Adaptable Mobile Applications: Exploiting Logical Mobility in Mobile Computing

Stefanos Zachariadis, Cecilia Mascolo and Wolfgang Emmerich

Dept. of Computer Science, University College London
Gower Street, London WC1E 6BT, UK
{s.zachariadis,c.mascolo,w.emmerich}@cs.ucl.ac.uk

Abstract. An increasing number of applications is being written for mobile hosts, such as laptop computers, mobile phones, PDAs etc. These applications are usually monolithic, featuring very limited interoperability and context-awareness and are usually difficult to deploy and update. Application engineers have to deal with a very dynamic set of environments that these applications are in contact with and it is becoming increasingly difficult to design an application that will be able to cater to all the user's needs in those environments. This new setting forces a shift from design-time to run-time effort in developing software systems. To solve these problems and to allow a new class of ubiquitous and adaptable applications to be built, we have designed and implemented SATIN, a middleware system that allows the flexible use of logical mobility techniques by applications running on mobile hosts which are connected to very different networks. In this paper we describe our approach and show how SATIN can be used to deploy and update applications on mobile devices easily and efficiently.

1 Introduction

With the recent developments in wireless networks (Wavelan, Bluetooth) and the sales of mobile computers of any kind (such as laptop computers, Personal Digital Assistants (PDAs), mobile phones etc.) soaring, we are experiencing the availability of increasingly powerful mobile computing environments which are exposed to an increasingly dynamic setting. As such, a new highly mobile scenario for mobile devices and applications is being created, potentially allowing for interaction with and adaptability to any changes in their setting, or the development of *context-aware* applications. The major characteristic of this scenario, is heterogeneity, in the software, hardware and networking levels as the devices that form it are composed of a large number of different applications, middleware systems and hardware and can access different networking infrastructures. This new setting forces a shift from design-time to run-time effort in developing software systems. The current industry state of the art proposes monolithic applications which feature little to no interoperability, forcing application developers to anticipate at the design stage what possible uses their software will have throughout its lifetime. To tackle the issues arising from such heterogeneity,

there is a need to create software systems that can automatically adapt to tackle changes to the environment and to users' needs. We have also recently witnessed the acceptance of logical mobility (LM) techniques, or the ability to ship part of an application or even a complete process from one host to another. As such, LM techniques have been successfully used to enhance a user's experience (Java Applets), to dynamically update an application (Anti-Virus software etc.), to utilise remote objects (RMI, Corba, etc), to distribute expensive computations (Distributed.net) etc. Whereas various mobile middleware systems have been developed, the use of LM in those systems has been very limited. We wish to show that providing the flexible use of LM primitives to mobile computing applications through a middleware system, will allow for a greater degree of application dynamicity, will provide new ways to tackle interoperability and heterogeneity as well as ease deployment. Section 2 continues with an introduction to LM as well as a summary of limitations of related work. Section 3 presents a case study for deploying applications on cellular phones. Section 4 identifies and describes the principles of our approach, while Section 5 describes how it can be used.

2 Background and Related Work

LM (LM) is defined as moving parts of an application or migrating a complete process from one processing environment to another. It has been classified[5] into the following set of paradigms: *Client - Server* (CS), a popular paradigm in traditional distributed systems, dictates the execution of a unit of code in a server, triggered by a client, which may receive any result of that execution. The most common example of this paradigm is the use of Remote Procedure Calls (RPCs). *Remote Evaluation* (REV) dictates that a host sends a particular unit of execution to be executed in another host. A result may or may not be needed, depending on the application. This paradigm is employed by Distributed.NET, Seti@Home and other similar distributed computing environments. In the *Code on Demand* (COD) paradigm, a host requests a particular unit of code from another machine, which is then shipped to the original host and executed. This is an example of dynamic code update, whereby a host or application can update its libraries and available codebase at runtime. Many examples of COD have recently emerged, due to the popularity of Java and its built-in class loading mechanism and object serialisation framework. A *Mobile Agent* (MA), is an autonomous execution unit. It is injected into the network by a host, to perform some tasks on behalf of the user or an application. The agent can migrate from a processing environment or host to another. The application of LM in Mobile Computing context has, up to now, been quite limited. Some research investigating the subject has, however, been carried out. Current efforts into this area can be roughly grouped into two categories: Approaches which use LM to provide reconfigurability in the mobile computing middleware itself, allowing applications to interact with services provided by heterogeneous platforms and middleware systems, and approaches that use certain paradigms of LM to provide particular

functionality to applications. Examples of the first category include ReMMoC[3], a middleware platform which allows reconfiguration through reflection and component technologies. It provides a mobile computing middleware system which can be dynamically reconfigured to allow the mobile device to interoperate with any middleware system that can be implemented using OpenCOM components. UIC[8], another example in the first category, is a generic request broker, defining a skeleton of abstract components which have to be specialised to the particular properties of each middleware platform the device wishes to interact with. Examples of the second category include Lime[7], PeerWare[4] and Jini[2]. Lime is a mobile computing middleware system that allows mobile agents to roam to various hosts sharing tuple spaces. PeerWare allows mobile hosts to share data, using REV to ship computations to remote sites hosting the data. Jini is a distributed networking system, which allows devices to enter a federation and offer services to other devices, or use COD to utilise services that are already offered. The problem of these approaches with respect to our work is that their use of LM is limited to solving specific problems of a limited scope. For example, Jini uses COD to offer services, and PeerWare uses REV to distribute computations. What the middleware system described in this paper does, is to provide the flexibility to offer the solutions that previous approaches do, but its use is not limited to these, as will be made clear in the following sections.

3 Case Study: Deploying Software on Mobile Phones

Mobile phones are becoming increasingly powerful: State of the art phones feature a fast CPU, large amounts of memory, are usually equipped with a number of networking interfaces, like IrDA, Bluetooth and GSM/GPRS and can mediate packets between the different networks. Hence, in addition to being able to connect to the network operator, modern phones have the ability to form Ad-Hoc networks with other devices that are in reach. These networks are very heterogeneous: Different hardware manufacturers provide different devices to users, equipped with different operating and middleware systems, as well as applications. Moreover, the environment to which these devices are exposed is inherently dynamic, as they are meant to be carried wherever the user goes. There have been some approaches that promote interoperability between applications running on mobile phones allowing for data synchronisation[6]; However, very few users actually update the software on their phones and mobile applications rarely react to their context and interact with each other. Industry state of the art mobile application development usually features large monolithic applications with very little code reusability.

Unfortunately, installing new applications and updating existing ones is still difficult. As a matter of fact, the only popular update is the download of ring tones and games to mobile phones. The source of the download is usually the network operator (see Figure 1(a)), and the cellular bandwidth, which is very expensive for both the user and the operator, is used for the transfer. This limited approach does little to tackle the problems of heterogeneity and context

Fig. 1. (a) Deploying applications (ring tones & games) to mobile phones. Even though the phones can communicate directly using Bluetooth, they are all downloading the application from the network operator using the cellular network.(b) Possible application deployment and update: dotted lines represent security certificate download (verifying the authenticity of the code). Solid lines represent update download

awareness mentioned in Section 1. As such, the rich resources that these devices provide are not used and users experience very little context interaction. These devices are powerful enough to be used for various augmented reality applications such as interactive museum guides and tourist guides, media players which dynamically learn new codecs and interact with various services on various middleware platforms (for example a printer in a Jini system).

We believe that a better scenario would be if the devices were to form peer-to-peer networks, where each phone can use others which are currently in reach to dynamically update itself. Advertising and discovery mechanisms would allow applications to search for particular functionality from peers and download it when needed, as well as react to any change in context. Upon entering a particular building for instance, the user would be able to interact using his or her mobile phone with services offered inside the building. The network operator could be used to provide some form of digital certificate as to guarantee the authenticity of the code. Charging for this service would still be feasible, as the provider could charge for the certificate. Popular applications would be easy to locate in the peer to peer network. Less popular ones would still need to be downloaded from the network operator or another centralised service. Figure 1(b) illustrates an instance of this scenario.

Advantages of this approach include efficient use of networking bandwidth as well as automated application update & reaction to context. It constitutes the use of LM primitives (COD specifically), in a mobile environment and it requires host & functionality identification, a way to pack, request, sign and ship this functionality, an advertising & discovery service and the ability to add functionality to the middleware at runtime.

4 Principles and The SATIN Architecture

To allow adaptation and flexibility in mobile applications and to realise the scenario mentioned above, we have identified a number of principles which we have implemented in our middleware system, SATIN. We give details of these principles below.

Fig. 2. (a) Our architecture in the context of the ISO/OSI networking model. SATIN capabilities represent the network, transport, session and presentation layers. (b) A high level view of SATIN. It is composed of modules, or capabilities, registered with a core.

Modularisation

For this approach to operate, we require the modularisation of both applications and the middleware system itself into a series of modules, or *capabilities*. A capability is a unit that provides a specific functionality to the user, the middleware or to other applications and adheres to a specific interface. As such, capabilities can range from a discovery technique, to a compression algorithm implementation, all the way to to a calendaring application.

The capabilities that are available to a particular host are registered with the host's registry, or *core*. The core is also a capability. An instance of SATIN is *statically* configured, if the core does not allow for the registration of new capabilities at runtime. Alternatively, it is *dynamically* configured. A reference to the handler of any capability is available to all capabilities that are registered with the core. We use a string identifier, to register capabilities with the core, which requires unique identification for each capability. As such, we do not allow for the existence of two different capabilities with the same identifier on a single host (a *locally* unique identifier). On the contrary, we propose the registration of the identifier on a centralised database, similar to the CreatorID that PalmOS[1] applications have (a *globally* unique identifier). The identifiers are thus defined by the Capability developers and checked with a centralised database to verify their uniqueness. Moreover, we allow for differentiating between revisions, or versions of each capability, by means of a version identifier. Note that implementations of the core can be distributed and not reside on the same host as the capabilities that are registered with it.

Sharply contrasted with the monolithic application development that is the current state of the art, this approach has various advantages. Considering that SATIN encourages and allows the addition of new capabilities at runtime, the identification approach allows for easily building dependency graphs of capabilities, which allows us to know whether a capability will be usable on a particular host. Moreover, it allows for identifying a capability without transmitting its interface. This decoupling approach combined with versioning allows for fine-grained application update and deployment, whereby we can update individual components of applications and libraries at runtime.

Advertising & Discovery

As our goal is to promote interoperability, adaptation and flexibility to mobile

applications, it is important to be able to advertise and discover what functionality and or services are in reach. Moreover, as we are dealing with such a heterogeneous environment, it is expected that there will be many different ways to do advertising and discovery: We might use broadcast or multicast techniques, registration to and querying from a centralised server, or even interoperation with an existing middleware system, such as Jini. The modularised infrastructure described above lends itself to this, as SATIN represents different discovery and advertising techniques as different capabilities, which can be added to a host dynamically when needed.

In SATIN, different functionalities are represented by different capabilities. Capabilities which wish to advertise their presence and some information about their functionality to other hosts are termed Advertisable Capabilities (see Figure 2). All advertisable capabilities have to define a text message that will be used to describe the capability. The message is encoded in XML. For example, let us assume an FTP capability, which provides an FTP server facility that wants to be advertised. In this scenario, the capability's message might be formated as `<port>21<port><anonymous/>`. This would imply that the server is listening on port 21 and allows for anonymous access. Herein lies the importance of global identifiers: If the FTP capability is registered and has a global identifier, hosts which receive this message can decode its meaning.

This approach decouples the advertising message from the advertising and discovery mechanisms. Advertisable Capabilities can decide which Advertiser to allow advertising them. An advertiser which is allowed to advertise a capability, receives the message from it, adds some information regarding the advertisable capability itself, and then sends it over the network. For example the FTP message given above would become `<capability id=''FTP'', version=''0''> <port>21</port> <anonymous/> </capability>`. The advantages of our approach to advertising and discovery are that by representing the advertising and discovery techniques as modules, we tackle the problem of network heterogeneity and allow devices to advertise and discover functionality that is available within their current reach using various different mechanisms which can be installed and removed when needed. Moreover, by decoupling the advertising message from the advertising mechanism, we promote standardisation on describing a capability regardless of the networking medium and mechanism it is advertised and discovered with, making it easier for developers to advertise their functionality and use others that are available. Note that an advertiser can be an advertisable capability itself. This can allow devices to learn of particular advertising techniques (multicast groups for example) which are currently in reach.

Application Adaptability through Logical Mobility

Using abstractions defined by modern programming languages, we can define the following logical items that can be transferred across the network: Classes, Objects, Remote Procedure Calls (RPC) and Application Data. Note that in this context, application data may include code that cannot be directly mapped to the underlying platform (sending Python code to a Java runtime for example). We define a Logical Mobility Unit (LMU) as a combination of any

of the above. In a mobile computing scenario, we add an extra dimension to the LM paradigms described above, the communication semantics. In a traditional distributed system (DS) synchronous communications is the most common communication paradigm, given the reliability of the network connection. In a mobile distributed system however, we are encountering both synchronous and asynchronous communications, with the latter being the norm, as mobile connectivity is usually fluctuating and error-prone. When tackling LM, there are more considerations that we need to take into account including how to use the received LMU, how to identify an LMU so that we are able to request it, how to formulate and structure an LMU for transfer and how to verify that the target platform is able to execute the received LMU, a problem arising from hardware and software heterogeneity. SATIN is well-suited for the construction of LMUs, because is promotes decoupling of applications into discrete modules of specified functionality, which we can identify and build dependency trees for. We represent LMUs as a container and provide mechanisms to specify the source and target hosts, the size of the unit when transfered and when deployed, an unpacker which can be used when the target host does not know how to utilise and deploy the unit received and an optional digital signature, specifying the validity of the LMU.

Recipients of LMUs can range from the core to any other capability present. Capabilities that can receive LMUs are called *extendable* capabilities (see figure 2). To use the LM mechanisms that we provide, capabilities need to use the *LM Deployment Capability* (LMDC). The LMDC is responsible for requesting, creating, sending, receiving and deploying LMUs to the appropriate extendables. The LMDC can do both synchronous and asynchronous transfer of LMUs, and allows extendables to query an LMU that is targeted at them before accepting it. A dynamically configured instance of SATIN must have an LMDC.

The advantages of this approach include that we have the flexibility of sending any logical part of an application to appropriate recipients flexibly, allowing for identification and security and for the building of dependency graphs of a particular unit. We also allow for inspection and rejection of LMUs by extendable capabilities, as well as for utilisation even if the recipient does not know how to. Our unpacker can be used to send threads around the network and our approach also allows for the arbitrary grouping of any logical parts as the situation may dictate.

5 Application deployment through SATIN

This section describes application deployment and updating on cellular phones, using the terminology of the principles we defined on Section 4.

Let us assume a mobile phone user, with a dynamic instance of SATIN and a media player application installed, using "MPLAYER" as capability identifier. "MPLAYER" is an extended capability, as it can be updated with new codecs. The telephone is equipped with a Wavelan card. The user decides to attend a conference. At the conference, there is a computer, also running SATIN, that

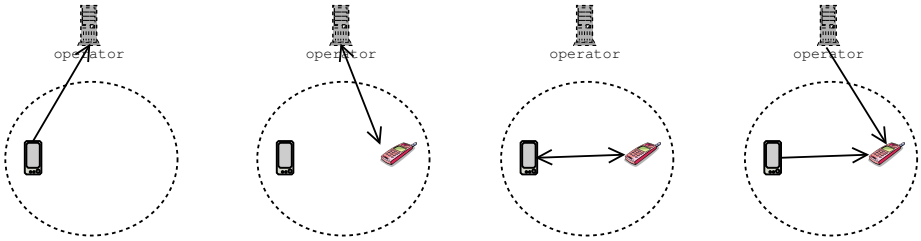

Fig. 3. (a) The conference computer's advertising service ("MULTICASTADV") registers its existence with the network operator. (b) The media player ("MPLAYER") running on the mobile phone queries the network operator for any other advertising services in the area, receives the capability "MULTICASTADV", gets charged for it and initialises it, so that it can listen to any other advertised services. (c) "MPLAYER" finds out about the stream and gets the theora codec ("THEORA" capability) from the conference computer. (d) "MPLAYER" receives a certificate for "THEORA", gets charged for it and receives the stream.

streams live videos of the presenters over Wavelan, so that attendees can get a better look of the presenter, or perhaps save the stream for later viewing, instead of taking notes. The user wishes to utilise this service, but there are the following problems: The service is advertised in a multicast group and the media player does not know which one. Moreover, the media player does not have the appropriate codec (say, theora) to display the video. We assume two advertising and discovery mechanisms, the first one registering and querying capabilities to and from a centralised host (the mobile phone operator) with identifier "CENTRALADV" and a multicast group discovery and advertising mechanism, with capability identifier "MULTICASTADV". The core's identifier is "CORE". We present a solution to this problem, using SATIN. Although some of the approaches described in section 2 could potentially be used to tackle this problem, the solutions would be too application dependent and not as flexible, forcing developers to take a number of limiting choices at design time. We illustrate the deployment of capabilities on the hosts on figure 4. Our solution is described below.

The conference's machine has an advertisable capability, "STREAM". Its advertisable message shows the location of the stream in the local network and the codec used. It only allows the "MULTICASTADV" advertiser to advertise it. "MULTICASTADV" is an advertisable capability itself, advertising the group and port it is advertising to. The "CENTRALADV" advertiser, advertises the existence of any advertisable capabilities that allow it to advertise them, to the cellular network's operator[1]. In this case, the only capability is "MULTICASTADV" which is thus registered with the cellular network's operator servers. The capabilities that "MULTICASTADV" advertises are "STREAM" and "THEORA" and their advertisable messages are periodically multicast to the group.

[1] This example assumes that the conference's computer will advertise its existence to the cellular network's operator. Obviously this is an oversimplification and done for explanation purposes

Fig. 4. (a) The deployment of capabilities before the update. (b) The deployment of capabilities on the phone, after the update.

Upon entering the conference, the user wishes to use "MPLAYER" to view the stream. He/she has the application find any available streams. "MPLAYER" queries all discovery services ("CENTRALADV" in this case), for the existence of any "STREAM" capability. It is not found and thus "MPLAYER" uses "CENTRALADV" to query for the existence of any other advertising service, currently in reach. "CENTRALADV" shows that there is an discovery service currently in reach, "MULTICASTADV". However, the mobile phone does not have the "MULTICASTADV" capability so it requests it from the mobile phone operator using the local LMDC. The phone operator sends an LMU containing "MULTICASTADV", a digital signature verifying the code's validity, and specifies the "CORE" as the deployment target, charging the user accordingly. The LMU also contains an unpacker, that deploys and initialises the "MULTICASTADV" on the local core. The LMDC receives the LMU, and notifies the target ("CORE"). It accepts it, seeing that the source is the network operator and the unpacker deploys it. MPLAYER initialises "MULTICASTADV" with the information received from its advertisable message, and has it listen to the local multicast advertising group. "MULTICASTADV" notifies "MPLAYER" of the existence of the "STREAM" capability. "MPLAYER" then presents the description of the capability to the user, who decides that it is the stream he/she wants. "MPLAYER" analyses the advertising message of "STREAM" and realises that it does not have the theora codec. However, the codec, encapsulated by the advertisable capability "THEORA" is available from the conference's computer. "MPLAYER" uses the local LMDC again to request capability "THEORA" from the computer. The computer's LMDC packages "THEORA" into an LMU and sends it to be deployed at "MPLAYER". The phone's LMDC receives "THEORA" and asks the network operator to verify the validity of the code. The operator verifies this and again charges the user. The LMDC then deploys the codec to "MPLAYER" which can now display the stream.

6 Conclusion and Future Work

The use of LM techniques in traditional systems and networks is well understood and has been extensively used. As such, its advantages are well known.

The novelty of our approach, is that we provide the ability to build adaptable applications by providing the flexible use of LM techniques, using an architecture that caters for the heterogeneity that this computing scenario entails. This differs from other approaches in that we provide a complete architecture for mobile computing applications together with an engineering approach, that is specifically geared for mobile applications that can use LM techniques and we do not place any limitations in the use of those techniques. We have showed the flexibility of the approach in allowing the development of adaptable and context-aware applications, as well as dynamically deploying functionality when needed. We believe that SATIN can be used to build a new class of self-organising systems and self-healing and context-aware mobile applications. The advantages of this flexibility are many: They include improvements in ease of use, better use of limited local and peer resources, more efficient use of the network resources etc. We have implemented SATIN in Java and have been able to run preliminary tests on laptops and PDAs. The current build weighs at around 100KB, including a number of capabilities. We also have some preliminary numbers: The middleware and a sample application takes 1155KB of heap memory and transfer and installation of a single capability takes 1452ms, on an unoptimised build over Ethernet. We plan on developing more applications based on our approach that promote interoperability exhibit dynamic behaviour as well as to better evaluate performance and scalability, using a number of devices as well as simulation.

Acknowledgements. We would like to acknowledge Licia Capra for her help and comments on a draft to this paper. This work is being sponsored by EPSRC grant number R70460.

References

1. Palmsource developers program. http://www.palmsource.com/developers/.
2. K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini[tm] Specification*. Addison-Wesley, 1999.
3. L. Capra, G. S. Blair, C. Mascolo, W. Emmerich, and P. Grace. Exploiting reflection in mobile computing middleware. *ACM SIGMOBILE Mobile Computing and Communications Review*, 1(2).
4. G. Cugola and G. Picco. Peer-to-peer for collaborative applications. In *Proceedings of International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, July 2002.
5. A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Trans. on Software Engineering*, 24(5).
6. C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Int. Journal on Personal and Wireless Communications*, April 2002.
7. Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, May 2001.
8. M. Roman, F. Kon, and R. H. Campbell. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware*, July 2001.