 31 metres [105], it would be unwise to conclude based on this data alone that Navizon is generally superior in accuracy. As has been stated, the density of access points and cell towers varies greatly between areas, and as both Navizon and *Place Lab*'s performance is directly dependent on this density, any testing performed in different regions should not be compared. Certainly, it is entirely possible that if *Place Lab* were used in the same regions, and had recorded the same beacons, it would perform as well or better.

[29] The version used in the trials was edited to output debug information about the locations it was calculating and which positioning technique it was currently using to a comma separated file. This was done simply to allow for the fast transfer of this information into spreadsheets used to create and calculate the tables presented here.

It is clear that one application area for which Navizon is not suitable is the accurate positioning—room level or below—of users or devices in indoor environments. Whilst Navizon does continue to operate in indoor locations, it must use 802.11 or GSM beacon positioning, which is not currently capable of delivering sub-metre accuracy. Clearly, for applications in which centimetre accuracy is desirable—for example, to track a device within a single room, as was achieved with high accuracy with the Active Bat [171]—Navizon does not deliver a suitable solution.

5.5 Conclusion

As has been previously stated, a user's location is the most important piece of contextual information for mobile systems. Navizon may aid mobile, context aware applications, by making location information *constantly* available. In this way, Navizon is an enabling technology, allowing novel mobile applications to be created which rely on the location data Navizon produces. Prior to Navizon, the majority of mobile applications relied on GPS alone and, as has been seen, this supplies extremely poor availability in urban and indoor areas. Navizon is able to deliver a location, accurate to under 100 metres, both indoors and outdoors, almost 100% of the time.

Navizon fulfils the goals set out at the start of this Chapter. It provides a position both indoors and outdoors virtually constantly with a relatively high degree of accuracy. This, and the fact that it runs on an extremely large number of mobile devices, makes it highly available to end users, and designers who wish to develop context aware mobile applications. As there is no need to sample any area before Navizon can be used, and as it requires only standard hardware sold on most mobile devices, the system requires almost no set up time or cost. As a self-generating content system, Navizon builds the information it relies on through normal use of the system. Furthermore, as the system is always transparently detecting and recording new beacon locations, it requires no conscious effort in specifically sampling an area.

Work on the Navizon technology was completed in 2004, and since 2005 Navizon has been commercially available from an American company called Mexens. Navizon has undoubtedly proved useful in, and relevant to, the mobile field as it is extremely successful commercially. At the time of writing there are over ten thousand Navizon users, and data on over five million beacons currently held in the database. In addition, several logistics companies in America and Europe now rely on Navizon as a positioning system for parts of their fleet tracking.

It is hoped that the main use of Navizon in the future will indeed be as an enabling technology for context aware mobile applications. New versions of Navizon already permit a user location to be continually tracked and made available to others through a website, and allow users to search for services such as restaurants or entertainment services in the local area when they are mobile. Hopefully, these are just the first mobile applications of many that will rely on Navizon to provide

constant positioning within their systems. Future applications using Navizon are discussed further in the Future Work section of Chapter 8.

Additionally, Navizon's 802.11 location technique has also been made available through a Java applet, which allows 802.11 positioning to be carried out when a website is viewed on a laptop computer. The location information provided can then be used to filter or verify information made available through the website. For example, in the USA, Major League Baseball is experimenting with receiving a position from the Navizon Java applet in this way to detect where users are, and then allowing them to view games involving their local team online and directing them to nearby ticket offices.

As with the outcome of Chapter 4, Navizon also aids in making mobile applications fit more accurately Weiser's vision of ubiquitous computing, in which the technology becomes invisible to the user, who becomes free to concentrate on his or her main task—rather than being forced to deal explicitly with the underlying technologies required to achieve that task. With Navizon, location information itself is made more ubiquitous. In prior systems location was only available periodically, and users had to be continually aware of how the positioning system was performing. In cases where the system was required but not available, users may have had to alter their own plans in order to move to a location where positioning information was available, or spend time better configuring a poorly performing system. This is the problem Messter et al. described, in which users are forced to detach from their current context and task in order to remain nomadic, and continue using a mobile system [116]. As Navizon provides a location both indoors and outdoors almost continually, location information can be assumed to be constantly available, and users have more freedom to continue with their own tasks, rather than having to work around a poorly performing system.

6 DISTRIBUTING DATA IN MOBILE, PEER-TO-PEER APPLICATIONS

At the end of Chapter 3, one of the pieces of infrastructure required for mobile, peer-to-peer applications was:

- A method for providing and distributing data within a peer-to-peer community

Mobile peer-to-peer communities are more fluid and transient than desktop communities in general, and even than peer-to-peer communities that run within desktop environments. Since the network topology within mobile, peer-to-peer communities rarely remains static for any length of time, and as clients can leave the community at any time and may never rejoin, there are unique challenges in routing data from one client to another.

The difficulty of routing data from one device to another is compounded by the fact that mobile devices have a relatively small amount of storage space. Thus, not only must data be routed in novel ways, once data arrives on a device, a decision must be made on whether to store it and, if so, for how long.

This chapter explores these two issues: firstly, routing data within a mobile, peer-to-peer community and secondly, deciding when to store and when to discard data. As these topics are extremely broad the concepts discussed are grounded in two applications, *FarCry* and *Samara*. Neither system is formally trialled; they are simply used as proof-of-concept experimental applications. *FarCry* is a file-sharing application, and is used to gain an overview of how data may spread within peer-to-peer communities whilst *Samara* is a tourist guide that relies on a recommendation system to guide users to interesting locations. A recommendation system is a particularly relevant topic within this area, as such systems require substantial amounts of data to be transferred between users and large amounts of data to be stored on their devices if they are to deliver a high quality of recommendation. Although in this chapter *Samara* is presented as a proof-of-concept system, in the subsequent chapter it is successfully demonstrated in operation within part of a larger sub system, *Domino*, and trialled in a mobile peer-to-peer game, *Castles*.

6.1 Epidemic distribution of data

For a mobile system to be classed as pure or hybrid peer-to-peer, and to enjoy the benefits of such architecture, it must be capable of distributing any required data within the community of peers without it passing through any external devices. For reliance-free content systems, this is a trivial matter, as messages are generally short-term and so communication is carried out directly between two devices that are in range of one another. However, for self-generating content systems the task is far more complicated. For example, imagine a pollution monitoring system that one carries with them throughout the day. As one travels, a sensor connected to one's PDA monitors and stores pollution

levels in various parts of the city, and a map and overlay is provided so that one can later browse the collected data.

Although such a system is of use, it would require a large effort for the user to map out a significant area of the city. Furthermore, pollution information would be unavailable if one were travelling to an area for the first time, and if one desired to avoid areas of high pollution the system would be of no help. However, if there were a community of users within the city, all using the system, they could share data—covering a far greater area of the city together than any one could achieve on their own. If each user had access to the entirety of the data collected by all users, then the number of areas for which one had data would be far greater. In addition, the larger number of users would also result in data being more reliable and up-to-date, as areas may be monitored more often by multiple users. Therefore, to enjoy these benefits, the data collected by each device must be spread throughout the entire community of users.

A standard technique in such situations is to rely on a central server in order to synchronise, and many existing mobile systems employ this method [154], [35], [43]. If a central server is used, there are two common methods in which devices connect to it. In one method, the user is required periodically to dock their device with a computer that is connected to the Internet. The dock then provides a pass-through network connection to the PDA that can then access the Internet itself and communicate with the server. It uploads the data collected since its last synchronisation, and downloads either the entirety of the available data, or only sections which relate to the local area or have altered since the last synchronisation. This technique has been successfully employed in systems such as those described in [46], [39] and [117].

A second method essentially works in a similar manner but eliminates the need to dock the device physically, by utilising the device's wireless network card to detect nearby networks and synchronise when a nearby network that provides Internet access is found. This technique is far less common, as it normally requires a greater effort on the designer's part. For mobile devices, a long connection period cannot be guaranteed as the user may suddenly move out of network range. Therefore synchronisation must be achieved through the atomic transmission of small segments of data rather than by sending the entirety of the collected data at once. The task of scanning for and successfully connecting to networks must also be carried out and is in itself a substantial task that requires direct control of the network card.

Finally, many developers may be hesitant to attempt utilising discovered networks due to the many legal issues involved, or to pay per-client costs to enable devices to access Internet connections provided by companies such as Vodafone and T-Mobile through infrastructure access points. Despite these negative aspects, there are a few commercial and research systems which do utilise this method to synchronise data. For example, the data collected by Skyhook Wireless' Bertha application is automatically synchronised over any T-Mobile access points that are detected. This is beneficial to

both the PDA carrier, who does not need to take the extra step of periodically docking their PDA, and to the organisation collecting the data, as the devices synchronise more frequently than if docking were required and therefore there is reduced risk of collected data being lost. This configuration of relying on paid connections is usually only employed when the data is of high value to a company, as it is normally the company that pays the associated costs.

Although these two techniques do provide a centralised solution that may be attractive for commercial reasons, the fact that they are centralised detriments the mobility and flexibility of the application. In all the systems listed, frequent synchronisation with a central server is required if functionality is to be maintained and data kept up-to-date. Indeed, the first time the applications are run, they provide little or no functionality until one manages to locate a connection to the server and synchronise one's PDA. Furthermore, even if peers are located within range of one another, the fact that one has synchronised will not benefit any other devices, each of which have to connect and synchronise separately.

Removal of the central server also minimises these negative aspects, but introduces the dilemma of how generated data is spread throughout the community. One alternative method that has recently been gaining popularity is epidemic algorithms. Epidemic algorithms seek to address the problem of disseminating information in peer-to-peer networks, and as such are highly suited for use in mobile communities. The core concept behind epidemic algorithms is that they act similarly to a contagious disease within a community; spreading from one node to another, and being stored within that node until another is encountered, when it can be copied to a novel node once more.

One of the advantages of epidemic techniques is that they are highly scalable and are not dependent on any one particular node being continually available. Indeed, Vogels et al. state that [168]:

> *These protocols allow systems to be built in pure peer-to-peer manner, removing the need for centralized servers...*

As mobile communities are never static, and have nodes that are highly transient, epidemic algorithms that are not dependent on any single node are particularly useful in mobile environments. Although epidemic algorithms were proposed for use on desktop systems as early as the 1980s, they are experiencing a recent resurgence due to this suitability to mobile systems. Despite the fact that Vogels et al. concentrate on the *Spinglass* project, looking primarily at epidemic algorithms for use in the desktop environment, they do go on to highlight more properties that help make epidemic algorithms effective in mobile environments [168]:

> *An epidemic-style protocol has a number of important properties: the protocol imposes constant loads on participants, is extremely simple to implement and rather inexpensive to run.*

The fact that epidemic algorithms are relatively simple and inexpensive to implement and run aids their use on mobile platforms, which typically have less processing and storage capabilities than their desktop counterparts.

Although epidemic algorithms have been widely known and discussed for some decades, one of the first serious attempts at applying them to mobile communities was in 2001 by Papadopouli and Schulzrinne. They describe 7DS (7 Degrees of Separation), an infrastructure they use to drive a mobile system for caching web pages on devices and spreading them throughout a larger community of users. This system allows mobile users who do not currently have access to the Internet, to continue to receive and view popular web pages whilst they travel. The authors define 7DS as [131]:

> *...an architecture and set of protocols enabling the exchange of data among peers that are not necessarily connected to the Internet.*

7DS demonstrates the strength epidemic techniques can have in a mobile environment. They can enable useful information to be delivered to members of a community who currently have no Internet access and, as previously highlighted, can do so in a lightweight and relatively simple manner.

Despite their increasing popularity, epidemic techniques have not been implemented and trialled in many actual mobile systems, and most remain theoretical. The primary reason for this may be that many epidemic-based systems need a large number of users to operate optimally, and it is often difficult to recruit the required high number in a research scenario. In such situations it is unlikely that enough trial participants will be available for the epidemic techniques employed to exhibit the full range of benefits they may potentially provide if a larger group were available. However, there are many reports on results from computer-generated models of how epidemic algorithms might perform. Papadouli and Schulzrinne report that their simulations demonstrate that epidemic algorithms can allow data to spread to close to 100% of users after only 25 minutes if there is a density of 25 users per square kilometre [131].

Lindemann and Waldhorst expand Papadouli and Schulzrinne's work, simulating how data can spread through the use of epidemic algorithms and considering for more parameters [108]. They find that changes to buffer size, among many other factors, can substantially affect the spread of data in epidemic-based networks.

In addition to providing the rapid dissemination of data in peer-to-peer communities, epidemic algorithms prove to be robust. EraMobile is an epidemic technique that continually monitors the node density of the overall community and number of neighbours around a node, and adapts buffer levels and transmission rates accordingly [130]. This allows EraMobile to reduce spread rates when the number of nearby nodes is high, reducing network traffic, yet increase spread rates when the node density is low, compensating for lower levels of encounters between peers.

Whilst there is a substantial amount of other literature focusing on epidemic models and their simulations ([120], [96], [107]), Demers et al. take a slightly more practical approach and describe how epidemic algorithms may be applied to use with databases [48]. The epidemic techniques Demers et al. detail, in particular *rumour mongering*, can be applied within a peer-to-peer community to spread data throughout the community. If we take the earlier pollution application example, we can envision how an epidemic algorithm would work in this scenario. From the user's perspective, the manner in which the application is used remains the same. The device is carried as normal and collects pollution data as it is moved around the city. A map display on the screen has an overlay showing all the pollution data available. However, when two users pass each other in the city, their devices now discover one another and exchange as much pollution data as they are able to. When one of these devices passes a third user, it transfers not only its own information about pollution but also the data from the peer it encountered previously. In this way, data introduced or generated by one peer is spread throughout the community at an exponential rate, as more peers gain copies of the data. In the pollution example, data is aggregated and so the number of instances of data about one particular location is not significant. However, if the application requires only the latest copy then updates can be time-stamped. Devices receiving information from a peer can then discard any entries that are older than copies they already hold. This ensures that the latest data continues to propagate throughout the community, whilst older data is spread only by peers who have not yet received the latest copy to others who have even older data.

Epidemic spreading techniques can also be utilised in a controlled manner and these are indeed already common in many systems. In *FolkMusic*, a user whose device was in range of another was able to browse the music library of a peer and download music he or she selected to his or her own device [173]. As this music was then shared by the two devices, a single song was able to spread throughout an entire peer-to-peer community in this manner. Although this technique was not used in a way to deliberately evoke epidemic spreading, the manner in which the song spreads throughout a community is typical of the three epidemic algorithms described in [48]. Unfortunately, although *FolkMusic* did rely on ad hoc networks and so appears applicable, it was not particularly mobile as it ran only on laptop computers. Despite mentioning that future work would be conducted to create a PocketPC version, no such version was ever implemented. However, Hakansson later implemented *Push!Music* which did run on mobile devices and provided almost the same functionality as *FolkMusic* [78]. *Push!Music* allowed users to share music in an ad hoc manner by allowing one user to "push" a song onto another's device when they were in range. Both *FolkMusic* and *Push!Music* rely on users themselves initiating the spread of a file to a peer device.

If information on device encounters is spread in an epidemic fashion, more sophisticated communication methods become available. For example, a message intended for a specific recipient can be efficiently directed through a peer-to-peer community in order to reach its target. Glane et al. describe such a procedure [71]. When a peer has a message it wishes to send to a specific target, it

reviews a history of past encounters. The history contains not only a log of the frequency it has encountered peers in the past but also ratings for how likely these peers are to encounter others. For example, imagine a device has a message for device *T* and by examining its history log it is calculated that there is 30% chance that this will be the next device the unit encounters. The next device that is encountered turns out to be *A*, which has a 20% chance of encountering *T* next. The decision made would then be to hold the message on the local device, as it has a slightly higher chance of encountering the target first. However, if *A* turned out to have an 80% chance of encountering a device *B*, which in turn had a 50% chance of encountering *T* then the decision would be made to pass the message over in the hope that the message would progress from *A* to *B* and finally to the target *T*. Clearly, this strategy is not suitable for messages that must be delivered in a prompt fashion but it does allow a message to navigate through a peer-to-peer environment without resorting to mass duplication and broadcast. It should be noted that although the navigation of the message through the community is not epidemic, the information about encounters which is maintained in routing tables on each device must be spread and updated according to epidemic algorithms if devices are to have enough information to route messages correctly.

An interesting technique for gathering data for later rapid distribution is found in [73]. In this antiviral system nodes are set up with the specific aims of detecting viruses, gathering or generating immunisation methods, and rapidly and strategically spreading the antiviral information to clients in order to stem the spread of the virus. These special nodes are called honey-pots and are capable of *leapfrogging* the virus in order to contact other honey-pots outside the infected area and prepare them to combat the virus. One of the reasons the honey-pots are so effective is that they are strategically located and actively seek out viruses. Applying similar techniques to aid in epidemic spreading rather than stem an epidemic viral spread provides an interesting strategy.

By moving clients of a peer-to-peer community to an area that is likely to be of high interest to the community, it is possible to gather and concentrate a substantial amount of content and to spread it rapidly to interested clients. For example, if a standard peer of a pure peer-to-peer system providing tourist information were placed in a museum it would automatically, as part of its normal function, contact and communicate with all the peers that visited the museum. Initial visitors would generate content that the peer permanently located in the museum would gather. In contrast, if peers randomly encountered each other, the information they exchange may be of value, but may not be immediately relevant to the area the users are currently in. However, here all the system's visitors to the museum are likely to be using the system to conduct their visit, and the statically located peer will gather a large amount of information specific to the museum itself. Subsequent visitors will then be guaranteed to have this substantial database of information made available to their device upon arriving in the museum. This strategy allows a location of specific interest to a community to be rapidly documented.

In short, removing the reliance on random encounters in a peer community can greatly aid the spread of information about a specific location. The most significant advantage of this strategy is that the client

is identical to all the others in the peer community, and so no specialised code is required and the setup time is minimal – simply placing one of the peer devices at the desired location is all that is necessary. To see how this method could be advantageous, it is useful to imagine a decentralised version of the *George Square* system. The *George Square* trials relied on a central server to collect the information from each set of users who visited a small square within the city, in order for these results to be used as content for the subsequent visitors. If the system were decentralised and information was spread in an epidemic fashion, then even if the trial participants were given the system over a long period of time it would be unlikely that any information would have been exchanged between peers – as they would have been unlikely to encounter one another. Therefore, when each user visited the square they would be starting with only the lowest amount of data, even if many users had visited previously. However, if a peer device were permanently located in the square for the duration of the trials then each subsequent user would have been guaranteed to receive the entirety of the information generated previously on the square.

This technique provides the benefits of a centralised system without actually requiring a specialised server device, or creating additional code or network configurations to support it. There is nothing unique about the device left at a specific location except that it is not currently being carried by a user; as a result the entire system is still entirely pure peer-to-peer. It is suggested that this technique, titled *Bothy*[10], can be of great value in preparing event locations or areas of interests within a community.

Epidemic algorithms appear to be an extremely suitable method of distributing data throughout a pure peer-to-peer community. However, as stated previously, they have not been implemented and trialled in many systems and instead are more often simulated. Therefore, epidemic algorithms and techniques detailed here are implemented and trialled in systems described subsequently in this thesis—both to aid the author's own experience with epidemic techniques and in order to ground the work in practical applications.

6.2 A mobile, peer-to-peer file-sharing application

File-sharing applications are one of the few mobile, peer-to-peer systems that are commonly found both in existing literature and commercially. For example, [173] implemented a file-sharing system that allowed users to browse and share each other's music collections. Files can also be shared, although not easily, using the functionality included as standard in Microsoft's PocketPC and Windows Mobile operating systems, which allows them to "beam" files to other users either through IR or Bluetooth hardware.

However, in all these systems the spread of data is limited in some way—most frequently, users themselves must initiate the transfer. In order to experiment with automated transfer, a system called *FarCry* was examined. It is important to note that *FarCry* was created solely by Malcolm Hall and that the author did not contribute to its design or implementation directly (although it does rely on previous

[10] After the notion of a Scottish bothy in which travellers utilise a shared hut, often noting details of their visit and travels in the bothy book located therein.

work from this thesis that the author did work on, including the enhanced wireless driver and SDS). However, the system is a perfect test bed for the issues discussed in this chapter and was thus seen as applicable for use here.

The *FarCry* sharing application for PDAs demonstrates the guidelines and infrastructure identified thus far in the thesis in a working and useful mobile application. It continually broadcasts its existence while simultaneously searching for peers, and when a peer is located a sound plays and a notification bubble appears to alert the user. Clicking the alert opens up the *FarCry* browser screen, which shows files on the peer device that the user has selected to share (Figure 36). Peers are likely to discover one another at approximately the same time, and it is possible for two peers to browse one another's files simultaneously. Shared files may be of any type, including music, video or photographs. The share screen also shows some basic information about the owner of the device – such as name and an avatar image.

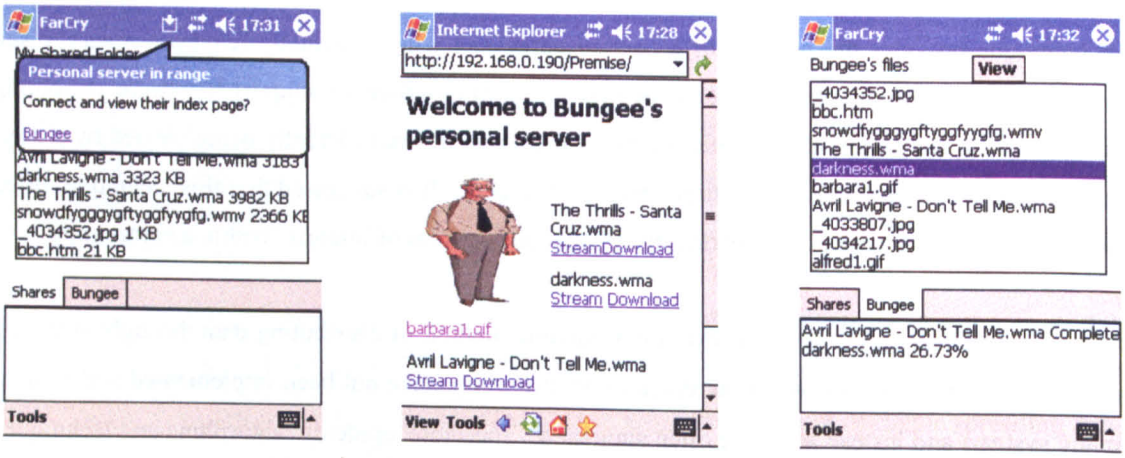

Figure 36: The *FarCry* user interface showing from left to right: the notification bubble when peers are in range, the introductory page when viewing another's library and the list of files on another device.

When one notices a file one is interested in, one can select to stream the file and play it in real-time (if it is of a suitable type such as audio or video) or copy the file to one's own PDA. Alternatively, rather than selecting files oneself, one can elect to enable 'leech mode'. Whilst in this mode one's PDA simply attempts to download as many shared files as possible from each peer it encounters. *FarCry* supports both epidemic spreading controlled by the user and, through leech mode, fully automated epidemic dispersion of files.

FarCry utilised the custom network driver and SDS to achieve its operation in its pure peer-to-peer environment. The application can be categorised as a self-generating content one as, from the application's perspective, new content is either received from peers or added directly into the application by the owner of the device.

FarCry was never formally trialled but informal testing proved worthwhile as it revealed the severity of letting epidemic spreading occur in all situations, and led to a simple categorisation of epidemic

spreading techniques: *blanket, user-controlled* and *system-controlled.* In systems in which one must pick the files one desires before they are transferred, the files spread in an epidemic but controlled manner, and this spread may be called *user-controlled.* For example, if user *A* downloads a music file from the original owner *O,* then there are now two peers in the community from which the file can subsequently be downloaded. Thus, user *B* may download the file from either *A* or *O.* This is clearly epidemic spreading, as files can be downloaded from peers other than the original creator or owner of the file and, as the file becomes more widely available, the rate of its spread through the community can grow exponentially. However, the spread of the file is not completely random or global. It spreads in a *controlled* manner, as a user must actively browse for and select the file before it is transferred. This type of spreading, and use of the system, was the default behaviour of the *FarCry* application.

When leech mode is enabled in *FarCry*, a *blanket* spread is observed, as each device attempts to gather a copy of every file shared by every peer device it encounters. This provides an exponential and extremely rapid epidemic spread and, if no new files are introduced, will result in the community reaching equilibrium, in which every peer has a copy of every file. However, in practice it was found that storage space on standard mobile devices is low (typically around 128MB built in with a slot for a 1 or 2GB memory card) and each device will usually run out of storage space after a relatively short time in such an environment. This is extremely problematic as, once storage space is full, the system completely stagnates as no new information can enter the system. In *FarCry* this results in a user ending up with a static set of arbitrary files, which he or she may or may not be interested in. Certainly, once the user reviews the files, *FarCry* is of no more use to that particular individual, unless additional storage space is made available or existing files are deleted—although it can still serve other users by continuing to distribute files. Obviously, this problem was experienced fairly quickly in our trials, as the devices used did not have large memory cards. However, regardless of how much memory is available on the device, this problem will eventually occur.

While this would suggest that a controlled spread method is generally more suitable for epidemic applications there are many situations in which it does not provide a rapid enough dispersal of the data. For example, in [71] the performance of the routing algorithms is dependent on the clients having the most up-to-date information possible in order for them to identify correctly the probability of one peer device encountering another. Indeed, Goldenberg et al. make it clear that a blanket epidemic spread is the most fail-safe manner in which to send a critical message in a peer-to-peer community [73].

Furthermore, even if a controlled epidemic spread technique is used, it is inevitable that the device will eventually run out of storage space unless old data is cleared. This is perhaps slightly less critical if the spread is *user-controlled,* as if one is selecting files or information to download, one is more likely to be aware of the current storage state, and to delete old or unwanted data if the store is full. If there is no input from the user during exchanges, and instead the system itself selects which data to exchange with peer devices, the epidemic spreading technique can be called *system-controlled.* When *system-controlled* epidemic spreading techniques are employed, as in [73], then just as with blanket spreading,

a method of clearing old data becomes essential. It should be noted that the *controlled* in *system-controlled* refers to the fact that the system provides some intelligent control or filter to the spread of data, rather than that the epidemic spread is simply automated by the system. Therefore, blanket-spreading is not a subset of system-controlled spreading, as with blanket-spreading there is no filtering applied, and the spread of data is simply as opportunistic as possible.

User-controlled spreading is obviously most useful when fine control over what is transferred is required. However, as it requires continual user input to operate, a user-controlled spreading method often results in many wasted peer encounters. For example, if two users are passing one another in the street their devices carried in their pockets may activate an alert but this can be easily missed in a busy environment or simply disregarded if a user is otherwise occupied. As the spread of data from peers is often fundamental in providing an up-to-date and relevant peer-to-peer application, these missed exchanges can take the data exchange rate below a critical level, so that the system eventually becomes stagnant and useless. In such situations, a system-controlled epidemic algorithm can prove far more useful, as its automated behaviour ensures that data exchanges with peers are carried out instantaneously whenever possible, and thus aid in maintaining the flow of data through a peer community.

In order to further discussion on epidemic techniques, it is useful to define explicitly the three types of epidemic spreading techniques for mobile, peer-to-peer environments:

- ***blanket***: whenever a peer device is encountered the entirety of available data held is attempted to be transferred
- ***user-controlled***: the user elects if and when to transfer data to or from a peer device
- ***system-controlled***: the spread of data is automated, but the system applies some filter to the available data so that a subset is transferred in each exchange rather than the entire dataset

It is clear that each category exhibits unique advantages in the mobile environment and that the choice of which to use is often specific to a particular application. As both blanket and user-controlled epidemic spreading are easily understood and relatively simple to implement, the next section concentrates on a system-controlled spreading technique and subsequent methods for culling data in order to prevent storage being rapidly consumed.

6.3 A novel method for distributing data in mobile, peer-to-peer environments

The brief investigations conducted with *FarCry* led to a realisation that an intelligent method for both a system-controlled epidemic spreading technique and a method for identifying and culling irrelevant or old data was required, in order to stop devices simply filling up their storage with data whilst maintaining a healthy flow of data through a peer community. Culling old data is particularly important in a mobile environment, as storage is normally substantially lower than in desktop machines. 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. It is therefore critical to ensure that only relevant data is stored.

It should be noted that whilst user-controlled and blanket epidemic spreading are generally quite limited in implementation options (user-controlled simply relying on the user's acceptance and blanket simply being as opportunistic as possible) system-controlled spreading methods can be implemented in many ways. Therefore, the following method is simply a single, hopefully elegant, solution presented in the hope that it can be reused in its current form and to illustrate the basic concepts involved in system-controlled epidemic spreading.

As stated, in a system-controlled epidemic spreading algorithm the system must make intelligent decisions about when to copy data from another device and which data, out of all that may be available, to copy, and in which order to copy it. Peer encounters may be short, occurring as users pass one another whilst walking in the street, or long, as when two colleagues occupy the same room for the duration of a meeting. Peer devices may meet in many circumstances and thus the length of peer encounters cannot be pre-determined. It is therefore important that devices are intelligent enough to transfer data that is likely to be of value in the time available during encounters.

Even if the length of an encounter were not an issue, any system-controlled epidemic spreading method cannot simply copy all the available data (this would be blanket epidemic algorithm), as this would quickly consume all the storage on the device and may not be possible if the encounter is not a particularly lengthy one. One obvious solution would be to choose data at random or take a small section of only the latest available data. However, with these techniques there is no guarantee that any relevant data on a peer would ever be retrieved. To obtain data that is relevant to a peer the system must attempt to identify segments of data that are likely to be of interest to the user and give priority to these when peer exchanges are conducted.

In order to identify sections of data or files that may be of interest when peers meet, a collaborative filtering algorithm was utilised to generate recommendations for files or data a peer is likely to find relevant. The *Recer* recommendation algorithm was again utilised (it had already been used in *George Square*) due to the prior familiarity Glasgow Equator had with it, and because it is an extremely generic recommendation system which is able to operate smoothly with heterogeneous types of data. As the system-controlled epidemic system being implemented was to act as an example and to, hopefully, be reusable in future peer-to-peer applications, a generic solution was favoured.

6.3.1 Recer

The Recer algorithm is simple yet powerful [28]. Actions which the user or application performs are monitored and logged. Typical actions may be accessing a file, browsing to a web page, sending a friend an instant message, or simply switching from one application or window to another. Each action is logged as a plain text entry in a database, along with a timestamp noting when it occurred and

additional metadata such as the type of entry it is and which user performed it. Over time, the entries create a continual log of the user utilising the application or their device, which may be thought of as one history among the various possible usage scenarios. Although it is possible to create recommendations based on a single history alone, it is not normally wise to rely only on a user's history of use, as this will result in no recommendations that are novel to them being delivered. Thus, the history of a single user must be compared to the histories of others if novel and interesting recommendations are to be made.

The actual recommendation algorithm works by first extracting all of the most recent entries from the user's own history. This recent activity forms a user's *context* – the items that are relevant to the current activity they are participating in (Figure 37).

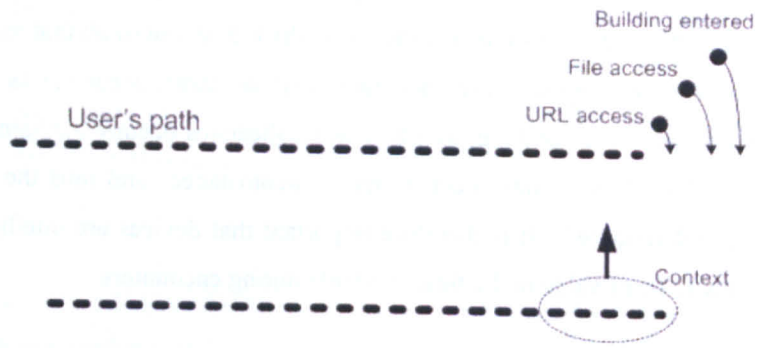

Figure 37: Overview of the Recer algorithm used to identify context. Items are recorded to the user's history (or path log) as they occur. The user's context can then be said to be the recent activity from this history and can be extracted.

Instances of each individual item from the context are searched for in the histories of other users. When a matching item is located, a *window* around it is generated by creating markers in time on either side of the item, and any collocated items found within this window are then extracted. Finally, the results from all the windows are amalgamated and ranked in order of number of occurrences (Figure 38). The highest ranked items are then presented as recommendations..

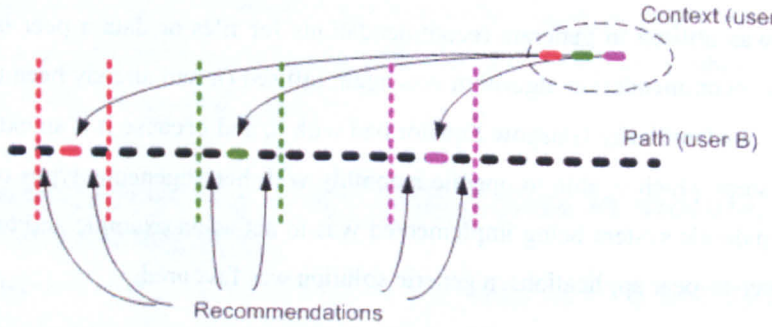

Figure 38: Overview of the *Recer* algorithm for generating recommendations. Items from one user's (user *A*) context are identified in another's (user *B*) history log. Items adjacent to these are then extracted and ranked as recommendations for user *A*.

Versions of the Recer algorithm were implemented by the author as APIs in Java, 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 PocketPC or Windows Mobile and desktop machines. When running on desktop machines, Recer can use Microsoft SQL Server, Postgres or mySQL as its database back end. When on a mobile device only a Microsoft SQL Server CE database can be used.

The length of the context, size of the windows and number of recommendations returned may all be set using the API in order to tweak 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 all the systems Recer has been used in for this thesis.

Obviously, Recer, like all collaborative filtering algorithms, relies on information from other users to provide high quality recommendations. In Recer's case, it is the history information from other users that is required, and a method of spreading this information in an epidemic manner within a peer-to-peer community was implemented.

The history information Recer generates 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. However, the compact size of the individual entries does not entail that a blanket epidemic spreading method should be used—the data must still be spread in a controlled manner. The length of a peer encounter may not be predetermined, and many encounters last only a short period of time and are cut off suddenly as the peers move out of wireless range of one another. It is therefore critical that the information transferred in the short period available is likely to be of value to the recipient.

When a peer conducts its first exchange, the histories transferred are indeed somewhat random as there is no previous experience about which data may be of most relevance. As the connection is far from reliable, the history information is transferred in small segments of 100 entries at a time. This aids in preventing loss of large amounts of data, as no more than 100 history entries (a relatively small number) will be lost if a connection is broken. To assist further 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 65 of a 100 entry segment is being transmitted then the secondary thread will be able to recover the 64 completely received entries. This robust protocol ensures a high successful transfer rate even in the extremely transient and unreliable peer-to-peer network environment.

Peers do not only share copies of their own paths but also those of others from which they have downloaded information. 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*. In this way, a single encounter with one peer can provide a wide variety of data from many users.

After history data has been received from an encounter with a peer, it can be used when generating recommendations. It is these recommendations that 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 and epidemically spread the Recer histories. However, this underlying metadata is shared only to allow 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 for items, these items can subsequently be actively searched for within the peer community if they are not immediately available. For example, if a device *A* receives history information from peer *B* that 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*. If it 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 whether he or she would be interested in accepting the recommendation and downloading the file to their device. Thus, this technique of recommending shared peer data not only filters the data to select that most likely to be of relevance but also supports both system-controlled and user-controlled epidemic spreading of data.

As discussed, the first transfer of the recommendation metadata is random as there is no previous data from which to calculate areas of interest. However, after a number of histories have been downloaded, the system periodically analyses them at times when the processor is otherwise unoccupied. It attempts to determine from which histories recommendations of the highest interest may be drawn 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. It is important that the final rating gives the average similarity on a per-item basis rather than over entire histories, as a history average would not give a meaningful ranking unless compared to a history of similar length. It is only segments of history that are downloaded during peer encounters—it would be rare that a whole history would be transferred and, if it were, 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.

Once history rankings have been calculated, all subsequent history exchanges during peer encounters rely on them to give the highest ranked histories priority. When a peer is subsequently encountered, it is asked 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. This process prioritises histories from which recommendations more relevant to the user are likely to come. Gathering data from these histories first helps ensure that the short encounters in a pure peer-to-peer environment are less likely to be wasted, by transferring data that is unlikely to prove beneficial to the user.

The list of similarity rankings is reused to provide a second beneficial feature—that of culling old or irrelevant data. In order to stop the storage space on a PDA being entirely consumed by the history information collected from peer devices, it is necessary that there be a process that periodically cleans it up. 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, from which recommendations are least likely to be found for the user, are discarded. If, after discarding 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. The continual cycle of discarding old and irrelevant path data whilst renewing it with paths downloaded from peers results in a recommendation system that is kept up-to-date and continues to provide novel recommendations.

The combination of the Recer collaborative filtering algorithm and the additions to allow its path data to be distributed and maintained in a peer-to-peer environment is named *Samara* (short for System for Ad hoc MetAdata RecommendAtions). A mapping system that was the first application to utilise *Samara* is detailed in the next section.

6.3.2 *Samara*

The first pure peer-to-peer mobile application to utilise the *Samara* infrastructure is a mapping and location recommending system designed to be used by tourists. As a tourist travels around, he or she carries a PDA, equipped with GPS, on which the application runs and records locations visited.

As data from the GPS unit is delivered in standard NMEA format, the main component of which is latitude and longitude delivered approximately once per second, locations of interest must be interpreted from this raw NMEA stream. One possible solution would be to rely on some of the many Internet services that maintain a database of latitude and longitude mapped to postcodes to convert the raw coordinates. Once the postcode was obtained from a web site that provided this conversion, another web service, such as that provided by the Yellow Pages website, could be used to supply a list of businesses at the current postcode. This method allows for intelligent guesses to be made about which building a user had entered.

However, in a pure peer-to-peer environment, in which this particular application exists, reliance on an external third-party is best avoided. 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 a country area with no particular nearby building. Such areas may be of great importance for someone who uses such a location to picnic, walk their pet or simply admire the view. Therefore, instead of relying on the discussed method, a self-generating content structure was implemented. This worked by recording coordinates in a database, and marking a surrounding region whenever the system detected the user was at a location that seemed to be important to them. Rather than relying on an external database, it is the user's own actions that are relied upon to identify the locations that are of value to them—allowing any location whatsoever to be recorded. Important locations are interpreted using the two main techniques outlined in [88] that allow for both outdoor and indoor locations (at building granularity) to be detected.

The first technique, which allows indoor locations to be detected, involves recording the time and locations where a GPS signal is lost. If the signal is not regained within a certain timeframe (five minutes in our system) then it is possible that the cause of the loss of signal is that the user entered a building. As in [114], 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 false positives such as loss of signal due to the user entering an area with poor line of sight to the satellites (such as a narrow city street), accidental disconnection of the GPS unit from the PDA or other random GPS 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 method, which allows outdoor locations of interest to be detected, simply involves monitoring the period of time a user stays within a specified range. If approximately the same location is maintained for a period of time (again, five minutes in this particular system) then the location is noted. Just as in the first method, three repeated 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 park bench that a user frequently rests on while eating lunch.

When a location is detected as important and entered as such in the database on the users PDA, the location is assigned a random, unique identifier string. The user is free to leave this identifier as the randomly generated string. However, if one desires, one may at any time review one's locations and rename them—replacing the random string with 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 (to protect privacy).

Once locations are recorded in the database, the user is marked as having visited them whenever they are again 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. In this manner the application identifies any locations important to the user and creates a record of the user's visits to them.

Whenever a user's device encounters a peer, the Recer algorithm and the *Samara* techniques previously discussed are employed to exchange the most relevant items for generating future recommendations. Thus, new locations and recommendations to visit them spread throughout the peer-to-peer community in a system-controlled epidemic manner. Recommendations for locations from a peer community can allow a user to learn about new locations, such as restaurants which have just opened, in their own area as well as providing a vital resource and guide to a tourist in a new city.

Samara essentially provides a novel peer-to-peer epidemic routing algorithm. It uses recommendations from a standard collaborative filtering system to drive the spread of data within the peer-to-peer community.

6.4 Conclusions

The work in this chapter examines methods of distributing and handling data in mobile, peer-to-peer environments. By looking at existing epidemic spreading techniques and examining them within *FarCry,* the fact that both the spread and storage of data within mobile peer-to-peer communities must be continually monitored and controlled is highlighted. This leads to the design and implementation of a novel *system-controlled* epidemic spreading technique in *Samara,* in which a recommendation system is used to drive the spread of data between devices. This, in turn, aids in ensuring that useful data be transferred first during peer encounters, which are often limited in time and thus do not normally provide opportunity for all available data to be transmitted.

In addition, the same recommendation algorithm is used to monitor data stored on individual devices, and to decide when and which data to remove when available storage is low. Thus, *Samara* provides a generally applicable solution for both spreading data within a peer-to-peer community, and monitoring and maintaining the data on individual devices.

The work presented in this chapter is crucial to the work in the following chapter, which relies on the techniques discovered from experimentation with *FarCry* and implementation in *Samara* to facilitate the creation of a mobile adaptive infrastructure.

7 ADAPTATION IN MOBILE SOFTWARE

One of the most important findings of the literature review that has not yet been discussed further, is that mobile, peer-to-peer systems should be highly adaptable—capable of adaptation at a low level, adapting the system itself when required. This concept is examined in this chapter and a system, called Domino, for providing an extremely high degree of adaptation in a peer-to-peer system is implemented and demonstrated.

Throughout the design and use of all the systems discussed, it has been clear that many forms of adaptability are required to allow mobile applications to shape themselves into the differing environments and contexts they are used in. Both *George Square* and *Navizon* experienced significant changes to their design and execution environment after initial testing, and the fact that they exhibited high levels of modular design greatly aided in rapidly reconfiguring them to operate in differing mobile environments. In *George Square,* the *EQUIP* engine allowed for a modular structure in which any single component could be activated, deactivated, moved from one system to another, upgraded or replaced without having to halt the entire system. Indeed, the system as a whole was capable of altering its behaviour and functionality as new modules came online providing extra information. For example, when a new user joined midway through a trial, the system would automatically detect their presence, hook into the EQUIP space running on his or her system, and append information about the new user's location, photographs and recommendations to those already using the system.

Similarly, the modular design of Navizon allows for individual components to be turned on or off, or replaced without halting the entire system. In its normal use, Navizon is continually hotswapping modules, selecting whichever is the most suitable at any given time. Modules which become available only after the system is already running (for example, when a user plugs a GPS unit into a previously empty slot or turns on a wireless card which was off) are automatically queried and the information they provide is smoothly integrated.

Realising the benefits modular design provides, which are particularly relevant in mobile systems, an attempt to implement a mobile modular architecture was undertaken. The main motivation behind this work was the desire to support fully flexible applications that can provide substantially different functionality depending on the user's context, in accordance with the similar discoveries from the literature review. For example, a flexible tourist support application may provide a general map of the city but when a user enters a particular complex such as a museum, a module providing a guide to the museum may be automatically installed and loaded on his or her device. Alternatively, if the visitor to a city desires food he or she may select to install and run a restaurant module that could overlay the locations of restaurants on the existing map and provide menus and ratings when selected.

It should be noted that the desired form of adaptation is one that puts the user, and his or her usage history and knowledge level, at the core. This is in contrast with walk-up pop-up systems in which an application adapts the information it displays based solely on external factors [19]. In such systems, the content delivered is identical regardless of who is using the system. Both [161] and [102] present such systems—in both the same information is displayed whenever a user moves to a specific location without regard to, or consideration of, any differences between the users. The findings of the literature review make it clear that adaptation should be achieved with a high regard for the particular user of the system and his or her context, rather than in this walk-up, pop-up fashion.

Bunt et al. distinguish between two types of adaptation: adaptive and adaptable [24]. Adaptable systems are ones in which the interface is highly customisable but in which the user must perform any customisations manually. Adaptive systems are ones in which customisations happen automatically and in which the interface tailors itself to the user. Bunt et al. propose that a combination of both types may be the most suitable—implementing customisations automatically but still allowing the user to undo automatic customisations or make his or her own. As will be seen, the *Domino* system does adhere to this advice and is both adaptive and adaptable. Bunt et al. also discover that although users often believe their own customisations are advantageous they frequently perform poorly and, in general, customisations that are automated and applied by the system itself often result in increased performance. A final finding of Bunt et al. is that users can become distrustful of adaptation if it is not made obvious to the user when it occurs, and if the effects are not fully disclosed. Again, as will be seen in the *Castles* game used to trial *Domino*, this is heeded and *Castles* does attempt to make adaptations apparent when they occur. However, this issue, in combination with *Seamful Design*, deserves further investigation than is presented in this thesis and so is mainly left as an item for future work (discussed in Chapter 8).

Adaptability can undoubtedly aid in providing a more useable interface, increased enjoyment and, perhaps most importantly, benefits to efficiency both in the interface and the underlying system itself. For example, Henricksen and Indulska demonstrate how a web browser and proxy can be used to adapt the quality, quantity and content of information being retrieved from the Web in order to suit the specific network bandwidth, processing capabilities and display characteristics of specific devices [87].

Findlater and McGrenere studied relatively shallow system changes – concentrating on interface changes and in particular on menu items [63]. They provide useful comparisons between three types of system: a static, unalterable one in which menu items were set, a user-alterable one which allowed menu items to be reordered as the user saw fit, and an automatically adaptive one which analysed the items the user accessed most frequently, and attempted to place them at the top, or most easily accessed, areas of the menu structure. The findings showed that the substantial majority of users preferred the adaptable menus but that they preferred the system under which they had full control, feeling that changes under a completely automated system were difficult to anticipate. In retrospect, this seems self-evident as finding a target has moved after becoming used to finding it at a particular

location is both frustrating and time-consuming to adjust to. Without forewarning or asking a user to authenticate a change, there is also no possible way for the user to foresee the exact moment the change will occur and so there is no alternative method for dealing with the problem except correcting from it after it has transpired. Despite this, the findings indicated that the users who did favour the fully automated system were extremely strong in their support for it. This led Findlater and McGrenere to suggest that the most suitable approach may be a 'mixed-initiative', combining adaptable and self-adaptive elements to create a system which is partly automated but requires user confirmation and validation before changes are implemented. Clearly, Findlater and McGrenere's findings are almost identical to those of Bunt et al..

Adaptive systems may not only aid in overall user experience and increase efficiency but may become required to manage the complex configurations that are now emerging in large software applications. Birsan reports on a 1000 plug-in system stating, that such systems are rapidly becoming common, and by their very nature, may be unmanageable for average users [15]. Thus, a form of automation which is able to filter the possible configurations before presenting a reduced list of adaptations to the user may be the only method to manage such complexity. Indeed, Findlater and McGrenere state "adaptable and adaptive interaction techniques are likely the only scalable approaches to personalisation". Crow and Smith come to similar conclusions suggesting that "collaborative dialogues with the user" might help to strengthen adaptive systems [44].

There are many existing mobile systems that do provide a rudimentary level of adaptation. For example, *MovieLens Unplugged* attempts to learn what movie genres and actors a user enjoys and deliver appropriate recommendations when the user is actually in the store [117]. 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 from which the system can draw to generate custom or adapted behaviour.

The requirement of a lengthy set-up process can negatively influence potential users. This negativity is increased for mobile devices, as many perceive these 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, many users 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 a mobile application that does require any time-consuming set-up period.

Another system, *HIPPIE*, attempts to customise information delivered to museum visitors. 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 [127]. Schiele et al. describe another museum system in 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 [148]. 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 recorded previously been made.

Although the systems discussed do adapt to the user's context and behaviour, the only elements that do so are the information that is delivered, and not the system or the functionality it provides. Currently, there are few, if any, known mobile applications, and few desktop applications, that adapt themselves, increasing or simplifying their own functionality to suit a user's needs. For example, although the desktop MovieLens application does learn what movies a user likes, it does not adapt its functionality based on this information [143]. It cannot learn to provide further types of information specific to that genre or start recommending anything other than movies. In short, mobile applications overwhelmingly remain static in their code and functionality; there are few mobile applications or systems able to have their code and functionality augmented or amended at runtime.

Adaptation of software and functionality has been proposed for numerous reasons but rarely implemented in desktop systems. The majority of existing work is theoretical, proposing architectures but not actually implementing functioning examples. Dashofy et al. describe a self-healing system architecture that potentially allows systems to recover from failure through the dynamic swapping of software modules at runtime, controlled by an intelligent software agent that can select an appropriate repair plan through analysis of system modules [45]. Whilst the architecture they propose is extremely strict, requiring substantial amounts of metadata on each software module and configuration, they make clear the requirement of at least a basic standardised interface for modules used in adaptive systems. Georgiadis et al. propose a similar architecture of modules, components and interfaces but apply it at the system design stage in order to provide rapid development and flexible software implementations [70]. Although they concentrate primarily on using components to construct the initial structure of a system they also show that such components can later support rapid reconfiguration at runtime. Dellarocas et al. also rely on software modules and interfaces to create a self-evolving architecture that uses dynamic reconfiguration to swap in new software modules in cases where it is detected that they may outperform existing modules [47].

It is apparent that the majority of adaptation architectures that support dynamic reconfiguration (at runtime) rely on modular approaches such as these. Predefinitions of modules and the architectures they are to be used in are required in any situations where a new software module is to be smoothly integrated into an existing configuration of modules. If care is not taken, architectures can become overly specified and lead to strict limitations being imposed on developers. As will be seen, the *Domino* system described later attempts to avoid this by enforcing only a few basic rules that modules must adhere to.

Recombinant systems, which rely on a set of common interaction patterns to enable rich communication to occur between devices that have little or no previous information about one another, are beginning to emerge. The *Speakeasy* system is one example which relies on 'interaction patterns' to support data exchange, user control and contextual awareness between devices which have no a priori knowledge of one another [122]. Through the use of interaction patterns, *Speakeasy* systems can adapt to discover and use external resources at runtime, even if the resources are of types not previously encountered. *Speakeasy* achieves its adaptation through a set of several fixed elements: a small set of fixed domain-independent interfaces which modules can use to initiate communication, mobile code that supports dynamic extension of functionality, and "user-in-the-loop" interaction that bestows ultimate control on the end user in deciding if and when communication between entities will occur. *Speakeasy* supports only a small number of components associated with devices and services in the local context, and filters mainly on the basis of known locations and owners. However, the set of information which it relies on to achieve its filtering is static and the system is incapable of updating to suit even gradual change. Furthermore, *Speakeasy* again relies on external factors to make its filtering decisions – completely ignoring a user's current context in a manner akin to walk-up pop-up models.

Adaptation on the desktop is, unsurprisingly, slightly more evolved. One of the earliest studies in software adaptation and user collaboration centred on the Buttons system [110]. Buttons allowed software functionality to be shared via email, permitting users to both collaborate over which functionality they used and to match the system to their personal work practices. Users were able to customise 'buttons' by setting parameters in pop-up menus while more expert users were capable of more sizeable changes at a code level. In Buttons a 'tailoring culture' evolved, in which expert users would make substantial changes, hoping to increase usability or efficiency and then share their adapted components via email to a few users. Users who found the changes advantageous would often mail them out to others, thus allowing positive functionality to spread gradually throughout the community. Thus, Buttons aided users who were not experienced in a particular area or who were occupied on other tasks to acquire highly customised amendments from experts who had both the knowledge and time to implement beneficial changes. This model, in which the majority of people within a community rely on expert users to customise their own work environments rapidly, is an echo of social interaction without technology where it is common to simply ask a knowledgeable expert to give recommendations on tools to utilise in order to perform a particular task.

Many users may not have the time to seek out such recommendations or may not even be aware improvements in their current applications are possible. For example, a user who relies on FireFox to browse to websites and write about ones they find interesting in their own blog may be unaware that a plugin that would simplify this task exists. By providing a button that, when clicked, automatically pastes the current URL and page heading into an input form for their own blog, the plugin can simplify and expedite the user's task. Similarly, a Visual Studio developer who manually deploys supplementary images that their application uses to a mobile device may not know that there is now a plugin available that automates this process. Users may continue to neglect plugins or changes that

could ease the tasks they frequently carry out even if they are commonly known within the community. Traditionally, knowledge of such improvements comes through discussion with co-workers, friends, or over the Internet through group forums or mailing lists. However, many users simply do not have time to seek out these recommendations from others—particularly now that the number of technologies used by the average person has substantially increased. There are simply too many technologies to keep up-to-date with, and finding a professional user who is abreast of current developments can be difficult even through the Internet. Conversely, fewer recommendations are voluntarily offered as the number of applications and combinations of software that users now run makes it difficult to gain the in-depth knowledge of their software environment, required to recognise when a particular module or application would be a worthwhile addition to a user's configuration. These recommendation and awareness problems are not isolated to desktop examples and also prevail in the mobile environment. For example, in the tourist application described previously, one may simply not know that a module which could deliver increased functionality in an area one is visiting is available and, even if one does, may not have the knowledge required to install and run it.

This would suggest that any adaptive system must be capable of automatically identifying adaptations that would benefit a user and subsequently, and automatically, acquiring, installing and running them. Without the ability to recommend improvements, an adaptive system is likely to languish as users find they do not have time to search out and learn to use new functionality. In short, a system that can fully adapt *itself* to a user's usage habits, interests and surrounding context is desired. Such a system, titled *Domino*, is described in the next section.

7.1 System adaptation based on users' context

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. The functionality that is spread is activated and deactivated on individual devices based on the user's current and recent context.

The *Domino* infrastructure consists of three individual sub-systems that were originally conceived and implemented independently. The first part is the communication infrastructure used for discovering peers, which was described in Chapter 4. The second part is responsible for creating recommendations and is the *Samara* infrastructure, which was described in Chapter 6. The final part of *Domino* is responsible for dynamic software adaptation at runtime. As this final piece of the entire *Domino* system was also originally titled *Domino* it will from this point on be referred to by the acronym DAF (*Domino Adaptation Framework*) in order to distinguish it from the overall system created by the merger of the three components.

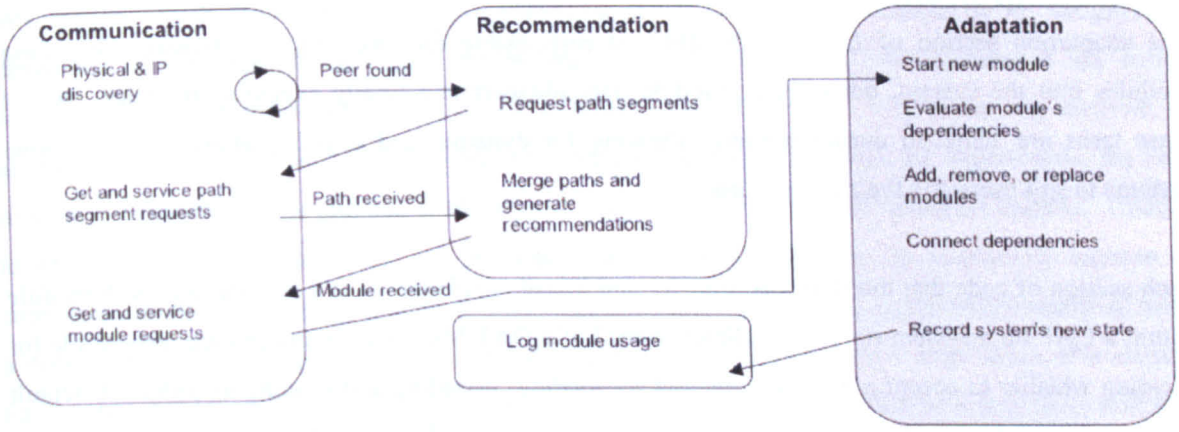

Figure 39: Overview of the Domino system.

Figure 39 shows how the three sub-systems involved link together to form the *Domino* system. The communication section is responsible for ensuring that peers are consistently connected to the networks that will maximise their opportunities for discovering other peers on the network, and for initiating the initial communication handshakes to allow transfers to occur. The communication section utilises the custom wireless driver originally constructed for and used in *Feeding Yoshi*. As before, this driver maximises opportunities for encountering peers as it ensures two devices within wireless range of one another will consistently connect to the same network—switching between infrastructure and ad hoc as necessary. Again, once on the same network, SDS is used to allow peers to broadcast their presence, and detect one another and the services they provide.

The recommendation section of *Domino* is the *Samara* infrastructure and is identical to the version previously used in the *Samara* mapping application. As this infrastructure has already been described it will not be discussed here. However, the method in which it interfaces with the other components in *Domino* will be discussed after a description of the adaptation section.

The three individual parts combined to create *Domino* were created independently, before *Domino* was conceived. The communications section is a combination of the enhanced wireless driver, SDS and communications required by Recer. Therefore, it is all my own work. The recommendation section is *Samara* that, as was detailed in the previous chapter, is based around Chalmers' Recer algorithm but has been completely re-implemented and heavily augmented by myself to operate in the mobile, peer-to-peer environment. The final adaptation section is the *DAF* which was designed and implemented entirely by Malcolm Hall.

The idea of combining these three components in order to provide adaptation in mobile, peer-to-peer environments was conceived by Malcolm Hall and myself. The considerable amount of additional implementation required in order to combine these three components into the final *Domino* system was achieved through equal efforts by Malcolm Hall and myself.

7.1.1 Adaptation

The adaptation section of *Domino*, the *DAF*, is responsible for receiving 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 suit users and their current tasks.

Each section of code that the DAF can transfer and install is referred to as a module, and each module within a DAF on a system runs under direct control of a DAF Manager. Managers are responsible for deciding whether to accept a new module, and for loading, installing and running modules. A default Manager is provided in the *Domino* system, which is fully capable of performing all of these tasks. However, if a developer desires, he or she is free to extend this default Manager in order to create specialised behaviour. For example, after receiving module recommendations from the *Samara* component in *Domino*, the default Manager requests a final decision from the user to determine whether the module should be installed and run. If such behaviour is not desired, the Manager class could be edited to bypass this final authorisation request. In general, DAF 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 easily to remove adaptations and additions that turn out to be less advantageous than expected. Indeed, in all the *Domino* systems we have created at Glasgow, the Manager used has been the default one and has been added to the application's main GUI form itself by simply having the form extend the default Manager class.

Domino modules themselves can contain any code and thus be of any size; from a small plugin that simply displays the time to an entire application such as a web browser. The only constraint on DAF modules is that they must conform to the DAF code interface in order to allow a DAF manager to control integration with other modules already active on a system. Module developers can easily achieve the required conformity either by implementing the DAF interface directly or by extending an included base class which already conforms to the interface. The interface itself, exposed through a .NET DLL, is extremely simple and contains only seven method stubs: SetManager, GetDependencies, FindDependency, AddDependent, Start, Pause and Destroy.

The first four methods enable new modules to be integrated smoothly into a running Domino system, which may have other modules already active and running on the system. When a new module arrives on a *Domino* system, the first method called is SetManager, which couples the DAF Manager to the module. This allows the Manager to analyse the module subsequently and control its initialisation. During initialisation the Manager checks if the module has any dependencies by calling its GetDependencies method. A dependency is a requirement one module has on another Domino module, which must be available within the system, either already running or known and executable by the Manager. For example, a map layer module may have a dependency on a map viewer, or a module that shows a user's location on a map may have a dependency on a module that provides NMEA

information. Such dependencies must be identified and coded into the module by its designer, or designers. 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 FindDependency methods of all currently active modules. If any of the modules 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 FindDependency are necessary, as in certain cases one module may be technically capable of supporting another but, 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, if it already has 5 connected layers then a call to FindDependency on behalf of yet another new map layer will fail. 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 FindDependency fail and the Manager does not have access to a required module DLL on the local device, the Manager will note the module that is required 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 correctly ensure a module is installed on 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.FindDependency(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 latter three methods—Start, Pause and Destroy—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 release any system resources it holds.

When a module is to be removed from the system, the *Domino* Manager again uses GetDependencies to identify any other modules relying on the module to be removed. These modules are 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 to 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 8.

The seven methods are all that are required to exist for any C# class to conform to the *Domino* interface. In many cases, not all the methods have to contain code and may remain as stubs. For example, some simple modules may not have any dependencies whatsoever, and their GetDependencies method will simply always return a null value. Similarly, some modules may not contain independent threads, and may be controlled by other modules on the system. These modules will simply always return true on a call to Pause, as they are not independently active. Samples of code from some *Domino* modules are included in the Appendix. Note that as some of these classes are from earlier iterations of the *Domino* system, their method names may not fully match the current ones described. For example, the GetDependencies method was titled GetChildren in early experimental iterations of *Domino*.

As Managers can verify if a module's main thread is active, they are able to check this thread to determine if a module has crashed or become stuck in a loop. In cases where a module with an execution thread does fail, a Manager can release the resources the module may have been using, and reconfigure any modules that may have been dependents of the failed one. Similarly, if a module fails to respond to a call to Pause in a timely manner, it may have crashed or be a malicious module. In such cases, a Manager can forcibly halt the module's execution and release its resources. If the module is believed to be malicious, the manager can completely remove it from the system, by deactivating any active instances and deleting the DLLs which hold the module. The ability of the Manager to monitor modules in this way adds an extra layer of security to the *Domino* system, and makes it more robust—allowing dynamic reconfiguration when modules fail. General security concerns, as well as privacy concerns, regarding *Domino* are discussed as issues for future work in Chapter 8.

DAF, like the other components in *Domino*, is written entirely in C# for the .NET platform. Each DAF 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 DAF uses

the .NET reflection capabilities to dynamically instantiate the class at runtime; enabling its use from that point on.

7.1.2 Example use

To understand the manner in which the three sections of *Domino* interact, and to understand better each section individually, it is useful to walk through a real, yet relatively simple, example from an actual *Domino* application in which three users participate. This is now described with reference to the first *Domino* application created, a simple mapping tool.

The core application simply displays a map and allows the user to pan and zoom the viewport onto the map (Figure 40). The application is built using the *Domino* system and the application's main form, which is responsible for displaying the main GUI and all the widgets within it, is a DAF Manager, capable of providing all the dynamic adaptive functionality this entails. Within the application, in two separate threads, the other two *Domino* components of communication and recommendation are also active. Thus, the application is continually searching for peers to exchange data with, and logging module use. The GUI component, which provides the core map view with pan and zoom functionality, is a DAF module and so its use is recorded in the logging process.

Figure 40: Overview of a Domino test application. The left image shows the user interface displaying map, pollution layer and GPS location. Each of these is actually a separate *Domino* module that may potentially be recommended, transmitted, installed and executed on peer devices.

Three users, who will be referred to as *A*, *B* and *C*, all run this *Domino* system on PDAs. From the possible modules, *A* only has the map whilst both *B* and *C* have the map and additional GPS and pollution modules. All the participants have spent some time using their systems, so logs of their use of the three modules have been recorded.

A walks close to an infrastructure access point and the communication section of *Domino* automatically switches his PDA wireless card to infrastructure mode, sets the card's SSID to that of the access point,

and uses DHCP to retrieve an IP address. It then starts broadcasting the existence of a *Domino* history server to the network whilst simultaneously listening for similar broadcasts from peer devices.

B also enters the range of the same infrastructure access point and her device goes through the same steps. As both devices are broadcasting and searching for peers, they discover each other's presence on the network at approximately the same time (within one second). The communication section on *A*'s device notifies the *Samara* section of *Domino* that a peer is available. *Samara* then informs the communication section that it wishes to gather usage history from the peer by sending it details of the request to make. As it has not encountered any peers before, it simply asks for the history of the user of the device itself to be sent and, if time permits, any history logs from other users stored on *B* to be sent subsequently.

The request is made, and the server running on *B*'s device services it by using *Samara* to query the history data and deliver it back to the communication section from where the results are sent to *A*'s device. The *Samara* instance on *A* stores the newly gathered history and reruns its recommendation algorithm to check if any new recommendations are available. Recommendations for both the GPS and pollution modules are generated but the GPS recommendation is ranked higher (simply because *B* has used it more frequently).

Both recommendations with rankings are sent to the *DAF* running on *A*'s device, which through the Manager displays the recommendations. As the GPS layer is ranked highest, *A* selects to try it first. The *DAF* section now requests that the *Samara* instance retrieve the recommendation. This, in turn, uses the communications section to contact *B*'s device once more and to retrieve the actual GPS module which is delivered back to *Samara* and on to the DAF. The DAF then instantiates the code in the DLL and checks for any dependencies the module may have. As the GPS module contains a map overlay it returns a dependency for a base map. The DAF Manager is aware that a base map is already running and, as the map returns true to its FindDependency call, elects to connect the new GPS module to the base map before execution. It is helpful to note at this point that the GPS module is not simply a map overlay alone but also contains code to search for and connect to a GPS device, and to parse the NMEA it outputs—and that all this code is contained within a single DAF module. The GPS module's Start method is called, and it connects to the GPS, parses the output and finally displays the user's position on the pre-existing map module.

A subsequently moves away from the infrastructure access point and, as his device detects this, the communication section automatically switches the wireless card to ad hoc mode, sets a predefined SSID and assigns a self-generated IP address. It again begins to both advertise its own presence and search for peers. During this period, the *Samara* instance on the device detects that activity is low and takes the opportunity to run a comparison of *A*'s usage history against the other histories on the device. Although the only other history on the device is that of *B*, it is still recorded as being similar to the current user's history.

Subsequently, *C*'s device is encountered and a request for history data is made. However, as it is known that *B*'s history is similar, the request this time asks for history data from *B* first, if it is available, before asking for data from *C* or any other users. As *C*'s device contains no data from *B* it returns only its own user's history data. Again, *Samara* runs the recommendation algorithm and this time generates a recommendation for the pollution module alone. As the input to the recommendation algorithm contains a list of the currently running and recently used modules, it never recommends anything the user currently has active.

A accepts the recommendation for the pollution module and the request is made to fetch it from *C*'s device. However, before the module can be delivered across the network, the devices move out of range of one another. As *A* has already accepted the recommendation, the failure to gather the actual module is noted by the *Samara* instance on the device and a record is made to seek the module from peers encountered in the future. If *A* subsequently encounters *B*, *C* or any other user who has the pollution module, *A*'s device will immediately request and receive it from them. Once the pollution module is gathered, the user will be asked if they still wish to install it and, if so, it will be dynamically installed and run on their system.

This simple example demonstrates how the three sections of the Domino system combine to provide a system that is able to monitor continually the activities a user conducts on their device and adapt to suit their needs better. As Domino logs the activities of multiple users, there is a varying skill set between its users. This allows experienced users, or users who have more time to experiment with the available modules, to discover good combinations of modules that work well in particular tasks. The Domino system allows these combinations to be tracked and subsequently recommended to less experienced users. This tracking of experienced users, and the filtering of their findings and knowledge to less experienced, mimics the patterns of sharing software Mackay found occurring in normal office environments [109]:

> *The act of customizing one's own software applications is usually viewed as a reasonably solitary task. ...However, because people have varying levels of desire and ability to customize, and have limited amounts of time, they often look to friends and colleagues for customization ideas. Borrowing customizations has numerous advantages for individual users. They can reduce both the time spent learning how to customize and the risk of making errors, which increases the time available for accomplishing actual work. They can also experience how other people work, find out new ways of doing things and benefit from each other's innovations.*

The adaptation *Domino* provides may be viewed as a partial automation of the processes Mackay describes here—taking the role of *translator*, filtering efficient software customisations from highly experienced users to the rest of the community of users. However, *Domino* goes further in that it does

not blindly recommend software from experienced to less experienced users, but actively monitors a user's context and provides continual adaptation around the current activity and context.

7.2 Applications of Domino

In addition to the behaviour described in the previous example, *Domino* provides other advantages. One of the most important is in disambiguating which modules should be utilised when fulfilling dependencies in cases where there are multiple possibilities. For example, it is easy to envision a slight extension of the previous example in which a system is running two map modules rather than one.

When a recommendation for a new overlay module is accepted it could potentially be connected to either of the existing maps. In cases like these, *Domino* is able to analyse the past usage history of the maps and their existing relationships with other overlays to determine which map is the most suitable candidate. It achieves this by referring back to *Samara*, and asking that it runs the recommendation algorithm but considers only the new module as the sole item of context. The results returned from such a Recer query are simply a ranked list of the modules the new overlay is commonly found to appear in conjunction with. We refer to these as *associated* modules. Thus, if the new module is one that provides an overlay of the locations of museums, it might be found that it is associated with other tourist-based overlays such as ones displaying restaurants, clubs or theatres. Once the list of associated modules is gathered, *Domino* can use it to check which of the currently active modules, in this case which of the two base maps, contains the largest overlap of associated modules. Thus, if one of the two maps contained mainly other tourist overlays, whilst the second map contained pollution information, *Samara* could determine that the first map is the most suitable candidate in which to place a new museum overlay, based on the past usage of the modules involved..

This automated disambiguation of where to place incoming components greatly benefits *Domino*'s core functionality as it directly addresses one of its goals—the reduction of time spent understanding a system and how it could be improved through the addition of new plugins. That is, *Domino* not only discovers new functionality and improvements to a system but also aids in inserting the new component smoothly into the running the system in an appropriate place.

It is hoped that *Domino*'s ability to spread functionality through a community will permit functionality to cluster at locations it is most appropriate. For example, over time, it could be expected that users visiting a museum would come to use similar modules at that particular location. Indeed, it is possible that programs specific to the museum itself, such as a guide, could be created as *Domino* modules. As most visitors to the museum would find such a module useful, it is likely that it would be widely used at the location. If the number of users and time spent in the museum allowed sufficient overlap between visitors, the module's functionality could perpetually pass to incoming visitors – the functionality the module provides would essentially cluster at the one location. When such behaviour is desirable, it can be encouraged using the *Bothy* technique described earlier. Placing the device in a location from which it is likely to be able to connect to transitive devices allows it to collect the

maximum number of visitor history logs and modules most relevant to the location. Thus, even if a new visitor selects a relatively quiet time to visit the museum, he or she will still receive *Domino* module recommendations from the on-site device rather than having to rely solely on chance meetings with peers. Furthermore, as the device it does encounter has been able to accumulate the collective modules and history from all previous visitors, it is likely to provide a far higher quality of recommendation than the average peer encounter. Functionality and information may naturally cluster around a particular location and, if required, can be aided through the use of the *Bothy* technique.

The fact that *Domino* was designed to spread functionality and information throughout a community of devices results in it being capable of spreading modules not only between separate users, but also between the many devices owned by a single user. For example, one may use a tourist application written on top of *Domino* on one's desktop machine at home to conduct a pre-visit of a city one is planning to visit. This could be similar to the pre-visit facility offered by the *George Square* system. As one is likely to either dock one's PDA with the desktop machine or have it near whilst conducting this activity, any *Domino* application running on the PDA would be able to exchange information with the application running on the desktop. Thus, if one added modules to the application on one's desktop as one found them useful—perhaps a tool for retrieving public train or bus schedules—the same modules could be automatically copied to the PDA. This is distinct from synchronisation of information and could be the first form of functionality synchronisation. As the devices of a single user move in and out of range of one another, and as the user is likely to have somewhat similar usage patterns on each device, the modules installed on one device would likely be recommended to others. Obviously, not all modules would be copied to all devices, and it is likely that *Samara* would recommend only those suitable for use.

One shortcoming of *Domino* is that the recommendations it makes are only for additional modules, ones that the user does not have installed. This may pose a problem, as *Domino* never recommends that a user *remove* a module from a configuration. Over time, if a user continued to accept recommendations for modules, and did not manually remove other modules, any *Domino* system has the potential to become unmanageable to both the user and the system, as it is overwhelmed by an ever-increasing number of modules, which increase complexity and consume resources. Whilst a user can manually deactivate and remove any module they desire, this negates the benefits *Samara* provides in automating the recommendation of software. Recommendations for the removal of *Domino* modules is important in systems where a large number of modules are available, and is clearly a vital item for future work. As such, it is discussed further in the Future Work section of Chapter 8.

7.3 Castles

In order to verify that the *Domino* system works as expected, and that users find value in using it, a test system was built using *Domino* and a trial was conducted. The trial was primarily a technical one, conducted to ensure that the *Domino* system behaved as hoped within an application—with peers discovering one another, exchanging recommendation information and modules and adapting to the

users behaviour as expected. This section describes the game itself, the set up of the trial and some initial findings.

7.3.1 Game description

The system constructed to test the *Domino* architecture is a mobile strategy game called *Castles*. This section describes the game and its rules from a standard player's perspective, before detailing the game technically in terms of *Domino* and modules.

Games have a wide social and financial impact, and form an interesting application area in themselves. However, we selected a game to experiment with *Domino* for several reasons. Firstly, as a game does not have to be constructed to fit a particular external task, it can easily be suitably constructed in a manner that allows the exploration of the specific technical issues raised by the wider research. Rather than forcing the architecture into an existing problem where it might receive only periodic or light use, a game can be deliberately developed in a way that ensures maximum use of, and experimentation with, the components of particular interest to the research.

Furthermore, as users are frequently keen to play games, and often spend substantial periods of time experimenting with games they find engrossing, they are more likely to find ways to stretch one's designs, assumptions and concepts. 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. Games are also an application area in which users are already often involved in radical re-engineering of systems. Many games now have extensive 'modding' communities (such as Half Life[31]) and so the idea of an adaptable or changing application is not one which is alien to game applications or to the people who use them.

7.3.2 Gamer's perspective

From a game play perspective, Castles is similar in theme to other strategy games where the player must create a building infrastructure, which in turn allows for the construction of armies—such as in the Age of Empires, Stronghold or Settlers games[32]. The majority of the Castles game is played in a solo building mode. Players begin the game with a low number of base resources (stone, wood and food) and must utilise these to construct buildings, which will in turn produce more refined resources such as shields or swords. Each building costs a certain amount of resources to construct initially and, once built, consumes resources each game cycle in order to output more refined resources. For example, a blacksmith building might take 10 stone and 5 iron units to construct initially and, once built, it will consume 5 iron units and 1 food unit each game cycle, and will output a single sword. Players must ensure that the buildings they construct combine in a manner that allows all the building inputs to be fulfilled each game cycle. The rate the game cycles controls the speed of the solo game section, and is set at once cycle every ten seconds. This value was selected simply because we find it is

[31] http://half-life2.com/ (game information), http://www.planethalflife.com/features/motw/ (modding information)
[32] URLs where more information on these games can be found are http://www.microsoft.com/games/empires/, http://www.stronghold-game.com/ and http://www.settlers4.com/

neither too fast nor too slow—players do not become bored waiting long periods of time for resources to accumulate before they can construct new buildings, and they do not become confused or anxious because the game is moving too quickly to follow.

Buildings can be made more efficient through the use of tool upgrades. Tools are constructed in the same manner as buildings (with a certain resource cost) and can be added to any building the player owns. However, they affect each building in a unique way. For example, the scythe tool has little effect (only a 20% increase in output) if combined with a fisherman's hut but will more than double output if combined with a farm. In short, tool upgrades can have differing affects depending on which building they are added to.

The buildings that produce the end army units, the most important for the next section of the game, are the only ones that do not automatically consume resources each game cycle. Instead, once the other buildings have produced the required goods, the player can select the building and then click to 'buy' the unit. For example, if a player has accumulated ten swords and shields, he or she can select the barracks and click to 'buy' ten soldiers. When the game begins, there are over forty types of building and eleven types of army unit available to the player. These can be combined in any way the player chooses so there are many techniques to construct efficient building infrastructures and many paths that will lead the player to construct a suitable army. However, in general, the more efficiently players construct a building infrastructure and utilise building upgrades, the more resources they will be able to produce, and so the more army units they will be able to purchase. The interface displayed to users whilst playing this portion of the game, and creating buildings, is shown in Figure 41.

Figure 41: Castles main game interface.

The aim of the game is to win battles against others, and once a player has constructed an army he or she can use it to attack other players. When a player chooses to engage in battle with another, they are

given an opportunity to select which units they wish to build the three rows of their army—front, back and reserve. Like the building part of the game, battles are conducted through a series of game cycles although this is largely transparent to the user who simply perceives time passing. The interface displays a representation of the battle with the front and back rows shown attacking one another. This display changes after each battle cycle so that the battle progress is shown within it – inevitably with one army beginning to overcome the other. At any point the player can elect to send in the reinforcements they selected before the battle, attempt to retreat, or surrender. In any battle, the winner is rewarded by receiving some of the defeated player's resources as well as having the results of the battle recorded on their device to display to other players at a later date.

7.3.3 Technical perspective

Throughout the design process Castles has been developed to take advantage and experiment with the *Domino* system wherever possible. The entire game is built using *Domino,* and each user interface component, including the map on which buildings are placed, is a *Domino* module. Every individual building, building upgrade and army unit is a *Domino* module in itself. Thus, 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.

The *Samara* recommendation system in the game works as previously described—continually monitoring the player's own module use and collecting history information from peers whenever possible. This allows for players to be aided throughout the game by the use of recommendations. As there are a high number of buildings, adapters and units, there is significant variation in the types of society (module configurations) that a player may create to support an army. 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. When new buildings and units are available to be run but not yet instantiated, the user is notified of the new additions by the use of highlighting in the menu of available buildings.

The way in which Castles finds and recommends building modules to the user, and automatically and appropriately installs them if accepted, is an example of the typical functionality *Domino* can provide. In many systems users can struggle to identify when new functionality exists, where it can help, how to install it, and with what other existing software it may work well in combination with. Castles exemplifies the general situation here, demonstrating a practical example of how *Domino* may be applied to aid users in these tasks. The fact that installation, and activation, of new modules can be achieved at runtime, additionally gives *Domino* its adaptive functionality.

Samara monitors the player's own module use, and by comparing his or her current building configuration to the history of others', can generate recommendations on how the player should proceed. Such help may aid the player in strategy games such as Castles, where the number of possible choices can seem overwhelming for both new and experienced players. Again, this example within

Castles is an instance of the more general and common problem of users being overwhelmed by the amount of available software or information, or not having time to learn the intricacies of the many possible software items, before making a decision on which to install and use.

If the user desires, he or she can get additional information about a recommendation, such as the dependencies or the modules most frequently used in conjunction with it in the past in similar contexts. This information, obtained in a pop-up dialog by clicking the recommendation information button in the build panel, can help the player to understand better how the module might be used (Figure 42).

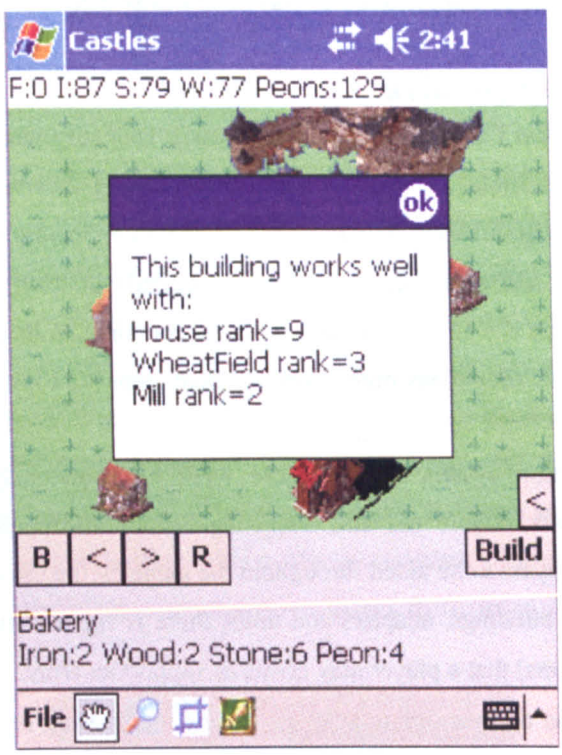

Figure 42: Recommendation pop-up in the Castles game. The pop-up displays what modules a module has been used in combination with previously.

Thus, 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. On the other hand, *Domino* does not go too far in automatically loading and running modules. It presents module recommendations in a way that permits a user to see them as he or she plays, and find out about their past use. 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.

When a Castle's player decides to construct a building upgrade tool, *Domino* is able to recommend which building it is best to place the tool into. For example, if the player does construct the scythe tool previously mentioned, *Domino* is likely to recommend it be combined with the farm. *Samara* is able to determine this recommendation as appropriate, as the history logs suggest that other players have

realised, through experimentation, that this building is one in which the scythe is particularly beneficial. Recommendations for building upgrades are vital as, unlike with buildings where users can view statistical information about resource inputs and outputs, with building upgrades the user is provided with little alternative information to determine where such upgrades will help.

A final aid the *Samara* recommendations provide to the user in the interface occurs when new modules themselves are available. When two players' devices are within wireless range, one may choose to attack another. Behind the scenes, *Domino* also initiates its history-sharing and module-sharing processes. When a battle commences, both players select from their army the troops to enter into battle. Players receive updates as the battle proceeds, and at any time can choose to retreat or concede defeat. At the same time, players can talk about the game, or the modules they have recently collected, or modules they have used and either found useful or discarded. After each battle, as the devices will have been exchanging history information in the background, the opportunity is taken to allow players to exchange *Domino* modules. In order to extend the length of the game, an artificial limit is placed to allow only one *Domino* module to be exchanged per battle. In a large community of users such a limit would not be necessary as there would likely be large variability in the number and combinations of modules on the peers encountered. However, as our trials were conducted with a limited number of users allowing an unlimited number of modules to spread could quickly lead to all devices carrying all available modules after only one or two encounters. The screen displayed after a battle presents a list of modules available on the peer's device. Although each available module is displayed, the ones that *Domino* recommends are highlighted—as well as being ranked in the order *Samara* recommends them. The user does not have to follow recommendations if he or she does not desire. Indeed, it is believed that the most experienced players, those who have had most time to experiment with the game, will generally ignore the recommendations. Instead, it is the novice and average users who will benefit most by being guided through the system in this manner. This echoes *Domino*'s original concept, which is more generally applicable, in which users who have had time to experience a large number of modules provide information that aids others, who simply do not have the time to invest in experimenting with the numerous possible combinations. *Domino* recommendations from the *Samara* engine permeate throughout the game community and *Samara* continues to provide recommendations on which to receive and how to utilise them in order to guide the user in several different ways. The visibility of successful adaptation is therefore part of the game, with successful players having to continue to innovate if they are to maintain an advantage over others.

As Castles is built on *Domino*, Castles' system components interact almost identically to the three *Domino* components themselves; which have been described earlier. The only significant addition is that of another instance of the communications section, which this time broadcasts, listens for and negotiates the transfers required for the battle stage. The game interface and algorithms are all contained within classes that are accessed from the *Domino* Manager within the application, which in this case is the main UI form.

7.3.4 Trial

With the exception of several very small test applications, such as the map with GPS and pollution layers example, Castles is the first application that makes substantial use of *Domino*. Both the communication section and *Samara* had been trialled previously in *Feeding Yoshi* and the *Samara* mapping application respectively, and found to be sufficiently robust. However, the integration of the three *Domino* components and the DAF itself had not previously been built into and tested within a substantial application.

Despite the fact that this trial was primarily a technical one, we also took the opportunity to get some initial feedback from users about whether they understood the module recommendations and were comfortable using the interface to select and install the modules. This was achieved simply through short interviews at the end of the trial.

The trial involved 4 participants playing the Castles game both on their own and with others during battles. Each player began the game with 54 common modules – modules of which every player had a copy. These modules consisted of 33 buildings, 10 building upgrades and 11 army units. In addition to the 54 common modules, each individual player also began with 5 modules that were unique to only him or her. For example, only one player began the game with the catapult-building module. It is realised that from a game play perspective, this likely introduces an inherent unfairness to the actual game, although it is believed that any such affect will be minor. As *Domino* systems automatically transfer information when in range, the trial was run in cycles of two repeating stages in order to allow time after each stage to ensure the spread of *Domino* modules was monitored.

Firstly, a period of isolated play, in which participants were able to concentrate on constructing their building infrastructure and army was conducted. During this period *Domino* logged players' own module use but, as no other peer devices were in range, did not exchange log data or modules. As the devices began the trial with an empty *Samara* database, recommendations on which buildings to construct were not available during this first round of isolated play. This period of isolated play was followed by a shorter period of play in which two participants met one another and conducted a battle. 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 phones or PDAs during a normal day and randomly encountering other players. During the battle period, *Domino* continued to operate in the background and exchanged history logs and recommended modules. In this trial, the *Domino* Manager did not request that users authorise the installation of recommended modules. This meant that every module recommendation that was generated was automatically accepted, copied over the network, and made available inside the game interface, rather than requiring users to authorise them. In short, the way *Domino* was applied meant that after each battle players automatically gained new building modules that could then be recommended to players to construct in subsequent periods of isolated game play.

In addition to the modules themselves being transferred during the battle, history logs from the current opponent, along with any other players' histories that were also held on the opponent's device from previous encounters, were exchanged. After the initial battle, players begin to receive building help through recommendations for the first time during their second stage of isolated play. When the player begins to construct a new building from this point on, he or she always has at least one recommendation available for which building to construct. In this way, players gained new functionality over time, and were guided in how to apply that functionality within their current module configuration. The modules players received were based on their current configurations, and so as new modules were recommended, installed and activated, the functionality provided adapted around each player's own use of the system; around his or her context of use.

7.3.5 Findings

In the trial the *Domino* technology used performed as anticipated. 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. There were no crashes or failures and the application ran without incident throughout the trial.

During the trial, an average of 2141 history entries were exchanged between peers during their encounters, and it took an average of 10.06ms 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 99.45 *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 3 other participants, the average size of the history database on each device was 1452KB.

Following the trial, participants were briefly asked about their experiences with the game in the hope that suggestions that would improve future versions would be made. 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 began receiving recommendations. However, as none of the trial participants had a computing science background, none of them realised that the modules that were transferred could contain potentially harmful code. Whilst participants felt secure in the controlled trial environment, the risk from harmful modules is real. This raised both security and privacy concerns 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 *Samara* 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.

Prior to this trial, *Samara*, both in the mapping application and in *Domino*, had always transmitted a unique ID with every history log entry. This is important for the recommendation algorithm as it needs to identify co-occurrences of items within a single usage block, and to identify usage blocks it must be known that all entries were generated around the same time by the same user. Although the device's unique ID is not obviously displayed to the user at any point in Castles or on the PDA itself it would be possible, with enough effort, to use it to link a set of *Domino* history data to a device and therefore to a user. To combat this, an alternative to using the device's ID has been implemented. The system behaves the same but rather than using the device's ID, a completely random ID is generated for use in *Domino* and used during all *Domino* transactions. This technique greatly increases the complexity required to identify a user, as there is no direct link between their log entries to the device itself. Furthermore, as any *Domino* device can hold logs from many users, not just the owner of the device, it is impossible to be certain if data transferred from a peer was actually originally generated on the peer. As the randomly generated ID has over 10^{16} possible combinations, it is improbable that any two generated IDs would be identical. One of the main advantages 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 be high. However, if it was found that previous data had been received from this user in the past, in addition to this new set, the chance of recommendation could be significantly higher. This technique, of using a randomly generated ID to replace the device ID is the one currently implemented in *Domino*.

In addition to the use of a random ID, the ability to halt all logging over a period of time has been added. Users can choose to stop their *Domino* system from creating any new log entries but still have the recommendation system, and peer exchanges, continue to provide its recommendation and software adaptation functionality using the data it has already recorded. Finally, a list of keywords not to be logged can be entered by the user. Before history entries are generated and stored, this list is consulted and if any of the words are found inside the entry, it is completely discarded without being used or stored. For example, a user called Charles might enter his own name as a keyword, and subsequently any *Domino* entry that referred to a web visit to a URL which contained the word 'Charles', such as 'http://www.example.com/mail/~Charles', would be discarded by *Domino*. Whilst the benefits of a keyword list are perhaps not immediately clear with respect to Castles, in which only modules are recorded, they are obvious when considering that *Domino* also has the ability to record and recommend not just software modules but other data, such as the text of a URL, contents of a document, or a location, as in the initial *Samara* mapping application.

Security is a serious issue for any system using mobile code that moves between different devices, and has been considered following the initial trial. One 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 in which the module provides at least some useful functionality, and does not exhibit any negative behaviour, switches to executing malicious code.

Currently, one of the most widely used techniques for deciding which applications to trust is that of signing, in which a trusted authority analyses applications or modules, and assesses whether they may be harmful. 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 any signed authorisation, or lack of.

Whilst employing signing for *Domino* would provide an accepted solution to security concerns, there are severe disadvantages that have, so far, negated its implementation into the *Domino* infrastructure. 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 coded in the .NET language, it already runs through the CLR (Common Language Runtime)—basically a virtual machine.

Another possible solution is to use a permission–based model, in a manner similar to the Java language and to most modern operating systems. For example, if a *Domino* module wanted to access a file on the local device, it would first have to ask permission from the user who could deny, accept once or accept forever the module's request. Whilst this method is employed by many languages that run on virtual machines, it would possibly be too intrusive to users in a *Domino* environment. Previously, this method had usually been used where the number of new modules or applications was relatively low, and so the user was required to intervene on an infrequent basis. In a *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 obtain expert tools quickly, 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* 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 [58]. 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 final potential technical 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 he or she could, after removing it, add it to a list of known bad modules. From then on, the list could be transmitted to any encountered *Domino* peers. A *Domino* client which had received this information could then refuse to accept the module if it were recommended. 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, adding it to the blacklist of modules to avoid. 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 had done any damage, the spread of the information that the module was malicious would, hopefully, be faster than the spread of the module itself. In this way, viral outbreaks of malicious modules could generally be prevented. However, this solution is not perfect, as 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 'honeypot' implementations, this has been shown to be particularly effective at stopping the spread of conventional computer viruses [73].

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 trustworthy module. Alternatively this history information could be fed into an algorithm such as that in [33] or [147], to give a calculated level of trust. Although this technique, a form of distributed trust model, 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 we are researching these and other possible solutions and will continue to do so in future work, we have not yet settled on a single robust solution that we fully trust. For this reason, we have so far avoided creating 'mission critical' applications based on the *Domino* architecture 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 broader security issues, limiting the potential damage to users' devices.

The initial trial of *Domino* in the Castles game was primarily a technical one, ensuring that the technology used behaved as expected, and practically demonstrating that high levels of adaptation, based on users' context, are possible in mobile systems. Whilst the trial did fulfil this primary goal, it was admittedly small, and there are many questions left open, particularly regarding the user experience with the adaptation a *Domino* system provides. For this reason, a second larger trial was conducted in December, 2006, involving a greater number of participants using the system over a longer period of time. The goals of this trial were to examine the user experience, discover if an adaptive system is acceptable to users and if they value the recommendations for software modules that *Domino* provides. Results from this larger trial are currently being analysed, and are an item of future work discussed in Chapter 8.

7.4 Conclusion

The *Domino* system implements the fourth, and final, piece of infrastructure that was identified as required for mobile, peer-to-peer systems at the conclusion of Chapter 3:

- A method for adapting the system itself around the user's activities

Adaptation is an important element in mobile systems, as an increasing amount of software becomes available, and users find it harder to manage the software, and combinations of software, on their devices. Adaptation that occurs in an *appropriate* manner, and at *appropriate* times, is vital within the mobile environment, where context can generally change more rapidly, and at a greater level, than on desktop systems.

Domino allows a far higher level of adaptation than many adaptive systems on desktop machines are able to achieve, and the adaptation it provides is based primarily on one's current context and one's own, and peers', past actions. This is a novel approach compared to most existing adaptive systems, which are generally not aware of the user's current task, or their context. Indeed, outside self-healing systems, which adapt only when failure occurs, most systems in the area are *adaptable,* but not *adaptive*, relying on direct user input to control adaptation.

In addition to providing a novel form of adaptation, *Domino* actively seeks, collects, and installs, modules from peers—supporting adaptation through a peer-to-peer community, which was not previously possible. The *Castles* game demonstrates one application of *Domino* in a mobile, peer-to-peer environment. It is clear from the trial that adaptation around a user's activity is possible, and that functionality can be adapted in the mobile environment whilst applications are actively running.

Whilst work is ongoing to further improve the *Domino* infrastructure, and is discussed in the Future Work section of Chapter 8, *Domino* already allows a new level of *appropriate* adaptation in mobile systems. Research on the *Domino* system was published at the Pervasive conference and can be read in [11].

8 Conclusions

In this chapter a summary of the results and contributions of the thesis are presented, as well as a discussion of its limitations and proposals on how they could be addressed by future work.

8.1 Summary of thesis

A review of existing literature in Chapter 2 revealed that there is an overwhelming desire in the research community to create mobile applications that are adaptive and context-aware, rather than simply being reduced versions of desktop software. Previous work identifies that context-aware systems are of substantial importance to the mobile field, and that system adaptation is a necessity if flexible mobile systems are to be implemented. In addition, location is singled out as being the most vital item of context information in mobile systems. Furthermore, problems with existing positioning, communication and peer discovery techniques are highlighted.

Following the literature review of Chapter 2, an analysis of two mobile systems was conducted. The first analysed the *Lighthouse*, a system that failed to be particularly mobile, whilst the second investigated *George Square*, a mobile system aimed specifically to address failings apparent from the *Lighthouse*. Both investigations result in the confirmation of many problems stated in previous literature and the formation of design guidelines for mobile systems aiming to overcome many of these problems. Certain problems are found to be caused by a lack of infrastructure tools available to the mobile field, and four of these are implemented over the subsequent chapters. These specifically address issues discovered in the literature review, and are reinforced by the analysis of the *Lighthouse* and *George Square*, such as a positioning system that makes location information available to mobile devices and an infrastructure that facilitates dynamic reconfiguration and adaptation based around a user's context on mobile devices.

In total, seven systems (*George Square, Treasure, Feeding Yoshi, Navizon, FarCry, Samara* and *Domino*) were designed, implemented and trialled. However, the amount of useful code developed for mobile systems has been so high that it has been impossible to cover all of it even within this thesis. Through the implementation and trials of many mobile and mobile, peer-to-peer systems discussed in chapters 2 to 7, and through substantial experience with mobile, peer-to-peer development in general, further guidelines and categorisations were extracted where relevant throughout the thesis. These are collected and summarised in the next section.

The substantial number of systems implemented, as well as their trials and analysis in the course of this thesis, has facilitated a holistic approach to a variety of issues within the mobile field. Whilst a holistic approach is generally typical of Equator's research, and has been encouraged at Glasgow, it has been particularly helpful to the work conducted for this thesis, aiding in the provision of general design guidelines, and the identification of issues that span a range of mobile systems.

8.2 Contributions

This thesis set out to advance the mobile, peer-to-peer field. In particular, it aimed to address the problem that, despite mobile devices having the potential to deliver highly adaptive, context-aware applications, the majority of mobile applications today remain static, inflexible versions of desktop applications. The fact that this is indeed a problem in the field was strongly supported by findings in the literature review, in which an overwhelming number of researchers paraphrase the same issue.

The problem is addressed in several ways in this thesis. Firstly, a set of design guidelines is presented. Adhering to them should lead to more highly adaptable and context-aware mobile, peer-to-peer systems:

- Support users changing roles
- Support fluid group dynamics
- Avoid reliance on pre-authored content
- Allow information to be shared and stored by multiple peers
- Avoid centralised peer-to-peer architectures whenever possible
- Expose characteristics of underlying infrastructure where appropriate

Each guideline was discussed in detail, and advice and methods for implementing each were provided. Over the subsequent chapters, many mobile systems were implemented, and each attempted to follow the guidelines in order to demonstrate their relevance within the mobile field.

The issues arising from a reliance on pre-authored content were highlighted in one of the guidelines. In addition, a categorisation of content systems was offered, and compared and contrasted with Triantfillou et al's categories of peer-to-peer systems. This categorisation allows designers to select combinations of peer-to-peer and content-type architectures in order to avoid problems which can be caused by centralisation, or by the requirement of authoring substantial amounts of content for systems.

Four pieces of infrastructure were identified as being necessary for implementing dynamic, context-aware, mobile, peer-to-peer systems but currently missing in the field:

- A mechanism for intelligently selecting which networks to use and for reliably discovering peers on these networks
- A hybrid positioning system that requires no initial setup and yet has high availability in providing location both indoors and outdoors
- A method for providing and distributing data within a peer-to-peer community
- A method for adapting the system itself around the user's activities

For each of these infrastructure items, a chapter was devoted to its discussion and implementation. Each one is fully implemented and demonstrated within applications that have been successful either within the research community or commercially.

As the first item of infrastructure—a method for intelligently selecting which networks to use and for reliably discovering peers—was dependent on an underlying communication technology, an investigation into 802.11, Bluetooth and GSM was conducted. This examined the two most important factors relevant to mobile, peer-to-peer systems; the length of time it takes for peer devices to discover one another, and the amount of data that can be transferred between peer devices during encounters. This trial resulted in 802.11 being a clear choice over Bluetooth or GSM in mobile, peer-to-peer systems. The final solution presented an enhanced wireless driver combined with SDS providing a comprehensive solution for reliably and rapidly discovering peer devices in a mobile peer-to-peer environment. The solution enables peer-to-peer mobile systems such as *Feeding Yoshi*, *Samara*, *FarCry* and *Domino* to be created far more rapidly than was previously achievable. Furthermore, designers and users are no longer forced to pre-configure the device's network settings to a single, prohibitive network that works for only a single system, as had to be done in [62] and [23]. Instead, no pre-configuration is necessary, many peer-to-peer and standard applications can run simultaneously, and users can continue to access the Internet on their devices in a manner that is actually easier than was previously possible. This behaviour, allowing normal network usage when available, will be increasingly important as more peer-to-peer applications, and applications that are designed to be used over *longer* periods of time, are developed. *Feeding Yoshi* is an example of this type of mobile application, one which is designed to be played over a long period of time, carried with the user constantly and available at any location. If such applications are to be accepted, and used over long periods as intended, it is vital that they employ the technology developed as part of this thesis, or akin to it, to allow users to continue to use their mobile devices for a range of purposes, rather than being forced to configure them for a single, specialised task.

As the literature review identifies, location is the most fundamental piece of context information in mobile systems yet there are currently no systems that can provide a location reliably in mobile environments. This problem was addressed by the implementation of a novel hybrid-positioning system titled Navizon. Navizon combines GPS, GSM and 802.11 positioning techniques to provide high accuracy and uptime to mobile applications. Navizon is currently the only system that successfully combines all three of these positioning systems onto a single PDA or phone. With Navizon, location information is available far more often than it has previously been with other systems, and thus new forms of mobile application can be constructed. Whereas previously mobile system designers could only plan to use location information either indoors or outdoors, but not both, they are now free to assume that location information is always available. The fact that the system is compatible with an extremely large number of PDAs and phones, and that it requires no additional hardware and runs on a single device (unlike *Place Lab* which requires both a phone and PDA working in unison to provide both 802.11 and GSM positioning) means that mobile applications which rely on

position information can now be rapidly developed using many existing devices. Furthermore, because Navizon adheres to NMEA standards and can fully emulate GPS hardware it can be rapidly retrofitted to work with many existing mobile applications. The contribution Navizon makes is stated by Kolodziej [99]:

> *Not until the availability of the NAVIZON wireless positioning system, GPS or cell tower-based systems were the only potential means for basing location-based services. As evidenced by the slow growth in LBS services, these traditional systems proved to be costly to implement as they required specialized equipment, and prone to problems with accuracy and reliability.*
>
> ...
>
> *With NAVIZON, any Wi-Fi or cellular product designed for broadband data networking can be used for location and tracking with no hardware changes. NAVIZON leverages the existing 802.11 hardware already resident in over 120 million Wi-Fi-enabled devices as of 2005, which is estimated to reach 430 million in 2009. Also, there are an estimated 40 million fixed Wi-Fi access points in the US, which constitute a wide wireless coverage.*
>
> ...
>
> *NAVIZON enables LBS applications and services with consistent and accurate location information. Consumer applications include information-centric service such as maps and directions, what is the nearest (Yellow Pages finder service), location-targeted advertising, photos/video, and MMS to mobile-to-mobile applications such as Buddy Finder and Social Networking. Enterprise applications include field force automation (managing fleets and tracking mobile workers), real estate and custom solutions to emergency services.*

In order to address the issue of distributing and maintaining data in a mobile, peer-to-peer environment a novel distribution method has been implemented which combines collaborative filtering techniques with epidemic routing algorithms. As part of this process, a categorisation of epidemic spreading techniques is derived from previous work and experimentation with the *FarCry* application and is presented and explained. The resulting infrastructure, *Samara*, has been explained and demonstrated as part of the larger *Domino* system and the *Castles* game. It provides a solution for both selecting data to route to peers and which data to maintain, and which to erase. It is believed to be the first epidemic spreading system relying on a recommendation system at its core. By using its recommendation system, *Samara* provides both a novel epidemic spreading algorithm and, importantly, a system for intelligently culling data on mobile devices.

The final piece of infrastructure implemented, *Domino*, specifically addresses the requirement of mobile systems to be highly adaptable to the user and their context. By relying on *Samara*, *Domino* continually monitors a user's context, records their actions and actively adapts, at runtime, to the

current tasks the user is involved in. By utilising *Samara* to distribute both the history logs and segments of functionality or code, 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, going beyond the level of dynamic adaptation that has been achieved in most adaptable desktop systems.

In conclusion, the work within this thesis addresses the two original research questions introduced at the outset of the thesis:

RQ1 *How can mobile developers design and develop more flexible and context-aware mobile systems?*

RQ2 *Are there software components lacking in the mobile field, hindering the development of flexible, context-aware mobile systems?*

The first question is addressed through both a literature review and the analysis of two systems. This process identifies existing problems and allows guidelines to be suggested which target these problems. From the same process many vital pieces of infrastructure that are currently missing in the mobile area are identified which relate to the second question. These are each implemented as part of the thesis and each is demonstrated in at least one successful system.

8.3 Limitations and Future Work

The work carried out for this thesis has resulted in a substantial number of design recommendations and the implementation of a large amount of new mobile infrastructure that was not previously available to mobile developers. Both the various design recommendations and the creation of vital infrastructure specifically for use on mobile peer-to-peer applications are, hopefully, of great benefit to future designers and developers. However, due to the fact that such a large amount of infrastructure was implemented, there has not been sufficient time to test each individual piece as comprehensively as had been hoped. Therefore, one of the most obvious items for future work is the continued testing and integration into future applications of the infrastructure created as a result of this thesis.

Undoubtedly, work following the guidelines identified and relying on the infrastructure created will continue at the University of Glasgow. Indeed, several such systems are currently in development at Glasgow including *Shakra*, a mobile, peer-to-peer fitness application that runs on phones. Work on improving *Domino* is also ongoing at Glasgow University, and as previously stated, a second, larger trial using *Domino* was completed in December 2006, the results of which will be included in Malcolm Hall's thesis. In addition, *Domino* is also being considered for use with Scott Sherwood's research, which investigates how a digital representation of a user can be displayed to others. *Domino* could be used in this work to provide adaptive behaviour in an application, which uses differing modules to display various aspects of a digital presentation of a user. All of the work implemented as part of this

thesis, with the exception of Navizon which is now a commercial application, has been implemented within the Equator group and, as such, will be made available as part of the Equator software archive in the second half of 2007. This includes the enhanced wireless driver, SDS, Samara and Domino—as well as the larger applications the infrastructure was tested in, such as *Treasure*, *Feeding Yoshi* and *Castles*.

Whilst work relying on the infrastructure developed as part of this thesis is ongoing, there are currently a few existing problems with several of the infrastructure components that have yet to be addressed. Whilst the wireless driver and SDS combination does greatly increase the chances of peer devices encountering one another, and allows users to continue to access standard wireless Internet connections when they are available, it does have a few failings. As discussed in Chapter 4, there are situations where two peer devices within range of one another can join separate networks, and thus fail to discover one another. One possible solution presented was the addition of a second network card, either a physical or virtual one. However, it is unrealistic to expect devices, or users, to have access to a second physical network card. Furthermore, this would introduce greater power drain, and increase the overall size of any mobile device. Unfortunately, a virtual network card is currently not possible on mobile devices, and there is no guarantee that such technology will be available for mobile devices in the near future. Therefore, further work may be necessary to overcome the situation in which peer devices do fail to meet when in range, which occurs when two separate groups of devices form whilst connected to different infrastructure networks. This may be achieved by creating groups of devices which switch networks simultaneously, or by finding another solution, such as relying on ad hoc mode even when infrastructure networks which *do not* provide an Internet connection are in range. A second problem with the driver is that it can only identify Internet connections available through networks that provide DHCP services and do not require a proxy. Although it might be possible to self-assign an IP address and test again for an Internet connection, this is not the behaviour of the driver, as networks which require such configuration may be set up in this way by the owner, in an attempt to stop others using it. As it is not our wish to attempt to overcome such issues, and use networks in cases where the owners do not wish them to be used, the driver does not attempt connections if they can not be made in this way. However, one small addition that could handle this situation would be to allow users to configure network settings manually if they desire. Currently, if a user's own network is not a standard configuration, the wireless driver will not be able to access the Internet through it, even if the user would like to do so. By keeping the driver as it is, but additionally allowing the user to manually configure settings for specific networks when desired, connections to non-standard 802.11 networks could be supported.

As discussed earlier, *Domino* does not yet recommend the removal of modules, although it does allow users to deactivate and remove modules manually. The implementation of a method to remove modules is a priority for the *Domino* infrastructure, and a technique to achieve this is already implemented and will be trialled in the near future. The technique essentially relies on using the current history logs, and the Recer algorithm. The logs are searched for configurations similar to the

user's current context, and then checking forward from that point to inspect if any of the modules in that set were removed from another's configuration. This produces a list of modules that may be recommended for removal, which are presented to the user in a similar manner as the recommendations for adding modules. If a module is to be removed, any dependent modules must be correctly handled. The most obvious solution may be to use the *Domino* system itself again, this time to identify if any other active modules can fulfil the dependencies required, and if so the modules could be reconnected to these. However, in many cases the deactivation of a single module may result in dependent modules having no other method of continuing to operate. In these cases, a method for clearly showing the results of deactivating a module to the user must be found.

Another limitation of the work in this thesis is that it has not explored privacy concerns that may arise from distributing and sharing information between peers in a mobile environment. This is simply because there has not been enough time to organise a trial with the required high numbers of users to investigate such concerns adequately. With regard to privacy, it is the belief of the author that in order to allay fears that sensitive information is not being shared, an investigation into the types of information users believe to be sensitive may first be necessary. Researchers are often oversensitive or misdirected when they make assumptions on the data end-users wish to keep secure. This is apparent in existing research, which demonstrates that users often have no concerns in sharing data assumed to be sensitive by the designers, and that it is difficult to predict with which groups, such as family and friends, users will freely disclose data [140], [14], [5], [91].

In order to investigate the issue of privacy thoroughly it may first be necessary to conduct large trials, possibly with thousands of users, in which the types of data modern mobile devices can now gather are available for sharing as doing so will aid in identifying which types of data users do indeed find sensitive. It is only once the types of data that are generated and used in mobile environments have been categorised with respect to privacy that techniques for attempting to keep such data secure can be contemplated.

Although Navizon currently provides one of the most available positioning systems today—making location a ubiquitous element—work must continue if it is to maintain high availability, and increase accuracy. New wireless technologies and protocols, such as UWB Bluetooth and 802.11n, are due to appear on commercially available products in the next couple of years, and work may be required to keep Navizon compatible with these new developments. More importantly, it is believed that Navizon's accuracy could be improved with the existing technology. The results of the Navizon accuracy tests conducted in this thesis, and available in the Appendix, surprisingly reveal that for small sample sets an unweighted centroid algorithm outperforms the weighted centroid algorithm Navizon currently employs. One obvious reason for this may be that the signal strength value read from most 802.11 devices is unreliable, and actually introduces a significant amount of error. As an example of how a signal can vary, Schwaighofer et al. point out that a single person's body shielding a phone antennae simply by being between it and the base station can attenuate the signal by up to -3.5dBm

[150]. Investigation of wireless signal strength reliability and propogation is an interesting research topic in itself, as it has a range of applications outside positioning. It is clear that many researchers assume such values to be accurate. Experience at Glasgow reveals that this is not necessarily true, and investigation into true accuracy levels, and the factors which influence signal strength readings, may be of value.

This thesis has concentrated solely on applications that run on mobile devices alone. However, information generated by mobile systems in self-generating content architectures may be of use when taken off the mobile device. One example of this has already been discussed in the *George Square* post-visit online blog which allows users to relive their visit or rely on it to inform others, such as family or friends, of the type of experience they had during their visit. Similarly, although not discussed in this thesis, the *Navizon* application and website support a number of services tied to the user locations *Navizon* generates; such as the tracking of groups of friends through the *Navizon* website on a Google Map and the display of discovered beacons which is in itself useful for multiple purposes. It is apparent that much of the data created by self-generating, peer-to-peer applications will be of use to either the user themselves or friends or colleagues if transferred back to desktop machines and made available over the Internet or intranets. As much of the work in this thesis, particularly related to *Domino*, necessitates the tracking of user context and thus user activities, potentially useful information is being gathered on many of the mobile systems implemented and discussed. Techniques and technology for moving this data to the desktop and reusing it are currently being investigated and future work hopes to examine thoroughly which of this information can be reused in such a way and if users feel comfortable sharing such information.

Another issue for future work is that of investigating whether seamful design can influence users' trust in aspects of the system. For example, Cosley et al. demonstrate that user acceptance of recommendations is highly dependent on the interface used to display recommendations [41], as do Dourish and Redmiles [54]. Given that systems from this thesis rely heavily on recommendations—that is, *Samara* and *Domino*—an investigation to determine how seamful design might influence these systems may be prudent. The application of seamful design to mobile recommendations may improve trust in the recommendations delivered. Just as Cosley et al. demonstrated that the interface might affect user acceptance, so might information about where recommendations and *Domino* modules have been generated, where they have been used previously, and who used them. Such an investigation seems increasingly necessary, as both mobile recommendations of this nature and the high level of adaptation *Domino* enables are novel in the mobile field. As such, it is important to determine methods to allow appropriate levels of trust to build up, in order to facilitate a faster uptake of systems and a spread of the infrastructure into application areas and communities where it may be useful—and inhibited where people reasonably feel that it is not.

9 REFERENCES

[1] Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., and Pinkerton, M., "Cyberguide: a mobile context-aware tour guide," *ACM Wireless Networks, Special issue: mobile computing and networking: selected papers from MobiCom '96,* vol. 3, pp. 421–433, 1997.

[2] Apple Computer Inc, "Bonjour Printing Specification." vol. 1.0.2 http://developer.apple.com/networking/bonjour/BonjourPrinting.pdf, 2005.

[3] Bahl, P. and Padmanabhan, V. N., "Radar: An in-building rf-based user location and tracking system," in *IEEE Infocom*, Tel-Aviv, Israel, 2000, pp. 775-784.

[4] Bangerman, E., "New law requires some businesses to secure their WiFi networks," http://arstechnica.com/news.ars/post/20060421-6647.html ed: ars technica, 2006.

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

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

[7] Bassoli, A., Moore, J., and Agamanolis, S., "tunA: Local Music Sharing with Handheld Wi-Fi Devices," in *The Fifth Wireless World Conference*, University of Surrey, UK, 2004.

[8] Baus, J., Kruger, A., and Wahlster, W., "A resource-adaptive mobile navigation system," in *7th International conference on Intelligent User Interfaces*, San Francisco, California, USA, 2002, pp. 15-22.

[9] Bell, M., Chalmers, M., Brown, B., MacColl, I., Hall, M., and Rudman, P., "Sharing photos and recommendations in the city streets," in *ECHISE workshop at Pervasive*, Munich, Germany, 2005.

[10] Bell, M., Chalmers, M., Barkhuus, L., Hall, M., Sherwood, S., Tennent, P., Brown, B., Rowland, D., Benford, S., Capra, and Hampshire, A., "Interweaving Mobile Games with Everyday Life," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Montreal, Canada, 2006, pp. 417-426.

[11] Bell, M., Hall, M., Chalmers, M., Gray, P., and Brown, B., "Domino: Exploring Mobile Collaborative Software Adaptation," in *Pervasive*, Dublin, Ireland, 2006, pp. 153-168.

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

[13] Benford, S., Seager, W., Flintham, M., Rowland, D., Stanton, D., Bowers, J., Tandavanitj, N., Adams, M., Row-Farr, J., Oldroyd, A., and Sutton, J., "The Error of Our Ways: Lessons from Using Self-reported Position in a Location-Based Game," in *Ubicomp*, Nottingham, UK, 2004, pp. 70-87.

[14] Berg, S., Taylor, A. S., and Harper, R., "Mobile Phones for the Next Generation: Device Designs for Teenagers," in *Human factors in computing systems*, Fort Lauderdale, Florida, USA, 2003, pp. 267-296.

[15] Birsan, D., "On plug-ins and extensible architectures," *Queue,* vol. 3, pp. 40–46, 2005.

[16] Borriello, G., Liu, A., Offer, T., Palistrant, C., and Sharp, R., "WALRUS: wireless acoustic location with room-level resolution using ultrasound," Seattle, Washington, 2005, pp. 191–203.

[17] Boutin, P., "Feds Label Wi-Fi a Terrorist Tool," in *Wired News*: http://www.wired.com/news/wireless/0,1382,56742,00.html, 2002.

[18] Brooks, R. and Iyengar, S., *Multi-sensor fusion: fundamentals and applications with software*: Prentice-Hall, Inc. USA, 1998.

[19] Brown, B. and Chalmers, M., "Tourism and mobile technology," in *CSCW*, Helsinki, Finland, 2003, pp. 335-355.

[20] Brown, B., MacColl, I., Chalmers, M., Galani, A., Randell, C., and Steed, A., "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.

[21] Brown, B., Chalmers, M., Bell, M., MacColl, I., Hall, M., and Rudman, P., "Sharing the square: collaborative leisure in the city streets," in *ECSCW*, Paris, France, 2005, pp. 427-429.

[22] Brunnberg, L., "The Road Rager - Making Use of Traffic Encounters in a Mobile Multiplayer Game," in *Mobile Multiplayer Games*, Maryland, USA, 2004, pp. 33-40.

[23] Brunnberg, L. and Juhlin, O., "Keep Your Eyes on the Road and Your Finger on the Trigger - Designing for Mixed Focus of Attention in a Mobile Game for Brief Encounters," in *Pervasive*, Dublin, Ireland, 2006, pp. 169-186.

[24] Bunt, A., Conati, C., and McGrenere, J., "What role can adaptive support play in an adaptable system?," in *Intelligent User Interfaces*, Funchal, Madeira, Portugal, 2004, pp. 117-124.

[25] Burrell, J., Gay, G. K., Kubo, K., and Farina, N., "Context-Aware Computing: A Test Case," in *UbiComp*, Gothenburg, Sweden, 2002, pp. 1-15.

[26] Capra, M., Radenkovic, M., Benford, S., Oppermann, L., Drozd, A., and Flintham, M., "The multimedia challenges raised by pervasive games," in *Proceedings of the 13th annual ACM international conference on Multimedia* Hilton, Singapore: ACM Press, 2005.

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

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

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

[30] Chalmers, M., "A Historical View of Context," *CSCW*, vol. 13, pp. 223-247, 2004.

[31] Chalmers, M. and Galani, A., "Seamful Interweaving: Heterogeneity in the Theory and Design of Interactive Systems," in *ACM Designing Interactive Systems*, 2004, pp. 243-252.

[32] Chang, M. and Goodman, E., "FIASCO: game interface for location-based play," in *ACM Designing Interactive Systems*, Cambridge, MA, USA, 2004, pp. 329-332.

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

[34] Chen, M. Y., Sohn, T., Chmelev, D., Haehnel, D., Hightower, J., Hughes, J., LaMarca, A., Potter, F., Smith, I., and Varshavsky, A., "Practical Metropolitan-scale Positioning for GSM Phones " in *UbiComp*, California, USA, 2006, pp. 225-242.

[35] Cheok, A. D., Goh, K. H., Liu, W., Farbiz, F., Fong, S. W., Teo, S. L., Li, Y., and Yang, X., "Human Pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing," *Personal and Ubiquitous Computing*, vol. 8, pp. 71–81, 2004.

[36] Cheverst, K., Davies, N., Mitchell, K., and Friday, A., "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.

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

[38] Cheverst, K., Smith, G., Mitchell, K., and Davies, N., "Exploiting Context to Support Social Awareness and Social Navigation," *ACM SIGGROUP Bulletin*, vol. 21, pp. 43-48, 2000.

[39] Cohen, D., Herscovici, M., Petruschka, Y., Maarek, Y., and Soffer, A., "Personalized pocket directories for mobile devices," in *World Wide Web*, Honolulu, Hawaii, USA, 2002, pp. 627–638.

[40] Correia, N., Alves, L., Correia, H., Romero, L., Morgado, C., Soares, L., Cunha, J., Romao, T., and Dias, A., "InStory: A System for Mobile Information Access, Storytelling and Gaming Activities in Physical Spaces," in *Advances in Computer Entertainment Technology*, Universidade Politècnica de Valencia (UPV), Spain, 2005, pp. 102-109.

[41] Cosley, D., Lam, S. K., Albert, I., Konstan, J. A., and Riedl, J., "Is Seeing Believing? How Recommender Interfaces Affect Users' Opinions," in *Human Factors in Computing Systems*, Fort Lauderdale, Florida, USA, 2003, pp. 585-592.

[42] Counts, S. and Fellheimer, E., "Supporting social presence through lightweight photo sharing on and off the desktop," in *Human Factors in Computing Systems*, Vienna, Austria, 2004, pp. 599-606.

[43] Crabtree, A., Benford, S., Rodden, T., Greenhalgh, C., Flintham, M., Anastasi, R., Drozd, A., Adams, M., Row-Farr, J., Tandavanitj, N., and Steed, A., "Orchestrating a mixed reality game 'on the ground'," in *Human factors in computing systems*, Vienna, Austria, 2004, pp. 391–398.

[44] Crow, D. and Smith, B., "The role of built-in knowledge in adaptive interface systems," in *Intelligent User Interfaces*, Orlando, Florida, United States, 1993, pp. 97–104.

[45] Dashofy, E. M., van der Hoek, A., and Taylor, R. N., "Towards Architecture-based Self-Healing Systems," in *Proceedings of the first workshop on Self-healing systems*, Charleston, SC, USA, 2002, pp. 21-26.

[46] Davis, R. C., Landay, J. A., Chen, V., Huang, J., Lee, R. B., Li, F. C., Lin, J., III, C. B. M., Schleimer, B., Price, M. N., and Schilit, B. N., "NotePals: lightweight note sharing by the group, for the group," in *Human Factors in Computing Systems*, Pittsburgh, Pennsylvania, United States, 1999, pp. 338–345.

[47] Dellarocas, C., Klien, M., and Shrobe, H., "An Architecture for Constructing Self-Evolving Software Systems," in *ISA W3*, Orlando, Florida, USA, 1998, pp. 29-32.

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

[49] DePriest, D., "NMEA data," http://www.gpsinformation.org/dale/nmea.htm, 2001.

[50] Dey, A. K. and Abowd, G., "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.

[51] Dey, A. K. and Abowd, G., "Support for Adapting Applications and Interfaces to Context," in *Multiple Users Interfaces: Engineering and Application Frameworks*, Seffah, A., et al., Eds.: John Wiley and Sons, 2003.

[52] Dourish, P. and Bellotti, V., "Awareness and Coordination in Shared Work Spaces," in *ACM Conference on Computer Supported Cooperative Work*, Toronto, Canada, 1992, pp. 107-114.

[53] Dourish, P. and Bly, S., "Portholes: Supporting Awareness in a Distributed Work Group," in *SIGCHI conference on Human Factors in computing systems*, Monterey, California, United States, 1992, pp. 541-547.

[54] Dourish, P. and Redmiles, D., "An Approach to Usable Security Based on Event Monitoring and Visualization," in *New Security Paradigms Workshop*, Virginia Beach, VA, 2002, pp. 75-81.

[55] Dourish, P., "What we talk about when we talk about context," *Personal and Ubiquitous Computing*, vol. 8, pp. 19-30, 2003.

[56] Dourish, P. and Bell, G., "Yesterday's tomorrows: notes on ubiquitous computing's dominant vision," *Personal and Ubiquitous Computing*, vol. 11, pp. 133-143, January 2007 2006.

[57] Eagle, N. and Pentland, A., "Social Serendipity: Mobilizing Social Software," *IEEE Pervasive Computing*, vol. 4, pp. 28-34, 2005.

[58] Edjlali, G., Acharya, A., and Chaudhary, V., "History-based access control for mobile code," in *5th ACM conference on Computer and communications security*, San Francisco, California, United States, 1998, pp. 38–48.

[59] Ekahau, "http://www.ekahau.com/."

[60] Enge, P., Fan, R., Tiwari, A., Chou, A., Mann, W., Sahai, A., Stone, J., and Van Roy, B., "Improving GPS Coverage and Continuity: Indoors and Downtown," in *Institute of Navigation's GPS Conference*, Salt Lake City, Utah, USA, 2001, pp. 3067-3076.

[61] Equator, "I Like Frank," in *Equator IRC* Nottingham: http://www.equator.ac.uk/index.php/articles/727, 2004.

[62] Esbjörnsson, M., Juhlin, O., and Östergren, M., "Traffic encounters and Hocman: Associating motrocycle ethnography with design," *Personal and Ubiquitous Computing*, vol. Volume 8, No. 2, pp. 92-99, 2004.

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

[64] Flintham, M., Anastasi, R., Benford, S., Hemmings, T., Crabtree, A., Greenhalgh, C., Rodden, T., Tandavanitj, N., Adams, M., and Row-Farr, J., "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.

[65] French, J. C. and Hauver, D. B., "Flycasting: On the Fly Broadcasting," in *Joint DELOS-NSF International Workshop on Personalization and Recommender Systems in Digital Libraries*, Dublin, Ireland, 2001, pp. 89-93.

[66] Fukumoto, M. and Shinagawa, M., "CarpetLAN: A Novel Indoor Wireless(-like) Networking and Positioning System," *Lecture Notes in Computer Science*, vol. 3660, pp. 1-18, 2005.

[67] Gaver, W. W., Moran, T., MacLean, A., Lovstrand, L., Dourish, P., Carter, K., and Buxton, W., "Realizing a Video Environment: EuroPARC's RAVE System," in *Human factors in computing systems*, Monterey, CA USA, 1992, pp. 27-35.

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

[69] Gellersen, H. W., Schmidt, A., and Beigl, M., "Multi-sensor context-awareness in mobile devices and smart artifacts," *Mob. Netw. Appl.*, vol. 7, pp. 341--351, 2002.

[70] Georgiadis, I., Magee, J., and Kramer, J., "Self-Organising Software Architectures for Distributed Systems," in *WOSS '02*, Charleston, SC, USA, 2002, pp. 33-38.

[71] Glance, N., Snowdon, D., and Meunier, J.-L., "Pollen: using people as a communication medium," *Computer Networks*, vol. 35, pp. 429-442, 2001.

[72] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D., "Using Collaborative Filtering to Weave an Information Tapestry," *Communications of the ACM*, vol. 35, pp. 61-70, 1992.

[73] Goldenberg, J., Shavitt, Y., Shir, E., and Solomon, S., "Distributive immunization of networks against viruses using the 'honey-pot' architecture," *Nature Physics*, vol. 1, pp. 184-188, 2005.

[74] Goren-Bar, D. and Kuflik, T., "Don't miss-r --: recommending restaurants through an adaptive mobile system," in *9th international conference on Intelligent user interface*, Funchal, Madeira, Portugal, 2004, pp. 250-252.

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

[76] Grinter, R. E., Ducheneaut, N., Edwards, W. K., and Newman, M. W., "The work to make a home network work," in *Ninth European Conference on Computer-Supported Cooperative Work*, Paris, France, 2005, pp. 360-369.

[77] Griswold, W. G., Shanahan, P., Brown, S. W., and Boyer, R. T., "ActiveCampus - Experiments in Community-Oriented Ubiquitous Computing," *Computer*, vol. Vol 37, Issue 10, pp. 73-81, 2004.

[78] Hakansson, M., "Push!Music: Mobile Music Sharing with Media Agents," in *SIGCHI conference on Human Factors in computing systems*, Montreal, Canada, 2006, pp. 909-918.

[79] Hale, R. V., "Wi-Fi Liability: Potential Legal Risks in Accessing and Operating Wireless Internet," *Santa Clara Computer and High Technology Law Journal*, vol. 21, p. 543, 2005.

[80] Hallberg, J., Nilsson, M., and Synnes, K., "Positioning with Bluetooth," in *ICT*, 2003, pp. 954-958.

[81] Hallberg, J., Svensson, S., Östmark, A., Lindgren, P., Synnes, K., and Delsing, J., "Enriched Media-Experience of Sport Events," in *WMSCA*, Lake District National Park, United Kingdom, 2004, pp. 2-9.

[82] Handurukande, S. B., Kermarrec, A.-M., Fessant, F. L., and Massouli, L., "Exploiting semantic clustering in the eDonkey P2P network," Leuven, Belgium, 2004, p. 20.

[83] Hansen, F. A., Bouvin, N. O., Christensen, B. G., Gronbaek, K., Pedersen, T. B., and Gagach, J., "Integrating the Web and the World: Contextual Trails on the Move," in *Hypertext*, Santa Cruz, USA, 2004, pp. 98–107.

[84] Harter, A. and Hopper, A., "A Distributed Location System for the Active Office," *IEEE Network*, vol. 8, pp. 62-70, 1994.

[85] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P., "The anatomy of a context-aware application," in *MobiCom: 5th Annual ACM/IEEE international conference on Mobile computing and networking*, Seattle, Washington, United States, 1999, pp. 59–68.

[86] Hayes, A. and Wilson, D., "Peer-to-Peer Information Sharing in a Mobile Ad Hoc Environment," in *WMCSA*, Lake District National Park, UK, 2004, pp. 154-162.

[87] Henricksen, K. and Indulska, J., "Adapting the Web Interface: An Adaptive Web Browser," *Proc. of the 2nd Australasian User Interface Conference (AUIC'2001). Australian Computer Science Communications*, vol. 23, pp. 21-33, 2001.

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

[89] Huang, J. and Waldvogel, M., "The swisshouse: an inhabitable interface for connecting nations," in *Proceedings of the 2004 conference on Designing interactive systems*, Cambridge, MA, USA, 2004, pp. 195-204.

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

[91] Iachello, G., Smith, I., Consolvo, S., Abowd, G., D., Hughes, J., Howard, J., Potter, F., Scott, J., Sohn, T., Hightower, J., and LaMarca, A., "Control, Deception, and Communication: Evaluating the Deployment of a Location-Enhanced Messaging Service," *Lecture Notes in Computer Science*, vol. 3660, pp. 213-231, 2005.

[92] Janecek, A. and Hlavacs, H., "Programming interactive real-time games over WLAN for pocket PCs with J2ME and .NET CF," in *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, Hawthorne, NY, 2005, pp. 1–8.

[93] Jiang, C. and Steenkiste, P., "A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing," in *Ubicomp 2002, Lecture Notes in Computer Science*, Goteborg, Sweden, 2002, p. 246ff.

[94] *Kang, J. H.* and *Borriello, G.*, "Ubiquitous Computing Using Wireless Broadcast," in *WMCSA*, Lake District National Park, United Kingdom, 2004, pp. 72-81.

[95] Kerr, O. S., "Cybercrime's Scope: Interpeting 'Access' and 'Authorization' in Computer Misuse Statutes," *NYU Law Review*, vol. 78, pp. 1596-1668, November 2003.

[96] Khelil , A., Becker, C., Tian, J., and Rothermel, K., "An epidemic model for information diffusion in MANETs," in *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems* Atlanta, Georgia, USA: ACM Press </pre> </body> </html>, 2002.

[97] Kim, M., Fielding, J. J., and Kotz, D., "Risks of using AP locations discovered through war driving," in *Pervasive*, Dublin, Ireland, 2006, pp. 67-82.

[98] Knyrim, R. and Podoscheck, C., "Detektiv beutete Handy-Netzdaten," in *Rechtspanorama, 28th November*, 2005.

[99] Kolodziej, K., "Advances in GPS: Navizon," in *IndoorLBS*: http://www.lbszone.com/content/view/1171/1/, 2006.

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

[101] Kruger, A., Butz, A., Muller, C., Stahl, C., Wasinger, R., Steinberg, K.-E., and Dirschl, A., "The connected user interface: realizing a personal situated navigation service," in *9th international conference on Intelligent user interface*, Funchal, Madeira, Portugal, 2004, pp. 161-168.

[102] Kusunoki, F., Yamaguti, T., Nishimura, T., Yatani, K., and Sugimoto, M., "Interactive and Enjoyable Interface in Museum," in *ACE*, Valencia, 2005, pp. 1-8.

[103] Laerhoven, K. V., Schmidt, A., and Gellersen, H.-W., "Multi-Sensor Context Aware Clothing," in *Sixth International Symposium on Wearable Computers*, Seattle, USA, 2002, pp. 49-56.

[104] LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tarbert, J., Powledge, P., Borriello, G., and Schilit, B., "Place Lab: Device Positioning Using Radio Beacons in the Wild," in *Pervasive*, Munich, Germany, 2005.

[105] LaMarca, A., Hightower, J., Smith, I., and Consolvo, S., "Self-Mapping in 802.11 Location Systems," *Lecture Notes in Computer Science*, vol. 3660, pp. 87-104, 2005.

[106] Lammertsma, P. F., "Satellite Navigation," in *Institute of Information and Computing Sciences, Utrecht University*: http://paul.luminos.nl/documents/show_document.php?d=7, 2005.

[107] Lindemann, C. and Waldhorst, O., P., "Exploiting epidemic data dissemination for consistent lookup operations in mobile applications," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, pp. 44-56, 2004.

[108] Lindemann, C. and Waldhorst, O., P., "Modeling epidemic information dissemination on mobile devices with finite buffers," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, Banff, Alberta, Canada, 2005, pp. 121-132.

[109] Mackay, W. E., "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.

[110] MacLean, A., Carter, K., Lovstrand, L., and Moran, T., "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 Press </pre> </body> </html>, 1990.

[111] Madhavapeddy, A. and Tse, A., "A Study of Bluetooth Propagation Using Accurate Indoor Location Mapping," *Lecture Notes in Computer Science*, vol. 3660, pp. 105-122, 2005.

[112] Mainwaring, S., Chang, M. F., and Anderson, K., "Infrastructures and Their Discontents: Implications for Ubicomp," in *Ubicomp*, Nottingham, UK, 2004, pp. 418-432.

[113] Mantoro, T. and Johnson, C., "Location History in a Low-cost Context Awareness Environment," in *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, Adelaide, Australia, 2003, pp. 153-158.

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

[115] Marmasse, N., Schmandt, C., and Spectre, D., "WatchMe: communication and awareness between members of a closely-knit group," in *Ubicomp 2004*, Nottingham, England, 2004, pp. 214-231.

[116] Messeter, J., Brandt, E., Halse, J., and Johansson, M., "Contextualizing mobile IT," in *Designing interactive systems*, Cambridge, MA, USA, 2004, pp. 27-36.

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

[118] Mitchell, K., McCaffery, D., Metaxas, G., and Finney, J., "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.

[119] Muller, M. J., Geyer, W., Brownholtz, B., Wilcox, E., and Millen, D. R., "One-hundred days in an activity-centric collaboration environment based on shared objects," in *Human factors in computing systems*, Vienna, Austria, 2004, pp. 375-382.

[120] Musolesi, M., Mascolo, C., and Hailes, S., "EMMA: Epidemic Messaging Middleware for Ad hoc networks," *Personal Ubiquitous Computing*, vol. 10, pp. 28-36, 2005.

[121] Newcomb, E., Pashley, T., and Stasko, J., "Mobile computing in the retail arena," in *Human factors in computing systems*, Ft. Lauderdale, Florida, USA, 2003, pp. 337-344.

[122] Newman, M. W., Sedivry, J. Z., Edwards, W. K., Smith, T. F., Marcelo, K., Neuwirth, C. M., Hong, J. I., and Izadi, S., "Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments," in *Designing Interactive Systems*, London, UK, 2002, pp. 147-156.

[123] Newman, M. W., Voida, A., Grinter, R. E., Ducheneaut, N., and Edwards, K., "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.

[124] Ni, L. M., Liu, Y., Lau, Y. C., and Patil, A. P., "LANDMARC: indoor location sensing using active RFID," *Wireless Networks*, vol. 10, pp. 701–710, 2004.

[125] NMEA, "NMEA 0183 Standard," National Marine and Electornics Association, Publications and Standards, 2002.

[126] O'Shea, T., Lamming, M., Chalmers, M., Graube, N., Wellner, P., and Wiginton, G., "Expectations and Perceptions of Ubiquitous Computing: Experiments with BirdDog, a Prototype Person Locator," in *BCS/IEE Conference on Information Technology and People (ITaP)*, 1991.

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

[128] Östergren, M. and Juhlin, O., "Sound Pryer: truly mobile joint music listening," in *International Conference on Entertainment Computing*, Eindhoven, Amsterdam, 2004.

[129] Otsason, V., Varshavsky, A., LaMarca, A., and de, L., Eyal, "Accurate GSM Indoor Localization," in *Ubicomp* Tokyo, Japan: Springer, 2005, pp. 141-158.

[130] Özkasap, Ö., Genc, Z., and Atsan, E., "Epidemic-based approaches for reliable multicast in mobile ad hoc networks," *SIGOPS Oper. Syst. Rev.*, vol. 40, pp. 73-79, 2006.

[131] Papadopouli, M. and Schulzrinne , H., "Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking \& computing* Long Beach, CA, USA: ACM Press </pre> </body> </html>, 2001.

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

[133] Patterson, D. J., Liao, L., Gajos, K., Collier, M., Livic, N., Olson, K., Wang, S., Fox, D., and Kautz, H., "Opportunity Knocks: a System to Provide Cognitive Assistance with Transportation Services," in *Ubicomp* Nottingham, UK, 2004.

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

[135] Persson, P., Espinoza, F., and Cacciatore, E., "GeoNotes: social enhancement of physical space," in *Human factors in computing systems*. vol. CHI '01 extended abstracts on Human factors in computing systems Seattle, Washington: ACM Press, 2001, pp. 43–44.

[136] Persson, P., Blom, J., and Jung, Y., "DigiDress: A Field Trial of an Expressive Social Proximity Application," in *Ubicomp* Tokyo, Japan: ACM Press, 2005, pp. 195-212.

[137] Persson, P. and Jung, Y., "Nokia Sensor: From Research to Product," in *2005 conference on Designing for User eXperience* San Francisco, CA: AIGA: American Institute of Graphic Arts, 2005, p. 53ff.

[138] Place Lab, "www.placelab.org," 2006.

[139] Priyantha, N. B., Chakraborty, A., and Balakrishnan, H., "The Cricket location-support system," in *6th annual international conference on Mobile computing and networking* Boston, Massachusetts, United States: ACM Press New York, NY, USA, 2000, pp. 32–43.

[140] Raento, M., Oulasvirta, A., Petit, R., and Toivonen, H., "ContextPhone: A Prototyping Platform," *Pervasive Computing*, pp. 51-59, 2005.

[141] Rakkolaien, I., Kupila, H., Majahalme, T., and Salmenpera, H., "Improving GPS Accuracy for a Mobile 3D City Info," in *Multi-Dimensional Mobile Communications*, Pori, Finland, 2001.

[142] Randell, C. and Muller, H., "The Well Mannered Wearable Computer," *Personal and Ubiquitous Computing 2002*, vol. 6, pp. 31-36, 2002.

[143] Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., and Riedl, J., "Getting to Know You: Learning New User Preferences in Recommender Systems," in *Intelligent User Interfaces* San Francisco, California: ACM Press, 2002, pp. 127-134.

[144] Resnick, P. and Varian, H. R., "Recommender systems," *Communications of the ACM*, vol. 40, pp. 56–58, 1997.

[145] Ritter, H., Voigt, T., Tian, M., and Schiller, J., "Experiences using a dual wireless technology infrastructure to support ad-hoc multiplayer games," in *NetGames* Redwood City, California, 2003, pp. 101-105.

[146] Rudström, Å., Svensson, M., Cöster, R., and Höök, K., "MobiTip: Using Bluetooth as a Mediator of Social Context," in *Ubicomp*. vol. Adjunct Proceedings Nottingham, UK, 2004.

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

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

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

[150] Schwaighofer, A., Grigoras, M., Tresp, V., and Hoffmann, C., "GPPS: A Gaussian process positioning system for cellular networks," in *Neural Information Processing Systems*, Vancouver, Canada, 2004.

[151] Sellen, A., Eardley, R., Izadi, S., and Harper, R., "The Whereabouts Clock: early testing of a situated awareness device.," in *Conference on Human Factors and Computing Systems*, Montreal, Canada, 2006, pp. 1307-1312.

[152] Shardanand, U. and Maes, P., "Social information filtering: algorithms for automating "word of mouth"," in *SIGCHI conference on Human factors in computing systems* Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 210-217.

[153] *Simler, K. D., Czerwinski, S. E.*, and *Joseph, A. D.*, "Analysis of Wide Area User Mobility Patterns," in *WMCSA* Lake District National Park, UK: IEEE, 2004, pp. 30-40.

[154] Skov, B. and Høegh, T., "Supporting information access in a hospital ward by a context-aware mobile electronic patient record," *Personal Ubiquitous Computing*, vol. 10, pp. 205–214, 2006.

[155] Small, J., Smailagic, A., and Siewiorek, D. P., "Determining User Location For Context Aware computing Through the Use of a Wireless LAN Infrastructure," 2000.

[156] Smith, R. B., Hixon, R., and Horan, B., "Supporting flexible roles in a shared space," in *CSCW* Seatlle, USA: ACM Press, 1998, pp. 197-206.

[157] Sood, S., Hammond, K. J., and Birnbaumb, L., "Low-fidelity location based information systems," in *9th international conference on Intelligent user interface* Funchal, Madeira, Portugal: ACM Press, 2004, pp. 325--327.

[158] Spratt, M., "An overview of positioning by diffusion," *Wireless Networks*, vol. 9, pp. 565--574, 2003.

[159] Strohbach, M., Gellersen, H. W., Kortuem, G., and Kray, C., "Cooperative Artifacts: Assessing Real World Situations with Embedded Technology," in *Ubicomp*, Nottingham, UK, 2004.

[160] Takeuchi, Y. and Sugimoto, M., "An Outdoor Recommendations System based on User Location History," in *1st International Workshop on Personalized Context Modeling and Management for UbiComp Applications (UbiPCMM 2005)*, 2005, pp. 91-100.

[161] Terrenghi, L. and Zimmermann, A., "Tailored audio augmented environments for museums," in *9th international conference on Intelligent user interface*, Funchal, Madeira, Portugal, 2004, pp. 334--336.

[162] Terry, M., Mynatt, E. D., Ryall, K., and Leigh, D., "Towards Design Guidelines for Portable Digital Proxies: A Case Study with Social Net and Social Proximity," 2001.

[163] Terry, M., Mynatt, E. D., Ryall, K., and Leigh, D., "Social Net: Using Patterns of Physical Proximity Over Time to Infer Shared Interests," in *Human factors in computing systems*, Minneapolis, Minnesota, USA, 2003, pp. 816-817.

[164] Trevisani, E. and Vitaletii, A., "Cell-ID Location Technique, Limits and Benefits: An Experimental Study," in *WMCSA*, Lake District National Park, UK, 2004, pp. 51-60.

[165] Triantafillou, P., Ntarmos, N., Nikoletseas, S., and Spirakis, P., "NanoPeer Networks and P2P Worlds," in *3rd IEEE International Conference on Peer-to-Peer Computing*, 2003, p. 40ff.

[166] Tveit, A., "Peer-to-peer based recommendations for mobile commerce," in *1st international workshop on Mobile commerce*, Rome, Italy, 2001, pp. 26--29.

[167] Ujjin, S., "An Adaptive Lifestyle Recommender System Using a Genetic Algorithm," 2001.

[168] Vogels, W., Renesse, R. v., and Birman, K., "The power of epidemics: robust communication for large-scale distributed systems," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 131-135, 2003.

[169] Want, R., Hopper, A., Falcao, V., and Gibbons, J., "The Active Badge Location System," *ACM Trans. Inf. Syst.*, pp. 91-102, 1992.

[170] Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M., and Light, J., "The Personal Server: Changing the Way We Think about Ubiquitous Computing," *Lecture Notes in Computing Science*, vol. 2498, pp. 194-209, 2002.

[171] Ward, A., Jones, A., and Hopper, A., "A New Location Technique for the Active Office," *IEEE Personal Communications*, vol. Vol. 4 No. 5, pp. 42-47, 1997.

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

[173] Wiberg, M., "Mobile Peer-to-peer Entertainment: Putting FolkMusic Back on the Streets Again When Peer-to-peer Goes Mobile," in *Hawaii International Conference on System Sciences*, Hawaii, USA, 2004, pp. 288-297.

[174] Zeimpekis, V., Giaglis, G. M., and Lekakos, G., "A taxonomy of indoor and outdoor positioning techniques for mobile location services," *SIGecom Exch.*, vol. 3, pp. 19--27, 2003.

10 APPENDIX

10.1 Appendix A – Bluetooth and 802.11 peer discovery and transfer tests

These tests compare Bluetooth, 802.11 ad hoc and 802.11 infrastructure peer discovery times and transfer rates.

For all Bluetooth tests iMate SP5 phones were used. The primary reason for selecting this phone was that, at the time, it was one of the few Windows Mobile devices available that utilised the Microsoft Bluetooth Stack. This protocol stack is slightly easier to access than older Windows Mobile devices, which used a WIDCOM Bluetooth stack. Currently (August 2006), there are now many more Windows Mobile 5 devices on the market and nearly all devices sold with this operating system use the Microsoft Bluetooth Stack. Therefore, if the tests were to be run again in the future the number of devices that could potentially be trialled, with the existing trial code, is far higher.

For the 802.11 tests iPAQ 4150s were used—the same models that were used in trials of *Treasure*, *Feeding Yoshi* and *Castles*.

The author is well aware that there are many factors that may influence the results from any trial relying on wireless technology—be it Bluetooth, 802.11, GSM or other forms of wireless communication. Hardware factors such as the antennae used, the shape of the antennae, and the form of the device, which may shield antennae, all influence the strength of the wireless signals. Furthermore, external factors such as the weather—wind, air pressure and humidity—affect the signal as it travels between devices. The affects of both hardware and weather are likely to be negligible. The author's own experimentation of running the trial with over 15 models of PDA suggests that the results presented here for 802.11 and Bluetooth are indeed typical of the majority of mobile devices. Experiments examining Navizon's accuracy, conducted by Graham McKinley, show no significant correlation with 802.11 signal strength and weather.

Despite this, the author is wary, believing that the hardware and antennae used, and weather during trials, are bound to influence both 802.11 and Bluetooth performance at some level. The influence is negligible for the research within this thesis—there is a massive difference in performance between Bluetooth and 802.11 and these factors could not affect the results to the extent that this gap could be significantly reduced and the outcomes reversed.

10.1.1 Peer Discovery

The following tables show *successful* peer discovery times at 1, 5, 10, 25, 50 and 100 metre distances. Distances over 10m are not included for Bluetooth, as although peer discovery was attempted at 25m (an indeed even at 15m) there were no successful discoveries whatsoever. Similarly, for the 802.11

infrastructure test no recordings for 100m are given as, despite repeated attempts, there were none whatsoever at this distance.

In addition the results presented in the tables, notes on the success rate of discovery were recorded. At each distance 20 discovery attempts were made and if the peer devices had failed to discover one another after 30 seconds the discovery attempt was recorded as a failure. After 20 attempts, the ratio of successful discovery attempts were recorded. In instances were discovery had not been 100% successful, additional discovery attempts were then made until a total of 20 times had been recorded, which are presented in the tables.

Bluetooth discovery was successful 80% at 1m, 100% at 5m and 55% at 10m. At distances over 10m Bluetooth had 0% success in discovering peers. 802.11 was 100% successful at all distances in both ad hoc and infrastructure mode, with the exception of 100m in infrastructure mode, where it had 0% success.

Bluetooth

Distance (m)	1	5	10
Time (ms)	4043	4423	13143
	4039	4018	10443
	4127	3993	19358
	4204	4078	10619
	4201	4007	12543
	5004	4287	10943
	4062	4009	10746
	4236	4208	21899
	4445	4495	10750
	4058	4393	10312
	4047	4073	21443
	4361	4594	21879
	4912	4221	10688
	4275	4569	19522
	4210	4048	19589
	4019	4342	19472
	4114	4261	12969
	4159	4480	10409
	4080	4439	21660
	4538	4008	13245
Average time (ms)	**4256.7**	**4247.3**	**15081.6**

802.11 ad hoc

Distance (m)	1	5	10	25	50	100
Time (ms)	8	4	18	16	11	6

	18	3	6	18	22	9
	11	16	2	11	4	14
	6	4	9	17	19	9
	11	9	3	11	15	27
	12	18	11	8	8	24
	7	6	1	2	13	18
	14	17	18	9	4	16
	4	5	6	2	9	4
	11	13	11	16	17	31
	10	6	11	12	12	18
	15	11	7	15	19	9
	11	6	7	10	1	3
	10	11	10	5	8	17
	15	16	19	7	15	19
	3	4	3	7	10	11
	6	3	12	12	8	12
	5	6	7	2	7	6
	3	6	10	12	7	20
	18	16	10	9	15	24
Average time (ms)	**9.9**	**9**	**9.05**	**10.05**	**11.2**	**14.85**

802.11 Infrastructure

Distance (m)	**1**	**5**	**10**	**25**	**50**	**100**
Time (ms)	43	41	22	35	39	
	62	42	77	42	43	
	24	46	46	34	60	
	94	70	70	72	50	
	43	31	21	23	96	
	24	86	73	45	54	
	34	88	77	72	695	
	35	23	80	41	20	
	51	71	53	81	52	
	44	40	3	53	30	
	7	24	71	342	33	
	79	54	73	39	53	
	61	4	86	62	663	
	52	89	20	93	241	
	52	45	19	25	55	
	91	17	23	34	378	
	49	2	34	73	526	
	77	34	37	45	75	
	43	18	43	53	55	
	78	77	95	72	139	
Average time (ms)	**52.15**	**45.1**	**51.15**	**66.8**	**167.85**	**N/A**

10.1.2 Data transfer

The tables in this section show the times taken to transfer 1MB of data between peer devices using Bluetooth, 802.11 ad hoc and 802.11 infrastructure, *after* they have discovered one another using the peer discovery methods detailed in the thesis. As with the discovery time trials, it is believed that whilst hardware and weather factors may have influenced the results, the affects are negligible to this research.

As the peer discovery trials showed, Bluetooth discovery over 10m, and 802.11 ifnrastructure discovery at 100m was not successful. Peers could not detect one another at these distances nor could they transfer data, and so results at these distances could not be recorded.

Bluetooth

Distance (m)	1	5	10
Time (ms)	22968	19368	65566
	17546	19257	62098
	18767	20714	65266
	18006	22109	74949
	17903	18557	61649
	17706	19943	65873
	22144	19335	76786
	18093	23739	61145
	19020	20132	62908
	18481	21438	74001
Average time (ms)	**19063.4**	**20459.2**	**67024.1**

802.11 ad hoc

Distance (m)	1	5	10	25	50	100
Time (ms)	2551	2568	2594	2982	2686	3185
	2615	2621	2628	2691	2786	6586
	2598	2624	2705	2680	2667	2717
	2589	2610	2599	2736	2615	4041
	2589	2558	2646	2724	2740	5343
	2611	2669	2598	2794	2812	3729
	2573	2624	2594	2734	2843	6018
	2610	2604	2620	2916	2997	5461
	2550	2555	2726	2763	2649	5091
	2617	2581	2597	2726	2856	3338
	2584	2659	2703	2714	2677	4285
Average time (ms)	**2589.7**	**2606.6**	**2637.3**	**2769.1**	**2757.1**	**4526.7**

802.11 infrastructure

Distance (m)	1	5	10	25	50	100
Time (ms)	4306	4256	4480	4382	4372	
	4306	4378	4270	4244	4325	
	4276	4116	4202	4338	4639	
	4172	4256	4362	4382	4314	
	4336	4208	4442	4186	4826	
	4192	4336	4292	4442	4501	
	4236	4186	4220	4306	4882	
	4264	4290	4356	4160	4733	
	4344	4260	4270	4296	4674	
	4180	4138	4304	4278	4366	
	4250	4148	4463	4329	4649	
Average time (ms)	**4260.2**	**4233.8**	**4332.8**	**4303.9**	**4571.0**	**N/A**

10.2 Appendix B – Access point positioning in Navizon

The aim of these tests was to compare the technique Navizon uses to originally locate new beacons to other possible techniques.

The results from each test are shown in two tables. The first table shows the samples of GPS coordinates with the signal strength of the access point. The second shows the actual location of the access point and the locations found by Navizon, the first sample method and the strongest reading method – as well as the error of these methods from the true location.

Access Point 1

Latitude	Longitude	Signal Strength
55.8745667	-4.2918583	64
55.8744967	-4.2919	68
55.8744967	-4.2919	69
55.8744267	-4.2919433	78
55.8743533	-4.2920033	77
55.8742833	-4.2920617	64
55.8742833	-4.2920617	62
55.8742833	-4.2920617	59
55.87421	-4.2921	59
55.874145	-4.2921217	59
55.874095	-4.2921433	59
55.8747283	-4.2921883	59

	Latitude	Longitude	Error
Actual	55.87434	-4.29207	-
Navizon	55.87436903	-4.292022036	0.0000560633
First	55.8745667	-4.2918583	0.000310177
Strongest	55.8744267	-4.2919433	0.000153525
Average	55.87436403	-4.292028608	0.0000478589

Access Point 2

Latitude	Longitude	Signal Strength
55.8742833	-4.2920617	66
55.8742833	-4.2920617	68
55.8742833	-4.2920617	63
55.87421	-4.2921	63
55.874145	-4.2921217	63
55.874095	-4.2921433	63
55.874095	-4.2921433	63
55.874095	-4.2921433	61
55.8740617	-4.2921917	61
55.8740583	-4.292265	64
55.8740583	-4.292265	64

	Latitude	Longitude	Error
Actual	55.87412	-4.29225	-
Navizon	55.87415331	-4.292140963	0.0001140130
First	55.8742833	-4.2920617	0.000249246
Strongest	55.8742833	-4.2920617	0.000249246
Average	55.87415165	-4.292141673	0.000112857

Access Point 3

Latitude	Longitude	Signal Strength
55.8746467	-4.2917967	59
55.8744967	-4.2919	75
55.8740583	-4.292265	58
55.8740967	-4.2923533	58

	Latitude	Longitude	Error
Actual	55.87438	-4.29221	-
Navizon	55.87433759	-4.292065467	0.0001506265
First	55.8746467	-4.2917967	0.00049188
Strongest	55.8744967	-4.2919	0.000331238
Average	55.8743246	-4.29207875	0.000142463

Access Point 4

Latitude	Longitude	Signal Strength
55.8749983	-4.2915333	64
55.874825	-4.291655	64
55.874825	-4.291655	64
55.8747317	-4.29172	64
55.8747317	-4.29172	62
55.8746467	-4.2917967	68
55.8745667	-4.2918583	66
55.8744967	-4.2919	62
55.8744967	-4.2919	62
55.8746717	-4.2922367	61
55.8746717	-4.2922367	60
55.8747283	-4.2921883	62
55.8747283	-4.2921883	62

	Latitude	Longitude	Error
Actual	55.87474	-4.29192	-
Navizon	55.87470187	-4.291886851	0.0000505216
First	55.8749983	-4.2915333	0.000465033
Strongest	55.8746467	-4.2917967	0.000154621
Average	55.87470142	-4.291891408	0.0000480177

Access Point 5

Latitude	Longitude	Signal Strength
55.8721283	-4.2857917	59
55.872125	-4.2855533	66

55.872125	-4.2855533	66
55.87217	-4.2851617	62
55.87217	-4.2851617	65
55.87217	-4.2851617	64
55.8721733	-4.2851417	64
55.872175	-4.2851367	64
55.872175	-4.2851333	64
55.872175	-4.2851333	62
55.8721767	-4.2851317	61
55.8721767	-4.28513	60
55.872175	-4.28513	60
55.872175	-4.28513	59
55.872175	-4.2851283	60
55.8721767	-4.2851233	65
55.8721767	-4.2851233	65
55.8721767	-4.2851233	65
55.872175	-4.2851217	60
55.872175	-4.2851217	60
55.8721767	-4.2851033	65
55.8721767	-4.2851033	65
55.8721917	-4.2849967	63
55.872195	-4.2849567	61
55.8721867	-4.2848983	61
55.8721867	-4.2848983	61
55.8721733	-4.2848533	59
55.87216	-4.2848217	59
55.87216	-4.2848217	59
55.8721367	-4.284795	58
55.8720967	-4.2847833	58
55.8720533	-4.2848517	58

	Latitude	Longitude	Error
Actual	55.87234	-4.28526	-
Ours	55.87216405	-4.285100258	0.0002376442
First	55.8721283	-4.2857917	0.000572295
Strongest	55.872125	-4.2855533	0.000363662
Average	55.87216371	-4.285096094	0.000240717

Access Point 6

Latitude	Longitude	Signal Strength
55.8965833	-4.4410883	62
55.8966533	-4.4413617	62
55.8966533	-4.4413617	57
55.896705	-4.4416533	57
55.896705	-4.4416533	57
55.8962817	-4.4422617	58
55.8962817	-4.4422617	58
55.896265	-4.442045	58
55.8962983	-4.4419817	58

55.8962983	-4.4419817	59
55.8963417	-4.44196	60
55.8963733	-4.4419333	62
55.8963917	-4.441915	62
55.8963917	-4.441915	59
55.8963917	-4.441915	62
55.89643	-4.441925	60
55.8964317	-4.4419283	60
55.8964317	-4.4419283	60
55.8964317	-4.4419283	60
55.89643	-4.4419283	60
55.8964283	-4.4419267	60
55.8964283	-4.4419267	60
55.8964233	-4.441925	64
55.8964233	-4.441925	64
55.8964133	-4.4419267	62
55.8964133	-4.4419267	62
55.8964033	-4.4419233	61
55.8964033	-4.4419233	61
55.896395	-4.441915	61
55.8963917	-4.4419067	60
55.8963883	-4.441895	59
55.8963883	-4.441895	59
55.8963867	-4.4418767	61
55.8963833	-4.4418583	61
55.8963833	-4.4418583	60
55.8963833	-4.4418517	60
55.8963833	-4.4418517	59
55.8963833	-4.4418517	55
55.8963833	-4.4418517	55
55.8963833	-4.4418517	62
55.8963833	-4.4418517	60
55.8963817	-4.4418533	59
55.8963817	-4.4418533	59
55.8963817	-4.4418533	59
55.8963817	-4.4418533	60
55.8963817	-4.4418533	61
55.8963817	-4.441855	61
55.8963817	-4.441855	61
55.8963817	-4.441855	61
55.8963817	-4.441855	59
55.8963817	-4.441855	60
55.8963817	-4.441855	60
55.8963817	-4.4418567	60
55.89638	-4.4418567	60
55.89638	-4.4418567	59
55.89638	-4.4418567	59
55.89638	-4.4418567	60
55.89638	-4.4418583	60
55.89638	-4.4418583	61

55.8963783	-4.4418583	60
55.896375	-4.4418983	58
55.8963733	-4.441935	58
55.8963733	-4.441935	59
55.8963733	-4.441935	59
55.8963683	-4.441975	55
55.8963617	-4.4420517	59
55.8963617	-4.4420517	56
55.8963617	-4.4420517	56
55.8963667	-4.4419733	58
55.8963633	-4.4419733	57
55.8963633	-4.4419733	57
55.896365	-4.44197	57
55.8963733	-4.4419783	62
55.8963833	-4.4419883	63
55.8963833	-4.4419883	63
55.8963933	-4.4419917	61
55.8963933	-4.4419917	61
55.8964033	-4.441985	61
55.896415	-4.4419817	61
55.896415	-4.4419817	64
55.896415	-4.4419817	64
55.896425	-4.4419767	64
55.8964383	-4.4419617	65
55.8964483	-4.4419517	72
55.8964483	-4.4419517	72
55.8964583	-4.4419417	69
55.89647	-4.4419317	75
55.89648	-4.4419233	83
55.89648	-4.4419233	83
55.8965017	-4.44195	81
55.8965017	-4.44195	84
55.89649	-4.4419583	84
55.89649	-4.4419583	81
55.89649	-4.4419583	81
55.8964883	-4.4419667	82
55.8964883	-4.4419667	82
55.8964833	-4.4419867	84
55.8964833	-4.4419867	85
55.8964833	-4.4419867	80
55.89648	-4.4420067	80
55.8964767	-4.44202	84
55.8964767	-4.44202	84
55.8964717	-4.4420433	86
55.8964717	-4.4420433	83
55.89647	-4.442055	89
55.89647	-4.442055	89
55.8964633	-4.4420717	93
55.89646	-4.4420867	93
55.89646	-4.4420867	98

55.8964567	-4.4421	98
55.8964567	-4.4421	92
55.896455	-4.442115	83
55.8964517	-4.4421283	83
55.8964483	-4.4421433	82
55.8964483	-4.4421433	87
55.8964383	-4.4421683	88
55.8964333	-4.442185	88
55.8964283	-4.4422017	83
55.8964283	-4.4422017	86
55.896425	-4.4422167	89
55.8964217	-4.4422333	89

	Latitude	Longitude	Error
Actual	55.89658	-4.44194	-
Navizon	55.89642558	-4.441956244	0.000155269
First	55.8965833	-4.4410883	0.000851706
Strongest	55.89646	-4.4420867	0.000189528
Average	55.89642052	-4.441943364	0.000159516

10.2.1 Conclusion

For every access point the weighted centroid technique employed by Navizon outperformed both the first sample and strongest sample methods. The table below shows the ratio of the weighted centroid technique compared to both the first and strongest sample methods for each of the access points.

	First	Strongest
AP 1	5.532622	2.738414442
AP 2	2.186122539	2.186122539
AP 3	3.2655591	2.199070866
AP 4	9.204643291	3.060502413
AP 5	2.408202149	1.530278606
AP 6	5.485357871	1.220642828
Average	**4.680417825**	**2.155838616**

The results show that the technique Navizon employs outperforms the technique of relying on the first sample by an average of over 4.5 times and outperforms using the strongest sample by over 2 times.

Surprisingly, the average algorithm, which is an unweighted centroid, outperforms the Navizon algorithm for small data sets. This leads a possible change to Navizon in the future, and is discussed as part of the Future Work section in the last chapter of the thesis.

10.3 Appendix C – Domino example components

This section contains the source code from two example Domino modules, one from the *Castles* game and one from one of the early test applications. They are included simply to demonstrate that the Domino interface is simple to implement, and that not every method from the Domino interface actually requires code to be added, and can instead remain as a stub.

10.3.1 Example 1: Building from Castles

```
using System;
using System.Collections;
using System.Drawing;
using CastlesLib.Units;
using CastlesLib.Cargos;
using Domino.ComponentModel;

using System.Windows.Forms;

namespace CastlesLib.Buildings
{

        public abstract class Building : Domino.ComponentModel.IDominoComponent
        {

                protected static Package smallBuildingCost
                {
                        get
                        {
                                Package p = Package.Empty;
                                p.Add(new Cargo(CargoType.Iron, 2));
                                p.Add(new Cargo(CargoType.Stone, 4));
                                p.Add(new Cargo(CargoType.Wood, 4));
                                p.Add(new Group(new Peon(), 1));
                                return p;
                        }
                }

                protected static Package mediumBuildingCost
                {
                        get
                        {
                                Package p = Package.Empty;
                                p.Add(new Cargo(CargoType.Iron, 4));
                                p.Add(new Cargo(CargoType.Stone, 7));
                                p.Add(new Cargo(CargoType.Wood, 6));
                                p.Add(new Group(new Peon(), 2));
                                return p;
                        }
                }

                protected static Package largeBuildingCost
```

```
        {
            get
            {
                Package p = Package.Empty;
                p.Add(new Cargo(CargoType.Iron, 5));
                p.Add(new Cargo(CargoType.Stone, 10));
                p.Add(new Cargo(CargoType.Wood, 10));
                p.Add(new Group(new Peon(), 4));
                return p;
            }
        }

        private Recer.Paths.RankableItem[] contextItems;
        private IManager manager;
        protected Package baseProduce = Package.Empty;
        protected Package runCost = Package.Empty;
        protected int buildCycles = -1;
        protected Package cost = Package.Empty;
        protected string imageFile = "NoImage.png";
        protected string description = "BaseBuilding";
        protected string name = null;
        protected int buildRate = -1;
        protected Size size = new Size(4, 4);
        protected Point location = new Point(0, 0);
protected bool enabled = true;
protected bool shop = false;

        private int currBuild = 0;

        private BuildingAdapter buildingAdapter = null;

        private Package stock = Package.Empty;

        private bool active = false;

        public Recer.Paths.RankableItem[] ContextItems
        {
            get
            {
                return contextItems;
            }
        }
        public Size Size
        {
            get
            {
                return size;
            }
        }
```

```
        public Point Location
        {
                get
                {
                        return location;
                }
                set
                {
                        location = value;
                }
        }

public Package Produce
{
   get
   {
      return (Package)baseProduce.Clone();
   }
}

        public bool Active
        {
                get
                {
                        return active;
                }
                set
                {
                        active = value;
                }
        }

        // how often this thing cycles
        public int BuildRate
        {
                get
                {
                        return buildRate;
                }
        }

        public BuildingAdapter BuildingAdapter
        {
                get
                {
                        return buildingAdapter;
                }
                set
                {
                        buildingAdapter = value;
```

```
                buildingAdapter.Building = this;
            }
        }

        public void LeavePackage(Package pack)
        {
            stock.Add(pack);
        }

        public bool TakeStock(Package pack)
        {
            if (!stock.CanFulfill(pack))
                return false;
            stock.Remove(pack);
            return true;
        }

        public Package Stock
        {
            set
            {
                stock = value;
            }
            get
            {
                return stock;
            }
        }

        public virtual Package Cycle()
        {
Package outGoing = Package.Empty;

//this works for the barracks it seems like the baracks constructor is not setting the baseProduce;
//stock.Remove(runCost);
//outGoing.Add(new Group(new Soldier(), 1));

  if (!active)
    return outGoing;
  if (buildRate >= 0)
    if (currBuild++ < buildRate)
      return outGoing;
  currBuild = 0;
  if (stock.CanFulfill(runCost))
  {
    stock.Remove(runCost);
    outGoing.Add(Produce);//problem is with baseProduce here
  }
  if (buildingAdapter != null)
  {
```

```
        if (stock.CanFulfill(buildingAdapter.RunCost))
        {
            stock.Remove(buildingAdapter.RunCost);
            if (buildingAdapter.ActiveThisCycle())
            {
                outGoing.MultiplyCargo(buildingAdapter.CargoMultiplier);
                outGoing.MultiplyGroup(buildingAdapter.UnitMultiplier);
            }
        }
    }
    return outGoing;
                }

                public void Paint()
                {

                }

                public Package RunCost
                {
                        get
                        {
                                return runCost;
                        }
                }

public Package Cost
{
    get
    {
        return cost;
    }
}

                public int BuildCycles
                {
                        get
                        {
                                return -1;
                        }
                }

                public Image Image
                {
                        get
                        {
                                return new Bitmap(imageFile);

                        }
                }
```

```
/// <summary>
/// The building is a shop style
/// </summary>
public bool Shop
{
   get
   {
      return shop;
   }
   set
   {
      shop = value;
      buildRate = 0;
   }
}

                public string Name
                {
                        get
                        {
                                return name != null ? name : GetType().Name;
                        }
                }

                public string Description
                {
                        get
                        {
                                return description;
                        }
                }
                #region IComponent Members


                public void Start()
                {
                        //MessageBox.Show("Building dynamically constructed");
                        //Console.WriteLine("Asking manager for something that supports "+typeof(Building));
// doesnt output at all, use messagebox
                        IDominoComponent container = manager.FindDependency(typeof(Building));
                        if(container!=null)
                        {
                                //MessageBox.Show("adding to "+container);
                                manager.AddDependency(container,this);
                        }
                        else
                        {
                                MessageBox.Show("container was null");
                        }
```

```
}

public void Pause()
{
}

public void SetLocation(Point p)
{
}

public void StartComplex(Recer.Paths.RankableItem[] items)
{

}

public Recer.Paths.RankableItem[]  GetContextRankItems()
{
        return contextItems;
}


public void Destroy()
{
}

public bool CanSupport(Type type)
{
        return false;
}

public IDominoComponent[] GetChildren()
{
        return null;
}

public void SetManager(IManager manager)
{
        this.manager = manager;
}

public void AddDependent(IDominoComponent child)
{
}

public void SetContextRankItems(Recer.Paths.RankableItem[] items)
{
       contextItems = items;
}

#endregion
```

```
        }
}
```

10.3.2 Example 2: Map from Example application

```
using System;
using System.Data;
using System.Drawing;
using Domino.ComponentModel;
using Domino.Tools;
using System.Reflection;
using System.Windows.Forms;
using System.Collections;

namespace MapLib
{
        /// <summary>
        /// Summary description for Class1.
        /// </summary>
        public class MapLib : Control,IComponent
        {

                private IManager manager;
                private Font font;
                private Bitmap image;
                private ArrayList Layers;
                private Bitmap buffer;

                public MapLib()
                {
                        font = new Font("Times",12,FontStyle.Bold);
                        Layers = new ArrayList();
                        image = new Bitmap(Assembly.GetExecutingAssembly().GetManifestResourceStream("MapLib.map.png"));
                }

                protected override void OnPaintBackground(PaintEventArgs e)
                {

                }


                protected override void OnPaint(PaintEventArgs e)
                {
                        if(buffer==null)
                                buffer = new Bitmap(this.Width,Height);

                        Graphics g = Graphics.FromImage(buffer);
                        g.DrawImage(image,0,0);
                //      g.DrawString("Map",font,new SolidBrush(Color.Red),20,80);
                        foreach(Layer l in Layers)
```

```
            {
                    l.PaintScaled(buffer);
            }
            g.Dispose();
            e.Graphics.DrawImage(buffer,0,0);

        }

        #region IComponent Members

        public void SetSize(Size size)
        {
            this.Size = size;
        }

        public void SetLocation(Point p)
        {
            this.Location = p;
        }

        public IComponent[] GetChildren()
        {
            return (IComponent[])Layers.ToArray(typeof(IComponent));
        }

        public void StartComplex(Recer.Paths.RankableItem[] items){
        }

        public void Start()
        {
            // find a form
            Logger.WriteLine(this,"Asking manager for something that supports "+typeof(Control));
            IComponent container = manager.FindDependency(typeof(Control));
            if(container!=null)
            {
                Logger.WriteLine(this,"adding to "+container);
                manager.AddDependency(container,this);
            }
            else
            {
                Logger.WriteLine(this,"container was null");
            }
        }

        public void Pause()
        {
        }

        public void Destroy()
        {
```

```
        }

        public bool CanSupport(Type type)
        {
            return type.Equals(typeof(Layer));
        }

        public void SetManager(IManager manager)
        {
            this.manager = manager;
        }

        public void AddDependent(IComponent child)
        {
            Layers.Insert(0,child);
            this.Invalidate();
        }

        #endregion
    }
}
```

10.4 Appendix D – Navizon availability testing

To test the availability Navizon provides—how often it is able to locate the user with any of the methods it is capable of using—a trial participant was asked to carry a device running a slightly edited version of Navizon on an iMate SP5 during a shopping trip with her friends in the centre of Glasgow. The iMate was connected to a Bluetooth GPS unit and had 802.11 and GSM enabled, allowing Navizon to use all three forms for positioning. The only changes made to Navizon were the addition of counters which tracked the time that each of the positioning technologies was able to provide a location, as well as a counter that tracked the time that a location was not available from any of the three positioning systems.

The device was activated shortly after the participant arrived in town and collected and stopped just before she left. It did not record her journeys to or from the city. Altogether, the participant spent just under 4 hours in the city, and the device recorded data over this entire period. The tables below show the time, in milliseconds, and the ratios that each positioning technology was available.

802.11

	Time (ms)	Percentage (%)
No coverage	42074390	31.65
Coverage	90850779	68.35

GSM

	Time (ms)	Percentage (%)
No coverage	223748	00.17
Coverage	132701421	99.83

GPS

	Time (ms)	Percentage (%)
No coverage	102086823	76.80
Coverage	30838346	23.20

Any

	Time (ms)	Percentage (%)
No coverage	198184	00.15
Coverage	132726985	99.85

10.5 Appendix E - code to convert from latitude and longitude (WGS84) to OSGB coordinates

```
public class CoordConverter
        {

                private const double airyA = 6377563.396;
                private const double airyInvF = 299.3249646;
                private const double airyF = 1/airyInvF;

                private const double RADIANS_PER_DEGREE = 1.74532925199432957692369076848E-2;
                private const double Origin_lon_g = -2;
                private const double Origin_lat_g = 49;
                private const double X0 = 4.0E+5;
                private const double Y0 = -1.0E+5;
                private const double K0 = 0.9996012717;
                private const double b = (airyA) * (1-(airyF));
                private const double Eps2 = (airyF) * (2.0-airyF);
                private const double EE2 = Eps2 * Eps2;
                private const double EE3 = EE2 * Eps2;
                private const double Epps2 = (Eps2) / (1.0 - Eps2);

                private const double WGSa = 6378137.0;
                private const double WGSinvf = 298.257223563;

                private const double dx = 375;
                private const double dy = -111;
                private const double dz = 431;

                private static MapPoint translateDatum(double latitude, double longitude)
                {
                        double phi = latitude * Math.PI / 180.0;
                        double lambda = longitude * Math.PI / 180.0;

                        double a0;
                        double b0;
                        double es0;
                        double f0;

                        double a1;
                        double b1;
                        double es1;
                        double f1;

                        double psi;

                        double x;
                        double y;
                        double z;
```

```
double psi1;

a0 = WGSa;
f0 = 1.0 / WGSinvf;
a1 = airyA;
f1 = 1.0 / airyInvF;

b0 = a0 * (1 - f0);
es0 = 2 * f0 - f0 * f0;
b1 = a1 * (1 - f1);
es1 = 2 * f1 - f1 * f1;

if (latitude == 0.0 || latitude == 90.0 || latitude == -90.0)
{
        psi = phi;
}
else
{
        psi = Math.Atan((1 - es0) * Math.Tan(phi));
}

if (longitude == 90.0 || longitude == -90.0)
{
        x = 0.0;
        y = Math.Abs(a0 * b0 / Math.Sqrt(b0 * b0 + a0 * a0 *
Math.Pow(Math.Tan(psi), 2.0)));
}
else
{
        x = Math.Abs((a0 * b0) /
                Math.Sqrt((1 + Math.Pow(Math.Tan(lambda), 2.0)) *
                (b0 * b0 + a0 * a0 * Math.Pow(Math.Tan(psi), 2.0))));
        y = Math.Abs(x * Math.Tan(lambda));
}

if (longitude < -90.0 || longitude > 90.0)
{
        x = -x;
}
if (longitude < 0.0)
{
        y = -y;
}

if (latitude == 90.0)
{
        z = b0;
}
else if (latitude == -90.0)
{
```

```
                    z = -b0;
                }
                else
                {
                    z = Math.Tan(psi) * Math.Sqrt((a0 * a0 * b0 * b0) / (b0 * b0 + a0 * a0 *
Math.Pow(Math.Tan(psi), 2.0)));
                }

                psi1 = Math.Atan((z - dz) / Math.Sqrt((x - dx) * (x - dx) + (y - dy) * (y - dy)));

                latitude = Math.Atan(Math.Tan(psi1) / (1 - es1)) * 180.0 / Math.PI;
                longitude = Math.Atan((y - dy) / (x - dx)) * 180.0 / Math.PI;

                if (x - dx < 0.0)
                {
                    if (y - dy > 0.0)
                    {
                        longitude = 180.0 + longitude;
                    }
                    else
                    {
                        longitude = -180.0 + longitude;
                    }
                }
                return new MapPoint(latitude, longitude);
            }
```