

# Addressing Mobile Phone Diversity in Ubicomp Experience Development

Chris Greenhalgh<sup>1</sup>, Steve Benford<sup>1</sup>, Adam Drozd<sup>1</sup>, Martin Flintham<sup>1</sup>,  
Alastair Hampshire<sup>1</sup>, Leif Oppermann<sup>1</sup>, Keir Smith<sup>2</sup>, and Christoph von Tycowicz<sup>3</sup>

<sup>1</sup>Mixed Reality Lab, University of Nottingham

<sup>2</sup>University of New South Wales

<sup>3</sup>Hochschule Bremen

{cmg, sdb, asd, mdf, axh, lxo}@cs.nott.ac.uk,  
keirs@cse.unsw.edu.au

**Abstract.** Mobile phones are a widely-available class of device with supporting communications infrastructure which can be appropriated and exploited to support ubicomp experiences. However mobile phones vary hugely in their capabilities. We explore how a single dimension of *phone application type* embodies the critical trade-off between capability and availability, i.e. between what can be done and the fraction of potential participants' phones that can do this. We describe four different mobile phone ubicomp experiences that illustrate different points along this continuum (SMS, WAP/Web, and J2ME, Python and native applications) and the common software platform/toolkit, EQUIP2, that has been co-developed to support them. From this we propose four development strategies for addressing mobile phone diversity: prioritise support for server development (including web integration), migrate functionality between server(s) and handset(s), support flexible communication options, and use a loosely coupled (data-driven and component-based) software approach.

## 1 Introduction

A researcher who wishes to deploy a ubicomp experience outside of a lab setting has two options for technology platform: they can either deploy their own devices and infrastructure, or they must appropriate and build upon existing devices and infrastructure. The mobile phone is an excellent – and popular – example of such an existing device and supporting infrastructure.

In this paper, we describe the trade-offs which must be made when targeting mobile phones as a technology platform for ubicomp experience development. We illustrate these trade-offs through four different mobile phone experiences and the common software platform/toolkit, EQUIP2, that has been co-developed with them. From this we identify and reflect on four development strategies for effectively exploiting mobile phones.

We begin by considering the attractions of the mobile phone as a platform for ubicomp experiences. In section 3 we suggest that *handset application type* is the primary dimension for characterising handset capability, and identify some of the key

trade-offs inherent in targeting different points along this dimension. In section 4 we introduce the four proposed strategies for addressing mobile phone diversity in system and experience development. We then introduce four mobile-phone based ubicomp experiences and EQUIP2, the common supporting software platform that has been co-developed with these experiences (section 5). From these, in section 6, we reflect on the four strategies and their impact on EQUIP2 and the experience projects. Finally, section 7 presents our conclusions.

## 2 Background and Motivation

Perhaps the most compelling argument made in favor of using a mobile phone platform is its almost ubiquitous status in many countries today. For example, in 2006 in the UK mobile phones were owned by 85% of the adult population [1], were present in 90% of households [2, p.157] (the same fraction as have fixed line telephones) and had coverage across 99.9% of the country (2G) [2, p. 17]. For many users their mobile phone is an important and integral part of their everyday life: in the UK in 2005 73% of mobile users rated their mobile phone as “essential”, while 31% of all telecoms users used their mobile as the main method of making calls (up from 21% in 2004) [2, p.158].

We can identify two main ways in which ubicomp researchers can – and do – build upon and exploit mobile phones as a technology platform. The first approach uses the mobile phone as a system component which is readily available and relatively cheap (due to the large market). For example [3] uses mobile phones as personal interfaces to an augmented card-playing table, while [4] uses a mobile phone as a mobile client/interface for a mobile mixed-reality game. In each case the handsets are programmed and provided as part of a complete system and lent to users or provided in situ.

The second approach uses – or seeks to use – people’s own mobile phones as the technology platform. For example ContextPhone [5] augments users’ smart phones with a context-sensing software platform to support (e.g.) augmented contact management and tagged media sharing and [6] trials a location-based reminder application on mobile phones. Using people’s own mobile phones reduces the resources to be supplied by the ubicomp researcher, offers the possibility of a very large-scale deployment and means that the device is already part of a user’s everyday activities and routines.

However mobile phone handsets are extremely diverse, varying widely in display, input, memory capacity, processor speed, support for downloaded applications and networking. Many of the current projects which target mobile phones as a “ubiquitous” infrastructure actually target a very specific subset of handset capabilities. For example, [5] and [6] both require smart phones running a custom application (native or J2ME), and also (like many phone-based location-sensitive applications) require access to GSM cell ID which is only available via a native application on a particular range of operating systems.

In practice this restricts a project to some combination of: a small minority of current handset owners; giving or lending suitable handsets to participants; or a possible future in which those handset capabilities will dominate the market. In the

first case the handset is still part of the user’s own everyday routine, however the potential scale of the deployment is limited, and some areas of research are almost impossible (e.g. “viral” recruitment of social contacts who have different handsets). In all cases the potential scale of any current practical deployment (and therefore of any non-simulation evaluation) is quite limited.

### 3 Characterising Mobile Phone Diversity

We have been involved in designing and developing a number of ubicomp experiences based on mobile phone platforms, including those described in section 5. The most significant choice that has had to be made in each experience has been the type of handset application to be used within the context of the experience. The main trade-off is between the potential numbers of users on the one hand and the richness of functionality and interaction supported on the other hand. Figure 1 summarises the main options and trade-offs associated with this dimension of choice, and the remainder of this section considers each option in turn. We expect that the handset market will continue to maintain this diversity of capabilities, e.g. to reflect variations in user preference as well as the cost-sensitivity of the market.

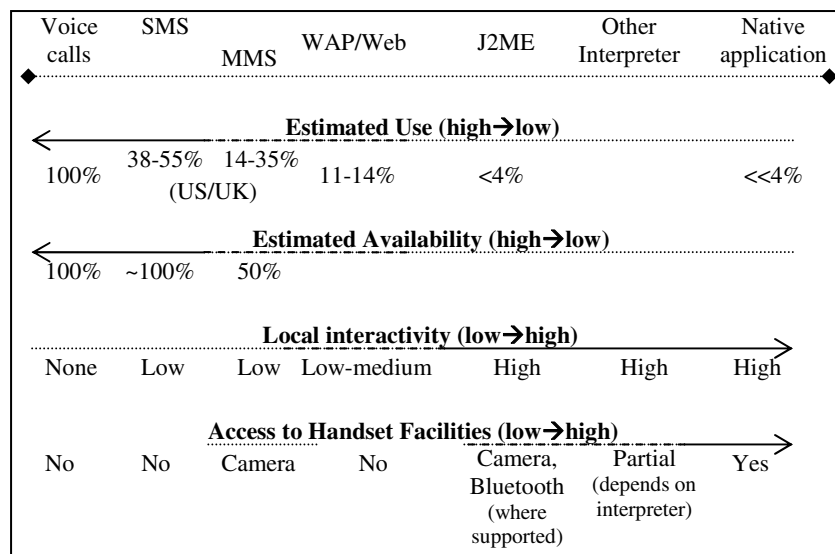


Fig. 1. Main options for mobile phone application type and associated trade-offs

#### 3.1 Voice Calls

Voice telephony is the only universally available service on mobile phones. However the use of voice telephony in ubicomp experiences appears to be limited to direct communication between players/participants without system mediation, e.g. [7]. Richer experiences could be supported by integrating VoIP (Voice over IP)

functionality into the system more generally, or by using phone operator location services which are available commercially on an explicit opt-in basis.

### 3.2 SMS/Text Messaging

After voice telephony, SMS or text messaging is the next most widely available option for incorporating mobile phones into a ubicomp experience. However availability and take-up varies worldwide, for example use of text messaging in the UK in 2006 was 85% of mobile phone subscribers but only 38% in the US [8]. Compared to voice, text messaging is much easier to integrate into a complete system (via one of the many commercial SMS gateway service providers).

A further sub-option here is MMS or photo messaging. On a phone with integrated camera this provides a richer capture capability for user content and context, or for interaction and location when used with server-based glyph reading technologies (as in CONQWEST [9]). However, take-up of picture messaging is substantially lower than for text messaging, with about 30.7% of UK subscribers and 14.5% of US subscribers in 2006 having used it [8] (36% and 38% of the numbers using text messaging in the UK and US, respectively). About 50% of subscribers actually have a handset with an integrated camera [10].

SupaFly [11] is one example of an experience (a pervasive game) based on SMS communication with players. Day of the Figurines (section 5.3) relies solely on SMS for mobile phone interaction, while LoveCity (section 5.4) uses SMS plus operator location as its “mass” participation option. SMS is also mentioned for direct player interaction in [7].

### 3.3 WAP/Web Browser

The next step up in capability is dynamic WAP/Web pages accessed via a (pre-installed) mobile phone browser. The main sub-options here are WML over WAP [12], which is more widely available on handsets (especially entry-level handsets) but limited in graphical richness, and XHTML over HTTP, which is currently limited to more capable (often smart phone) handsets but gives full HTML functionality and composition options. WAP/Web-based interfaces allow more complete interactive applications to be delivered via mobile phones. On some networks and contracts this may also be cheaper than using SMS. WAP/Web interfaces can be combined with WAP push/SMS to push new experience elements to the user, and can also be combined with operator location services.

However, as well as requiring a sufficiently capable handset, they require a contract which includes data services which often has to be configured by the user (at least selecting the correct “access point”). They also require reasonable network connectivity throughout the interactive session (each exchange with the server, which may be every page) and interaction latency is generally high (of the order of 3-5 seconds for even simple pages). The take-up of WAP/web-based services is much lower than for text messaging, with 14.4% of UK subscribers and (relatively higher) 11.7% of US subscribers using their mobile phone to browse news and information in 2006 [8].

Of the experiences described below Prof. Tanda (section 5.5) uses XHTML/HTTP as its main mobile phone interface, although in that case it is supported by an additional Python “trigger” application also running on the handset.

### 3.4 J2ME

The majority of ubicomp experiences on mobile phone platforms have a custom application running on the handset. Compared to the previous options, this can support richer interaction, reduce interaction time, give more complete access to the phone’s facilities (e.g. camera) and support peer-to-peer (e.g. Bluetooth) and disconnected operation. J2ME appears to be the most widely available platform for running applications on mobile handsets at the present time. In 2004 penetration of J2ME on 3G handsets was predicted to reach 75% this year [13], and 50% on 2.5G handsets. In the UK at the end of 2005 8.0% of households had a 3G handset/service [2]. In 2006, 4.2% of UK subscribers and 3.2% of US subscribers had actually downloaded a game (not necessarily J2ME-based) to their phone [8].

However J2ME is not a single option but rather a set of options. The minimal (most widely compatible) “profile” for J2ME is CLDC1.0/MIDP1.0, which is quite limited. Further sub-options include CLDC1.1 (adds floating point types), MIDP2.0 (adds better UI and more networking options), Bluetooth, Media (adds camera access) and so on. J2ME development also suffers from several complications, including buggy and inconsistent virtual machine and package implementations (see [14] and [15]). It is also not possible to access native APIs directly from J2ME applications; where this is required a separate native application must be deployed on the phone that can be communicated with via (e.g.) local HTTP (for example, ContextPhone [5] and PlaceLab [16] support such local interfaces).

[3] and [6] both use J2ME phone applications, which in [6] is also combined with a small native application (to obtain phone cell ID). The phone client for MobiMissions (section 5.6) is implemented in the same way.

### 3.5 Other Cross-Platform Interpreters

There are also a number of non-Java scripting engines or interpreters available for mobile phones (in particular, smart phones). In general these are native applications which, once installed, allow other content and applications to be downloaded and run. Current options include a Python interpreter (for Nokia Series 60 phones), Flash Lite (a cut-down version of Adobe Flash, available for Symbian and BREW) and Nokia’s MUPE multi-user application development platform [17] (the interpreter for which is a J2ME application).

These have the advantage of (generally) simpler development than for native applications, but typically require a large installation of the interpreter on the phone first, and (like J2ME) have more limited access to native phone facilities than a native application. LoveCity (section 5.4) has a Python-based smart phone client option, while Prof. Tanda (section 5.5) has a Python-based “trigger” application which runs in the background on the phone.

### 3.6 Native Application

As well as the general advantages of running a custom client, native applications are often essential to access the widest possible range of handset information. However, this also narrows the compatibility to very specific platforms and handsets. [14] and [15] deal at some length with issues of native application development and capability. At this level handsets are more diverse and less compatible than the common profiles of J2ME. The main platforms at present are Symbian (although the Nokia and Sony Ericsson UI layers differ), Windows Mobile and BREW (which is strictly a cross-platform development/deployment environment) (see [15]).

As already noted native helper applications are used in [6] and MobiMissions. The applications in [4] and [5] are realized as native applications.

Note that installing a native or J2ME application raises additional barriers to participation: the download may be slow (and expensive over the air), the user may not trust the application provider, or the user may not be familiar with this aspect of their phone's capabilities.

## 4 Introducing the Development Strategies

The following section introduces four strategies for addressing this diversity of mobile phones in the development of ubicomp experiences. These strategies respond to the characteristics and constraints of working with mobile phones as a hardware platform, and have shaped the development of the EQUIP2 software platform/toolkit introduced in section 5. We revisit these strategies in section 6 to explore *how* and they have influenced EQUIP2 and its applications and *how effectively* this has been.

Note that these strategies are primarily concerned with the software design and development process, rather than particular run-time capabilities. We have focused on strategies of this kind for a number of reasons. First, we are doubtful that a single run-time platform or set of run-time facilities can be applied to all possible types of mobile phone-based application/experience. Second, even if it were possible, developing such an all-encompassing solution would be a huge undertaking, beyond the scope of many research and development activities. Third, a single comprehensive software solution or framework can be a major barrier to use, requiring a broad range of skills and a large amount of up-front learning from any potential developer. Fourth, the resource-constrained nature of many mobile phones requires that optimizations (e.g. identifying and assembling minimal subsets of facilities) can be performed *before* deployment on a particular device, and dynamic code loading is not available on many phone platforms (e.g. J2ME CLDC). Finally, by reflecting on the choices and trade-offs made during development on practical projects we may identify new or critical elements to be incorporated in any future platform development or run-time infrastructure.

### 4.1 Prioritise Strong Server Support

At the present time the only truly mass-scale applications available and used on mobile phones are voice and SMS/text messaging. In both cases all custom application functionality must exist off the handset, normally on one or more "server"

machines which together comprise the “engine” for the experience. Even with J2ME or native phone applications many experiences still have a large server element, for example for communication and coordination.

In addition, many experiences of this kind have a web-based component (accessed from regular PCs) in addition to the phone-based component of the experience. This is true of all four experiences described here (and of others such as [11]). A standard web-based interface can provide richer and more extensive content than the phone interface (e.g. larger screen, various plug-in options), and is also useful “behind the scenes” for management, monitoring, authoring and “orchestration” of the experience (as in [18]).

Consequently we argue that a platform to support development of experiences delivered on mobile phones should prioritise support for server development.

#### **4.2 Migrate Functionality from the Server to More Capable Handsets**

It is relatively difficult and slow to develop applications on mobile phones, for example due to limited debugging support, inconsistencies and errors in virtual machines and libraries, and slow development cycles (see [14], [15]).

Consequently, especially given good support for server development (section 4.1), we argue that it is a good strategy to develop initial functionality on the server and then to migrate that functionality to higher-capability mobile phone client applications as required. This also allows the initial server-based version(s) to be used with less capable handsets.

Different kinds of functionality can potentially be migrated in different ways and at different times. At one extreme mobile code or agent approaches such as that of LIME [19] allow run-time migration of functionality between mobile devices. However this is contingent on the underlying platform’s support for code hosting and mobility which is absent in most mobile phone application types (including J2ME CLDC) and this is therefore not a general solution for mobile phone-based applications. At the other extreme – as described above – functionality can be migrated by the developers during the development, testing and deployment process. The challenge in this case is to make this migration as easy as possible, for example maximizing opportunities for re-use between server-based and phone-based versions.

#### **4.3 Support Very Flexible Communication**

We argue that a software platform to support ubiComp experience development for mobile phones needs to support very flexible communication in at least three respects.

First, any server(s) may need to communicate with different types of phone client including SMS, WAP/Web, J2ME and native application.

Second, in the case of custom applications running on the handset, it must be possible to carefully structure and optimize the communication between that client and the server. There are several reasons for this: communication with mobile phones is relatively high latency and low bandwidth; it may be expensive for the user; and there may be repeated or extended periods when a connection cannot be obtained or maintained. All of these can have a profound impact on the user’s experience, which may need to be designed to explicitly account for them and reflect them to the user.

Third, the server may also need to communicate with diverse other clients in addition to mobile phones, for example public and web clients as in Day of the Figurines and LoveCity (sections 5.3 and 5.4).

We argue that as a result there will be no single best solution for communication in these kinds of experiences. Consequently, a supporting platform should provide help with realizing and integrating communication options, rather than a specific solution (or a closed set of solutions).

#### **4.4 Use a Loosely Coupled (Data-Driven and Component-Based) Software Approach**

The last strategy that we argue for is a software system design approach which is data-driven and component-based, creating a loosely coupled and flexible system. This is not specific to mobile phone development, but rather is a general strategy which can enable reconfiguration and reuse of software elements within an application or family of applications. This is particularly important for mobile phone-based experiences because of their potential diversity and in order to effectively support the previous three strategies with limited development resources. There are three potentially complementary elements to such an approach.

First is an emphasis on starting with the key data structures (object classes, tuples, or whatever) to be used within the application. These provide a common language for design and development without the assumption of direct invocation (local or remote) implicit in an API, method or procedure call definition. This makes it closer to defining Data Transfer Objects than Domain Objects [20], and reflects the limited and variable connectivity experienced by many mobile phone-based experiences.

Second is the adoption of indirect communication and coordination models where possible. Options here include publish-subscribe event systems [21] and tuple-spaces and similar approaches [22]. Both support indirect multi-party communication through pattern matching, which in the case of tuple spaces has an additional element of statefulness, decoupling communication in time.

Third is the adoption of component-based software design and re-use wherever possible [23]. This differs from object-oriented design by a greater emphasis on re-use, in particular by configuration and assembly as against subclassing and inheritance. This avoids some of problems of fragility and framework buy-in common to object-oriented approaches.

## **5 Introducing EQUIP2 and the Experience Projects**

This section introduces the EQUIP2 platform/toolkit and four mobile phone-based ubicomp experiences that have been co-developed with it. Section 6 then explores the ways in which the strategies suggested in section 4 have been reflected in and supported by EQUIP2, and based on a developer evaluation of EQUIP2 also characterizes the perceived utility of those strategies as embodied in EQUIP2.

### 5.1 From EQUIP Version 1 to EQUIP2

EQUIP version 1 was developed and used to support a number of ubicomp experiences, primarily interactive installations, spaces and artefacts, for example in [24]. At the heart of EQUIP version 1 was a “dataspace” API, i.e. a tuple-space [22] which stores strongly typed (programming language) objects rather than untyped tuples. As the name implies a dataspace can be thought of as an information space into which objects can be placed and from which they can be retrieved or removed. Most retrieval/removal operations on a dataspace (or tuple-space) are based on pattern matching (i.e. “find objects like this...”). A single application may have one or many dataspaces, and each can be regarded as a sharable *model* in the sense of the Model-View Controller pattern/paradigm [25].

In this way EQUIP version 1 supported part of our fourth strategy: the dataspace approach supports loosely-coupled, data-driven communication. However EQUIP 1 was a substantial software framework for Java and C++, requiring extensive use and extension of framework classes, requiring a high level of buy-in from the developers and creating framework specific (less reusable) code. It also used a subset of CORBA IDL for language-independent type definition and had its own build system which were alien to most developers.

With regard to the strategies proposed here for mobile phone development EQUIP version 1 had many short-comings. First, there was no particular support for web-oriented server development (although there was some support for LAN-based servers, e.g. through multicast discovery and distributed dataspace facilities). Second, there was no support for J2ME or mobile phone operating systems such as Symbian, and therefore no way to migrate EQUIP elements to a handset. Third, the EQUIP version 1 dataspace had one built-in distribution protocol which worked well on LAN, wired broadband and local WiFi networks, but which did not cope well with intermittent or limited bandwidth connectivity (as with GPRS, Bluetooth or mobile WiFi).

EQUIP2 was developed to combine the best element(s) of EQUIP 1, in particular the dataspace approach, with support for experiences having mobile phone-based elements. Specifically, it was developed initially in tandem with and to support the Day of the Figurines and MobiMissions experiences described in sections 5.3 and 5.5.

### 5.2 EQUIP2 Overview

As with version 1, the core of EQUIP2 is a dataspace API (redesigned in some of its details compared to version 1). The dataspace API has two main parts. The first is a synchronous data storage and query interface which is very similar to an object database, and which (in EQUIP2) was inspired in part by the Hibernate open source Java Object/Relational mapping system. Like Hibernate, and unlike previous versions of the platform, the objects placed in the dataspace do not have to support particular (Java) interfaces or extend particular base classes. In most cases they are simple JavaBeans or “Plain Old Java Objects” (POJOs), and tend to be entirely passive data holders (with no internal threads or overridden methods).

The second part of the dataspace API is an asynchronous pattern-based notification facility, which allows sections of code to register an interest in certain kinds of objects

being placed into, changed or removed from the dataspace. This provides facilities comparable to a content-based publish-subscribe event system such as ELVIN [21], but tightly integrated with the dataspace's state management facilities: the "published" events are actually all changes made to the dataspace.

EQUIP2 is written in Java, using language features and classes common to both J2ME (for use on mobile phones) and J2SE (for use on more capable devices and server machines). Consequently it cannot make use of J2SE reflection facilities (which Hibernate does) since these are not present in J2ME, but has to provide its own simple reflection-like facilities (through a system of dynamically loaded "helper" classes). Similarly, it has to provide its own object marshalling framework (J2SE object serialization is again dependent on reflection), which currently supports XML and binary encoding options. The dataspace API is common across J2ME and J2SE, but different dataspace implementations are available on each platform. An experimental Java-to-C++ translator is used to generate the C++ version of EQUIP2 from a subset of the Java version, and currently supports Windows (MSVC++) and Linux (GCC/g++), with partial support for Symbian.

All of the experiences presented have an EQUIP2-based server component, which is based on a common template web application for use in a J2EE (Java 2 Enterprise Edition) servlet container such as Apache Tomcat. As illustrated in figure 2, this combines: a persistent EQUIP2 dataspace based on Hibernate over a relational database (in these cases, MySQL); an EQUIP2 Java Server Pages taglib for dataspace access from JSPs; a set of generic form views for browser-based viewing and editing the dataspace content; and forms for bulk XML and binary upload and download of the dataspace content. This also uses the Java Spring Framework [26] for declarative application composition and web-based MVC support.

The template web application provides a common starting point for each experience, which is then specialized by defining experience-specific data classes, followed by iterative development of experience-specific user interfaces (for participants, authors, operators and analysts) and application logic. Client applications can be developed concurrently as server capability matures, making use of EQUIP2 where appropriate (e.g. as in the MobiMissions J2ME client).

The remainder of this section briefly describes four experiences that been developed and deployed over the past two years using EQUIP2 and in collaboration with three different user groups: Blast Theory for Day of the Figurines and Prof Tanda; Active Ingredient for Love City; and FutureLab for MobiMissions.

### 5.3 Day of the Figurines (DoF)

DoF [27] is a role playing game for mobile phones that employs text messaging in order to be widely accessible to large numbers of players who can use their own mobile phones. The game follows twenty four hours in the life of a small virtual town which are mapped onto twenty four days of real time. It is therefore a long-term, slow-paced game that unfolds in the backgrounds of players' lives, requiring them to send and receive just a few messages each day. Players join the game by visiting a physical venue where they register their details and choose a plastic figurine to represent them. After they leave the venue, players control their figurine by sending and receiving SMS text messages.

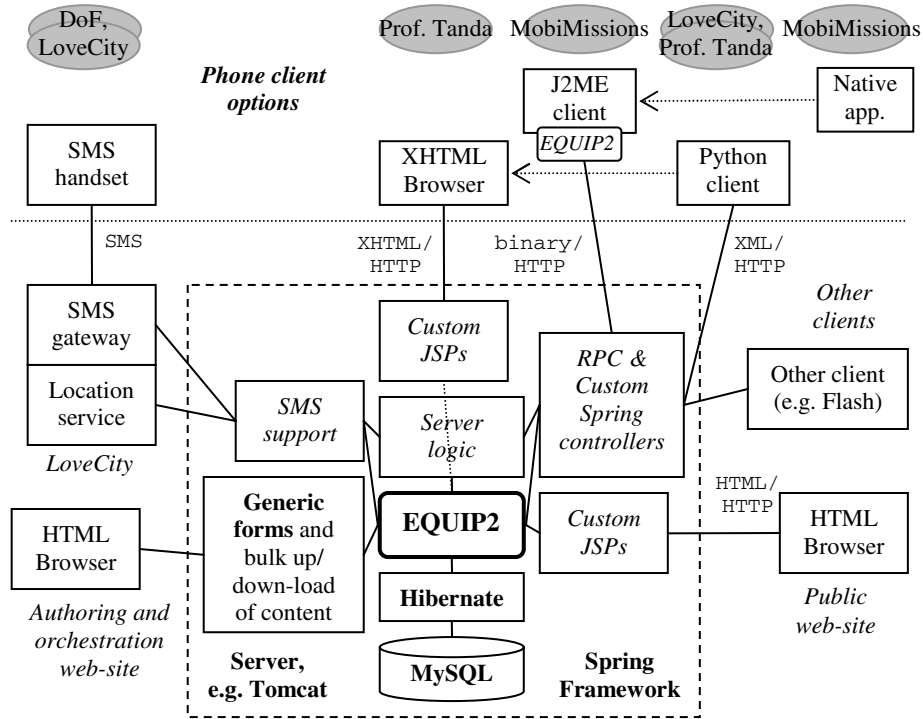


Fig. 2. Common and optional elements of the EQUIP2 server platform

As well as the SMS interaction with players’ phones there is also a public spectator interface where human game operators move the players’ figurines around a large physical game board that models the virtual city. A player can also view their figurine’s state and messages sent and received on a game web site.

As part of a process of iterative development, DoF has so far been deployed in London (85 players), Barcelona (160 players), Berlin (145 players) and Singapore (141 players). In its most recent outing in Singapore, players sent a total of 12,685 messages to the game, received a total of 21,767 from it, and 75% of the 24 who responded to a questionnaire said that they would play again.

### 5.4 Love City

Love City [28] is a location-based game for mobile phones that connects three different physical cities with and through a single virtual city. Players’ movements through their local physical cities are mapped to their location in the virtual Love City, enabling them to encounter one another. A player is given one chance to send a message of love – an anonymous text message – to each other player that they meet who may then choose to accept or reject it.

The implementation of Love City includes two different mobile phone clients. The first uses SMS messaging only on the handset in order to address as wide an audience

as possible. This is supported by network operator location of the handset where available (on three out of the four major networks in the UK in the initial trial). The second provides a more sophisticated graphical interface delivered through a Python client application. This utilises cell ID positioning in the client and communicates with the game server over HTTP. A central website enables players to gain an overview of the state of Love City including recent messages that have been sent, “offspring” that have been created and the virtual locations of other players.

For more details of Love City, including results from initial trials, see [28].

### 5.5 Prof. Tanda

Prof. Tanda is a mixture of a game and survey and is intended to engage players during their daily routines, providing them with amusement and information in return for data about their lifestyle, environmental actions and attitudes. Prof. Tanda is played up to twice each day for about ten minutes per session. The game is embodied through a quirky character called Professor Tanda who contacts the player, tries to guess where they are and what they might be doing, asks them questions and even gets them to undertake simple activities and experiments such as measuring the amount of water that they use when taking a shower by leaving the plug in the bath.

Prof. Tanda is currently realized as a “trigger” application implemented in Python and running in the background on the phone and XHTML pages generated by the server. Each night the trigger application downloads details of when it should next trigger a session from the server. It then monitors the time and current cell ID, and when the condition is met (or when the player explicitly “calls” the Professor from the trigger application) it directs the phone’s web browser to a session-specific URL on the server, which generates the XHTML pages for the player’s interaction with the Professor during that session.

This initial implementation of Prof. Tanda has recently been trialed by 20 players for two weeks. In general, they reported enjoying their interactions with the central character in the game, especially the way in which it engaged them in local activities, an aspect of the game that they would like to see extended in future versions.

### 5.6 MobiMissions

MobiMissions [29] is a game in which players use camera smart phones to create, complete and document real-world missions. The content and purpose of missions is left open-ended for players to define for themselves, each mission being defined by up to five photographs and/or five sections of text. Missions are located based on the player’s mobile phone cell ID. As a player carries out a mission they document their progress by capturing up to five photographs and adding short text annotations. They also rate how good the mission was.

The MobiMissions phone client is implemented in J2ME, specifically CLDC1.1, MIDP2.0 and the media API (for camera access). Communication with the game server is over HTTP. Cell ID is determined using the Placelab server (a native Symbian application) that runs on the handset [16], limiting the experience to Symbian Series 60 phones. The mobile game is supported by a website which allows

players to browse missions and responses, rate other players' responses to a mission and leave comments.

An initial trial of MobiMissions has been conducted with a group of seventeen 16-18 year old users who created 75 missions which generated 123 responses over a period of five weeks (see [29]). Feedback through participant diaries and interviews showed that these players generally enjoyed the experience.

## **6 Discussion**

We have now briefly described four ubicomp experiences which employ mobile phones in various ways and the EQUIP2 software platform/toolkit that has supported and been co-developed with them. From this, the following section returns to the four strategies identified in section 4 and assesses their impact on EQUIP2 and the development of these experience projects.

In addition to drawing on our own direct experience of development we have also surveyed the five core developers who worked on these four experiences using an optionally anonymous web-based questionnaire. These developers were not primary developers of EQUIP2 itself, and it was made clear to all respondents that an impartial response was desired, and that responses would be treated anonymously and used only for evaluation and planning for future extensions to the software platform.

### **6.1 Prioritise Strong Server Support**

This has been a major area of development in EQUIP2, in particular the template web application and associated reusable elements (taglib, forms, Hibernate-based dataspace implementation, SMS support). All of the experiences developed with EQUIP2 have had substantial web-based elements, including public web pages and operator pages (based on the JSP taglib) and authoring and monitoring pages (based on the generic forms).

All of the respondents to our developer survey thought that the use of EQUIP2 made experience monitoring and orchestration faster than it would have been with alternative technologies, and three of the five respondents reported that the generic database forms were one of the best elements of using EQUIP2.

### **6.2 Migrate Functionality from the Server to More Capable Handsets**

In MobiMissions all possible user interaction has been migrated to the J2ME phone client to integrate access to the phone's camera, minimize communication with the server and support play in areas of poor network coverage. This client application is built around a persistent EQUIP2 dataspace running on the phone, which caches information from the server (missions) to minimize communication with the server. The phone dataspace also accumulates new information and images (captured with the phone camera) as the user performs those missions, which can be uploaded to the server at a later time. The developer who worked on the phone client for MobiMissions rated the use of EQUIP2 for this as faster, less complicated and more flexible than using J2ME without EQUIP2.

In Prof. Tanda the trigger application represents a migration of one key function to the handset: monitoring certain aspects of user context (time and cell ID) in order to trigger a new interactive session. At present other phone interactivity is provided by the server-generated XHTML pages, however the server implementation of this uses Java over the standard EQUIP2 API and is designed so that in a future version this could be run directly within a J2ME MIDlet on the phone, similar to MobiMissions. The same approach allows (non-UI) EQUIP2-dependent code intended for a phone-based (J2ME) application to be initially developed and tested within desktop J2SE applications.

### **6.3 Support Very Flexible Communication**

EQUIP2 reflects this strategy because, unlike its predecessor, it has no fixed communication mechanism (e.g. no single built-in distributed dataspace protocol). Instead it provides a number of supporting facilities that can be assembled and tailored in different ways.

For example, in DoF there are a set of data classes and supporting web interfaces for linking to commercial SMS gateways which have been reused in LoveCity. DoF and LoveCity also use EQUIP2's object marshalling framework with a simple generic (reflection-based) RPC server skeleton to support interaction with public Flash-based visualization clients.

In MobiMissions the protocol used between the J2ME phone client and the server again exploits the marshalling framework (this time the binary encoding, for performance). However the protocol has been carefully crafted in tandem with the user interface and interaction. For example, it is relatively common for the upload of a completed mission to fail part way through. If the user then goes to the server web site they may see the mission as completed, or not, depending on whether the failure occurred during the final acknowledgement phase of the upload. Consequently the protocol, data model, user views and user interaction all reflect the fact that a mission upload may have succeeded, definitely failed, or be in an unknown state.

All respondents to our developer survey rated EQUIP2 as making networking development more or much more flexible compared to alternative technologies they might have used, and two respondents rated it as much less complicated and much faster than the alternatives (of the other two respondents to these questions one rated both neutrally and one – who worked on with the very first release of EQUIP2 – more complicated/slower). One respondent reported speed of networking development as a main reason for using EQUIP2 in the future.

### **6.4 Use a Loosely Coupled (Data-Driven and Component-Based) Software Approach**

We have already argued that the dataspace approach (and other indirect communication approaches) can support loosely coupled application development and code re-use. For example, the SMS “component” from DoF effectively forms a link

between a dataspace and an external SMS gateway. However it does not “care” (or need any re-coding to deal with) how the data objects corresponding to incoming and outgoing text messages are processed or generated within the rest of the application – it just adds a new received message object to the dataspace when a SMS arrives, and requests the sending of a new message by the SMS gateway when a new message request object appears in the dataspace.

A number of other aspects of EQUIP2 also reflect this strategy. First, the integrated notification facility, which is present in a subset of tuple-space systems, provides additional support for decoupled but timely coordination. Indeed, four of the five respondents to our survey identified this as one of the best elements of using EQUIP2. Second, the build process for the template web application encourages the application developer to begin by defining the data types (object classes) to be used in the application. This was put forward by two of the five respondents as an important element. Third, the template web application makes extensive use of the Spring Framework [26], which in turn embodies a particular approach to component-based software development called “dependency injection”, which makes it relatively easy to separate and then reuse the components (objects) that make up an application, without sub-classing or re-coding. Note also that the consistency of the EQUIP2 API across different platforms and implementations makes it easy to create unit tests (e.g. using JUnit) against test dataspaces without the overhead of creating and populating dummy relational databases or J2ME record stores.

## 6.5 Related Platforms and Toolkits

As we argued in section 4 our strategies – and priorities in EQUIP2 – reflect development-time concerns more than run-time facilities. EQUIP2 does provide a number of common run-time facilities (e.g. local dataspaces and marshalling) and so may be regarded as a run-time platform. However “out of the box” it is not a *distributed* platform for mobile phone application development, at least in its current form. This can be contrasted with specifically distributed tuple space systems such as MobiSpaces [30]. We plan to add some distributed dataspace options to EQUIP2 but these will only be applicable to particular deployment situations and handset application types. This mix and match approach gives EQUIP2 its toolkit character.

As discussed in section 3 there are several different platforms for (smart) phone applications, including J2ME, BREW, Symbian, Python and ContextPhone. In addition OSGi [31] (and JSR 232) supports modular application development and management, including for mobile devices, but requires the Java CDC profile rather than the CLDC profile normally found on mobile phones. The scope of EQUIP2 in terms of supported handset(s) is broader than any one of these, both in that the C++ version of EQUIP2 targets Symbian as well as J2ME (and potentially also Python through a suitable wrapper), and also in that the web server and SMS support is an integral element of EQUIP2 as a whole.

MUPE combines smart phone client development with server support, but the phone client is limited to J2ME and the server support is limited to communication with MUPE clients (for multi-user experience development) and has no particular support for web development or linking to other kinds of handset. Some other platforms or services such as BREW and SnapMobile [32] provide similar

combinations of smart phone client support (as a library) and complementary server support for developing multi-user applications. But these are all adjuncts to smart phone client development on particular platforms, and do not offer support for other kinds of handsets or migration of functionality. None of the other ubicomp experiences considered in this paper identify a supporting platform or toolkit.

## 7 Conclusions and Future Work

The mobile phone is a very attractive existing device with a supporting infrastructure on which to deploy ubicomp experiences. However mobile phones are extremely diverse in their capabilities, so that an experience designer must choose which handset application type(s) they will target for any particular deployment, for example SMS, WAP, J2ME MIDlet or native application. This forces them to make a trade-off between the functionality of the handset application (e.g. interactivity and access to phone facilities) and the potential number of users having a compatible handset.

We have suggested four strategies which can ease development for mobile phone-based experiences given this trade-off: prioritise support for server development (including web integration), migrate functionality between server(s) and handset(s), support flexible communication options, and use a loosely coupled (data-driven and component-based) software approach. These strategies have shaped the development of the EQUIP2 software platform/toolkit, which underlies and has been co-developed with the four phone-based ubicomp experiences described in section 5. As we have argued, these are primarily development-time strategies. The experiences presented and the feedback from the experience developers suggest that these strategies, as supported and partially embodied by EQUIP2, have been effective in developing experiences across a range of handset application types, both in different projects and within the same project.

EQUIP2 is still under active development and being used in other projects and activities. It is freely available under the “new” BSD open source license: see the SourceForge project ‘equip’<sup>1</sup>. To date we have chosen not to provide any particular distributed dataspace facilities. However the experience with MobiMissions in particular points to issues of communication uncertainty and scheduling which should be considered by any such facility.

## Acknowledgements

With many thanks to our collaborators on these experiences. This work has been carried out for the EPSRC/DTI funded Participate project (grant EP/D033780/1). The development of EQUIP2 and the described experiences have also been supported by the EPSRC through the EQUATOR IRC (grant GR/N15986/01), by the EU IST programme through the Integrated Project on Pervasive Gaming (FP6 - 004457), and by Futurelab, the Arts Council England, the European Regional Development Fund and the East Midlands Development Agency.

---

<sup>1</sup> [equip.sourceforge.net/equip2/](http://equip.sourceforge.net/equip2/)

## References

1. TimesOnline: Mobile madness consumes the UK (2006),  
<http://technology.timesonline.co.uk/article/0,19510-2189680.html>
2. Ofcom (Office of Communications): The Communications Market 2006 (2006),  
<http://www.ofcom.org.uk/research/cm/cm06/main.pdf>
3. Floerkemeier, C., Mattern, F.: Smart Playing Cards – Enhancing the Gaming Experience with RFID. In: Proc. PerGames, pp. 27–36 (2006)
4. Cheok, A.D., Sreekumar, A., Lei, C., Thang, L.N.: Capture the flag: mixed-reality social gaming with smart phones. *Pervasive Computing* 5(2), 62–69 (2006)
5. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.: ContextPhone - A prototyping platform for context-aware mobile applications. *Pervasive Computing* 4(2), 51–59 (2005)
6. Sohn, T., Li, K.A., Lee, G., Smith, I., Scott, J., Griswold, W.G.: Place-Its: A Study of Location-Based Reminders on Mobile Phones. In: Beigl, M., Intille, S.S., Rekimoto, J., Tokuda, H. (eds.) *UbiComp 2005*. LNCS, vol. 3660, pp. 232–250. Springer, Heidelberg (2005)
7. Smith, I., Consolvo, S., Lamarca, A.: The Drop: Pragmatic Problems in the Design of a Compelling, Pervasive Game. *ACM Computers in Entertainment* 3(3), 1–14 (2005)
8. Metrics, M.: M:Metrics Unveils Industry’s First Definitive Mobile Marketing Metrics (2006),  
<http://www.mmetrics.com/press/PressRelease.aspx?article=20061003-sms-shorttext>
9. CONQWEST. <http://homepages.nyu.edu/~dc788/conqwest/>
10. M:Metrics: Captured By Camera Phones (2006),  
<http://www.mmetrics.com/press/PressRelease.aspx?article=20060807-photo-messaging>
11. Jegers, K., Wiberg, M.: Pervasive gaming in the everyday world. *IEEE Pervasive Computing* 5(1), 78–85 (2006)
12. Kumar, V., Parimi, S., Agrawal, D.P.: WAP: present and future. *IEEE Pervasive Computing* 2(1), 79–83 (2003)
13. Sun Microsystems: Sun in Telecom (2004),  
[http://www.sun.com/aboutsun/media/presskits/ctia2005/Sun\\_Telecom\\_External\\_V2.pdf](http://www.sun.com/aboutsun/media/presskits/ctia2005/Sun_Telecom_External_V2.pdf)
14. Huebscher, M., Pryce, N., Dulay, N., Thompson, P.: Issues in developing ubicomp applications on Symbian phones. In: Proc. Future Mobile Computing Applications, International Workshop on System Support, pp. 51–56 (2006)
15. Coulton, P., Rashid, O., Edwards, R., Thompson, R.: Creating entertainment applications for cellular phones. *Comput. Entertain.* 3, 3 (2005)
16. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., Schilit, B.: Place Lab: Device Positioning Using Radio Beacons in the Wild. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) *PERVASIVE 2005*. LNCS, vol. 3468, pp. 116–133. Springer, Heidelberg (2005)
17. Suomela, R., Räsänen, E., Koivisto, A., Mattila, J.: Open-Source Game Development with the Multi-user Publishing Environment (MUPE) Application Platform. In: *Entertainment Computing – ICEC*, pp. 308–320 (2004)
18. Crabtree, A., Benford, S., Rodden, T., Greenhalgh, C., Flintham, M., Anastasi, R., Drozd, A., Adams, M., Row-Farr, J., Tandavanitj, N., Steed, A.: Orchestrating a mixed reality game ‘on the ground’. In: Proc. CHI, pp. 391–398 (2004)
19. Murphy, A.L., Picco, G.P., Roman, G.-C.: Lime: A Coordination Middleware Supporting Mobility of Hosts and Agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 15(3), 279–328 (2006)

20. Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison Wesley Professional, Reading (2002)
21. Segall, B., Arnold, D., Boot, J., Henderson, M., Phelps, T.: Content Based Routing with Elvin4. In: *Proc. AUUG Winter Conference, AUUG2K*, June 2000, Canberra, Australia (2000)
22. Gelernter, D.: Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems* 7, 80–112 (1985)
23. Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*, 2nd edn. Addison-Wesley Professional, Boston (2002)
24. Humble, J., Crabtree, A., Hemmings, T., Åkesson, K.-P., Koleva, B., Rodden, T., Hansson, P.: Playing with the Bits - User-configuration of Ubiquitous Domestic Environments. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) *UbiComp 2003*. LNCS, vol. 2864, Springer, Heidelberg (2003)
25. Reenskaug, T.: The Model-View-Controller (MVC): Its Past and Present, [http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC\\_pattern.pdf](http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf)
26. Spring Framework, <http://www.springframework.org/>
27. Flinham, M., Smith, K., Benford, S., Capra, M., Green, J., Greenhalgh, C., Wright, M., Adams, M., Tandavanitj, N., Row Farr, J., Lindt, I.: Day of the Figurines: A Slow Narrative-Driven Game for Mobile Phones Using Text Messaging. In: *Proc. PerGames* (2007)
28. Oppermann, L., et al.: Love City: A Text-Driven, Location-Based Mobile Phone Game Played Between 3 Cities. In: Magerkurth, C., Röcker, C. (eds.) *Pervasive Games - Concepts & Technologies* (2007)
29. Grant, L., Benford, S., Hampshire, A., Drozd, A., Greenhalgh, C.: MobiMissions: The Game of Missions for Mobile Phones. In: *Proc. PerGames* (2007)
30. Fongen, A., Taylor, S.J.: Mobispace - A Distributed Tuplespace for J2me Environments. In: *17th IASTED International Conference on Parallel and Distributed Computing and Systems* (2005)
31. OSGi Alliance: <http://www.osgi.org/>
32. Nokia: SNAP Mobile, <http://snapmobile.nokia.com/>