Department of Computer Science
University College London
University of London

Use of Logical Mobility for Mobile Self-Organisation

Stefanos Zachariadis
s.zachariadis@cs.ucl.ac.uk

Transfer Viva Report

November 2003

Abstract

Mobile systems are an extreme instance of highly dynamic distributed systems; mobile applications are typically hosted by resource-constrained environments and may have to dynamically reorganise in response to unforeseeable changes of user needs, to the heterogeneity and connectivity challenges from the computational environment, as well as to changes to the execution context and physical environment. Current mobile application development features static and monolithic applications - making them suitable for a fixed execution context but very limited for mobile devices. We argue that mobile computing systems can benefit from the use of self-organisation primitives. In particular, we argue that the application of logical mobility primitives, such as remote evaluation, code on demand and mobile agents as well as the componentisation of the system assist in building self-organising mobile systems. We show that a component model is required by the application of logical mobility primitives for self-organisation and discuss one. We also show how the primitives can be made available to application programmers using a light-weight component-based middleware and discuss SATIN (Self Adaptation Targeting Integrated Networks), a prototype component-based middleware system implementing these primitives. We also discuss the extent to which SATIN supports the development of self-organising systems, by building a deployment system that manages the dynamic installation and update of components on mobile hosts.

Contents

Structure of this report

The structure of this report corresponds to the structure we plan to give to our PhD thesis. There are six main chapters and three appendices all of which are subject to change. The appendices present our proposed thesis table of contents, the plan for completion and our list of publications thus far.

We indicate which sections are work in progress at the beginning of each chapter.

Chapter 1

Introduction, Problem Statement and Research Contribution

The recent advances in mobile computing hardware, such as laptop computers, personal digital assistants (PDAs) and mobile phones, as well as in wireless networking (UMTS, Bluetooth [Met99] and 802.11 [V. 96]) deliver sophisticated mobile computing platforms. We are observing a further and rapid decentralisation of computing, with computers becoming increasingly capable, cheaper, mobile and even fashionable personal items, which are exposed to a dynamic context and can wirelessly connect to information through variable networks.

Distributed applications deployed on these platforms, however, may be exposed to considerable challenges from their execution context. One of the main peculiarities of this environment is heterogeneity. This heterogeneity occurs in software, hardware and network protocols, particularly when mobile devices form distributed systems in an ad-hoc manner. Mobile systems are equipped a large number of different applications, use different operating systems and middleware, and often have more than one network interface. A further challenge is considerable variation in the computational resources available, such as battery power, CPU speed, volatile and persistent memory and network bandwidth.

Thus, mobile computing systems are an instance of *highly dynamic systems*. They dynamically form networks of various different topologies, they are heterogeneous and resource constrained and are exposed to a dynamic environment. Moreover, the requirements for applications deployed on a mobile device are a moving target: The context in which a mobile application is running can be highly dynamic; Changes in the environment may dictate changes to the requirements of an application; it may need to integrate with a new service for example.

However, the current state-of-practise for developing software for mobile systems offers

little to no flexibility to accommodate this heterogeneity and variation, forcing application developers to decide at design time what possible uses their applications will have throughout their lifetime. In fact, the current industry state of the art features static and monolithic applications which are more suitable for a fixed execution context rather than for a mobile device.

We argue that more suitable solutions would be required to automatically adapt to changes in the environment and to the users' needs. Power [Pow90] argued more than a decade ago that it is common in distributed systems that

> "when something unanticipated happens in the environment, such as changing user requirements and/or resources, the goals may appear to change. When this occurs the system lends itself to the biological metaphor in that the system entities and their relationships need to self-organise in order to accommodate the new requirements."

A self-organising system [BCB+02, BCB+02, GJK02, FH02, Sha02] is *a system that adapts to accommodate to changes to its requirements.* Using this definition, we can draw parallels between a mobile system and the requirement for self-organisation: A mobile system is a system that, by definition, will encounter changes to its requirements. We therefore argue that mobile systems could benefit from the use of self-organisational primitives.

There are various issues with self-organisation, particularly when it is applied to mobile systems. As mentioned before, one of the main peculiarities of this setting is heterogeneity, in the software, hardware and network 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. The network connectivity provided is highly variable and unreliable, with devices facing temporary and unannounced loss of it. Devices also have limited resources compared to desktop machines, discover hosts in reach in an ad-hoc manner etc. The current mobile industry state-of-the-art features monolithic applications which offer little to no interoperability, forcing application developers to decide at the design stage what possible uses their device will have throughout its lifetime and deploy them statically. We argue that more suitable solutions would be required to automatically adapt to changes in the environment and to the users' needs. On the other hand, the literature on self-organising systems largely focuses on the application of genetic algorithms, expert and agent-based systems [Pow90, PB01, MRS02]. These approaches tend to be rather heavyweight and appear unsuitable for mobile systems, that are, by orders of magnitude, more resource-scarce than the fixed systems for which these self-organisation primitives have been devised.

This work is an investigation into the use self-organising primitives by mobile systems. In particular, we argue that Logical Mobility techniques can assist in building mobile self-organising systems. Logical Mobility is defined as the ability to ship part of an application

or even migrate a complete process from one host to another. The increasing popularity of the Java programming language [Sun95] and environment has largely been correlated with the acceptance of logical mobility techniques, due to the inherent code mobility infrastructure that Java provides. 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 [RMI98], Corba [OMG95], etc), to distribute expensive computations (Distributed.net [dis]) etc. Note that LM is a superset of code mobility: Whereas code mobility specifically concerns object code, logical mobility includes the transfer of any application part, including application data and metadata.

Middleware has traditionally been used to enhance the development of software so that heterogeneity and distribution can be handled at a more general level and application development can be simplified [MCE02]. As such, a number of research efforts have resulted in mobile middleware systems that use logical mobility primitives. However, we believe and show further on, that these systems offer very limited use of logical mobility primitives, usually a subset that is used to solve a specific problem.

1.1 Scope of the Work

In order to clarify the scope of the thesis, this section points out explicitly what we aim to achieve and what we do not.

We claim that mobile computing systems are an extreme instance of highly dynamic systems. Mobile computing systems, by definition, experience changes and fluctuations to their environment and execution context as well as to their limited resources (memory, network connectivity etc.) As such, the requirements of a mobile system can rapidly change following these changes. Given this, one of the major limitations of current mobile applications is their rigidity. The industry offers monolithic applications, or statically deployed logic. These are systems do not adapt to their environment or interoperate with others. We define a monolithic application as an application the behaviour of which is defined a priori by the developer. Thus, a monolithic application features no components that can be changed at runtime. An *adaptive*, *dynamic* or *self-organising* application on the other hand, is one that adapts its behaviour to changes to the context [1] and the user's needs, or more generally, to changes to the requirements. We therefore claim that mobile systems can benefit from the principles of self-organisation.

The novel contribution of this work is an investigation of primitives for self-organisation in mobile systems. The literature on self-organising systems largely focuses on the application

[1] By context, we mean everything that can influence the behaviour of the application, from availability of other mobile and stationary hosts (host A is *in reach* with host B), to the status of local resources, all the way through to the availability or lack thereof of remote resources and applications.

of genetic algorithms, expert and agent-based systems [Pow90, PB01, MRS02]. These approaches tend to be rather heavyweight and appear unsuitable for mobile systems, that are, by orders of magnitude, more resource-scarce than the fixed systems for which these self-organisation primitives have been devised. We argue that the principles of Logical Mobility and Componentisation can assist in building self-organising mobile systems. In particular, we claim that they tackle the issues of heterogeneity and system monolithism. More specifically, this thesis investigates:

- **Logical Mobility for Mobile Self-Organisation.**
  Logical Mobility is defined as the ability to send part of an application or even migrate a complete process from one processing environment (or host), to another. This has been further classified into paradigms [FPV98]. Although its use has been well established in legacy distributed systems [set, RMI98, OMG95], there has been little investigation on the use of these primitives in a mobile environment with the peculiarities that this entails (heterogeneity, ad-hoc networking, etc). We believe that logical mobility can assist in building mobile self-organising systems, as it can tackle software heterogeneity and better utilise the limited available resources. We investigate and tackle sending and receiving, discovery and advertising as well as identifying and grouping elements of Logical Mobility.

- **Component-Based Mobile Systems.**
  We investigate representing the mobile system (applications & system libraries) as a collection of components. Representing them as a set of collaborating components, we can potentially change individual ones to tailor the overall functionality and behaviour of the system. This allows for a limited functional set of a system to be installed to a device and customised when and if needed. This is compared and contrasted to the more traditional approach of monolithic applications. We show how Component-Based systems tackle the issues of monolithism and assist in building mobile self-organising systems. The necessity of using a component-based approach in order to use Logical Mobility primitives will also be demonstrated.

- **Mobile Computing Middleware System.**
  As middleware systems have been traditionally used to handle issues that arise from application development in a more general and systematic way, a number of middleware systems have attempted to make use of logical mobility primitives to gain some of the advantages inherent in this mechanism. However, the use of LM has been very limited, as middleware systems that use it either take the "black box" route, by employing the use of logical mobility primitives internally and not exposing them to applications, or, alternatively, they expose a very limited subset. The transparency of usage in the first approach may be suitable in fixed distributed systems, in which the execution context is essentially static. However, transparency in mobile systems has been argued [CMZE01] to be disadvantageous to mobile applications. Moreover,

the exposure of a limited subset of logical mobility primitives as taken by the second approach, has led to the development of a plethora of incompatible mobile computing middleware systems [CP02, MPR01, WBS02], each one solving a particular and thus limited problem.

We investigate the design and develop a mobile computing middleware system, based on the principle of componentisation, that allows for the development of self-organising applications by allowing any component the use of any logical mobility primitive.

Although related, the following are considered outside the scope of this thesis:

- **Security.** Security in a mobile distributed system is a complex issue, especially when the system allows runtime adaptability of applications. We believe that addressing security issues would require extensive research efforts, and would be worth another PhD. We therefore leave this problem open and mention that there are other projects [KRL+00] working on it. We do provide an infrastructure that can be used to provide security, but this is considered to be beyond the scope of this work.
- **Context Inspection and Reflection.** This work assumes that we are able to inspect the current context and expose it to applications. Although closely related to our work, we do not investigate this mechanism further, as other groups [CBM+02] are researching this.
- **Paradigm Evaluation.** Our work aims to allow dynamic applications by flexibly exposing any logical mobility paradigm. However, one paradigm may be more effective than another, depending on a number of runtime parameters and the application itself. We do not investigate how to evaluate the performance of various paradigms and how to help the developer choose which one to use. We leave this as an open issue and mention that there are other projects [GM02] that try to tackle this.
- **Other Issues with Self-Organisation.** This thesis aims to specifically tackle heterogeneity and monolithism. Other issues with Self-Organisation, such as constraints are not tackled in this treatise. We mention however that there are other approaches [GJK02] that try to tackle these issues.

The use of Logical Mobility to enable self-organising mobile applications is a long-term research issue that will not be solved by hardware advances alone. The prerequisites for a new class of mobile applications have been satisfied and this work investigates the use of Logical Mobility and componentisation to achieve this. This is an open research issue that remains largely unexplored, with little, limited and ad-hoc work already done. We believe that this problem is feasible to be tackled in a three-year period of time and believe there is scope for a PhD in the three points outlined above.

1.2 Hypothesis Statement and Contribution

Mobile computing systems are an instance of highly dynamic systems. As such, the requirements for a mobile system are a moving target, following changes to the environment and execution context. Given this, monolithic applications which feature no interaction and are thus primarily designed to be executed in a static context are not representative of the potential of the ubiquity of new mobile devices and networks and new design principles have to be investigated. Our hypothesis is that these systems can benefit from the usage of self-organising primitives. In particular, we argue that logical mobility and componentisation can be successfully exploited, through a mobile computing middleware, to assist in building self-organising mobile applications, solving two main problems with mobile self-organisation: monolithism and heterogeneity.

Our expected contributions can be summarised as follows:

1. **Use of Logical Mobility for Mobile Self-Organisation.** We investigate how to use Logical Mobility in a mobile environment and how it can be used to offer self-organising applications. This includes grouping, sending and receiving and implementing paradigms, through a mobile middleware system.
2. **Component Model for Self-Organisation.** We explain the necessity of a component model for self-organisation, identify the requirements for it and present a design that satisfies those requirements.
3. **Middleware Design and Implementation.** We have designed and implemented SATIN (Self Adaptation Targeting Integrated Networks), a mobile middleware system that enforces our ideas of modularity and allows the flexible use of logical mobility by applications. We plan to release the middleware as open source.
4. **Evaluation of Results.** We evaluate our component model, our approach to logical mobility as well as our middleware system.

Chapter 2

Background and Motivation

-Work in Progress-

This chapter gives an introduction to Self-Organisation, Logical Mobility and Mobile Computing principles, as well as to component models. It also gives the motivation for our work, by presenting a number of case studies which the limitations of current mobile applications.

2.1 Self-Organisation, Distributed Systems and Physical Mobility

Traditionally, software has been designed and implemented for a fixed set of requirements. More recently, extension mechanisms such as scripting languages [CLP00] have been used to allow extending the functionality of software. However, the behaviour of such an extension is constrained by the limits of the scripting language and the requirements of the software; radical changes to the way the software system works are not allowed.

Legacy, fixed computer systems, execute applications in a static environment with little to no interaction between different software systems. With the advent of communication techniques, it has been possible to build distributed systems where computers and as an extension the software systems that are built on top of them are increasingly interconnected and communicate with one another. Distributed systems are becoming increasingly complex - from a small number of computers interconnected with a Local Area Network, distributed systems now scale to thousands of devices, interconnected over Wide Area Networks; some of them even include wireless links. Thus, the complex behaviour of a distributed system, emerges from the relatively simple interaction between its various components. The requirements of distributed software systems can potentially become more

difficult to specify: A distributed environment can be subject to change and changes to the environment can result in changes to the set of requirements. Moreover, the structure of a distributed system can be very dynamic - Nodes in the system can come and go and with the increasing variety of machines available, we can end up with a system composed of heterogeneous hardware and software as well as networking links.

To cope with such changes, a distributed system needs to *adapt*. More recently, terms borrowed from biological sciences have been borrowed to describe this behaviour: A distributed system needs to *self-heal* or *self-organise* in order to adapt to changes in its execution context. A self-organising software system is a system that automatically reconfigures itself in order to accommodate to changes to its requirements.

More recently, we have witnessed a new class of distributed systems, mobile distributed systems. With the increasing popularity of mobile devices such as PDAs, laptops and cellphones, and with the increasing abilities and resources of these devices, we now have the potential for a ubiquitous and highly dynamic distributed system, composed of both mobile and stationary hosts, connected using the full spectrum of networking hardware available. The potential for adaptation in the mobile environment is great: In addition to the issues that traditional distributed systems face, a mobile application is expected to operate in a dynamic context and environment: A user actively moves his/her mobile device and applications to different environments, where different types of networking connectivity and/or services are available. This, usually, cannot be defined a priori. As such, the set of requirements for a mobile application changes and hence the need to self-organise in order to accommodate the requirements becomes evident. Thus, mobile computing systems are *extreme instances of highly dynamic systems*. We argue that the principles of self-organisation can assist in tackling this dynamicity.

We identify six main problems with mobile self-organisation:

- **Heterogeneity.** The problem of heterogeneous machines of legacy distributed systems is more evident with mobile devices. We identify three types of heterogeneity:
  - **Network Heterogeneity.** As will be demonstrated in Section 2.2.2, mobile devices can connect to a network using various different hardware and topologies, with vastly different characteristics.
  - **Physical Heterogeneity.** The devices that form a mobile distributed system are composed with vastly different hardware.
  - **Logical Heterogeneity.** The devices often ship with different software and middleware systems.

  Heterogeneity makes it difficult to adapt to changes: For example it is difficult to discover services that are being offered, when using different advertising and

discovery techniques. It is also difficult to interoperate with different applications: Playing an audio file encoded with an unknown codec is an example of this.

- **Monolithism.** Mobile systems are dominated by monolithic applications. A monolithic application cannot adapt as its behaviour which is defined a priori. Statically deployed, a monolithic application offers little scope for customisation (perhaps through a scripting language as mentioned above) and needs to be replaced in order to alter its behaviour to something outside its initial requirements. The majority of mobile applications are monolithic and there is no infrastructure provided for modular applications.

- **Constraints.** It is difficult to define the constraints of self organisation - the limits of how much an software system can change. We do not tackle this issue, but mention that there are other groups [GJK02] trying to.

- **Trigger.** There is also the issue of what is the source of the self-organisation: Is it the environment that can trigger the evolution of a mobile device, or is it the device that actively responds to changes or requests to and from its environment? Again, we do not tackle this issue.

- **Context Awareness.** In order to adapt to changes to the environment it is important to be able to detect changes to it. This treatise considers only service advertisement and discovery. We mention other groups that target context-aware applications specifically [AAH+97, LKAA96, CEM01, FS01].

- **Security.** It is difficult to guaranty security in a mobile distributed system. We do not treat this problem, but mention some possible solutions as future work (see Section 6).

2.2 Introduction to Logical and Physical Mobility

2.2.1 Logical Mobility

This section builds and expands on [FPV98].

Logical mobility is defined as the ability to move parts of an application or migrate a complete process from one processing environment to another. It is a superset of *code mobility*, in that Information transfered can include binary code, compiled for a specific architecture, interpreted code, bytecode compiled for a virtual platform, such as the Java Virtual Machine (JVM) but also application data such as profiles, remote procedure calls etc.. We define an *execution unit*, as a collection of any of the above. As such, using Logical Mobility can be represented as composing the execution unit and transferring it

from one processing environment to another. The processing environments can range from different machines, to sandbox processes residing on the same host.

Code Mobility has been described using the *code pushing* and *code pulling* model, which is used to identify who sends the unit of code in a transfer. In code pushing, a host sends a unit of code to a remote host. In code pulling a host requests and receives a unit of code from a remote host. We extend this concept to Logical Mobility and *execution units*.

The use of LM has been classified into a set of paradigms:

Client - Server

Client - Server (CS) dictates the execution of a unit in a server, triggered by the request of 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). In this case, the execution unit is formed by the client and the execution unit is located at the server. The transfer of the execution unit is handled by the client, that *pushes* the request to the server.

Note that Client - Server is mostly used in traditional distributed systems.

Remote Evaluation

Remote Evaluation (REV) describes a host sending or pushing a particular unit to be executed remotely. A result may or may not be needed, depending on the application. The unit is usually executed by a particular application or processing environment. If we try to apply traditional distributed computing models, we can consider the machine prepares and sends a an execution unit as a server and the hosting environment that executes it as a client. In this sense, REV can be considered a reversal of the CS paradigm.

This paradigm is employed by large distributed computing projects, where very large computations can be split into manageable parts by a server and sent to a large number of clients. Examples of those include SETI@home [set] and Distributed.NET [dis].

Code on Demand

In the code on demand (COD) paradigm, a host requests a particular unit, which is then shipped locally and can be used. This is an example of dynamic code update, whereby a host or application can update its libraries and available codebase at runtime. In this case, the execution unit is prepared and transfered by the host that receives the request and it is executed by the client that makes the request. The host requesting the unit thus pulls it from the remote location.

Many examples of COD have recently emerged, due to the popularity of Java and its built-in class loading mechanism and object serialisation framework[Sun98]. Java applets and Jini [Wal99] are examples of the use of this paradigm.

Mobile Agents

A Mobile Agent[WPM99] (MA) is an autonomous execution unit. It is injected into the network, to perform some tasks on behalf of the user or an application. The agent can potentially migrate from a processing environment or host to another. In this case, the execution unit is prepared by the computer injecting or pushing the agent into the network. The agent can be hosted in various environments (usually sandboxed) and depending on the implementation, either the agent or the hosting environment itself handle transferring the MA to another host. The main difference between REV and MAs, is that the MA has the ability to migrate from host to host.

There are a number of Mobile Agent platforms, such as Aglets [LO98] and μCode [Pic98]. There have been few mobile middleware systems that employ them - For example LIME [MPR01] uses μCode to offer data-sharing in a mobile environment, using Mobile Agents. There has been extensive work in the use of MAs in network management [BPW98, BGP97, GGGO99].

2.2.2 Physical Mobility

Portable computing devices, such as laptops or high-end PDAs, can be equipped with a variety of networking hardware; this implies that throughout its lifespan, a device will be connected to different types of networks with different characteristics and configurations, in terms of bandwidth, reliability, cost and routing.

This section continues with a brief analysis of different types of mobile networking.

Nomadic Computing

This paradigm describes users connecting to the network temporarily, from various different (usually stationary) locations. Connectivity is therefore an exception and lasts for as long as users stay in the particular location. Thus, applications cannot be connected to the network and be physically mobile at the same time. Examples of this type of mobility include wired networks like Ethernet, and dial up networks.

Very pervasive in today's corporate culture, nomadic computing usually offers high speed access to either corporate networks or the Internet. It is often the case that in such

configurations, firewalls enforce strict access rules, limiting the ability of devices to send information to some ports, use UDP, or even accept incoming connections. Thus, the device's ability to access information and exploit the network resources is therefore limited. However, such connections are usually low cost, meaning that users do not pay for the time they are connected or for the data that they download. Laptop computers are often connected to such a network, using other networking mobility paradigms to connect when being mobile.

The characteristic of this type of mobility is that it allows users to access a fixed network (the Internet or an intranet) with reasonable speed where routing is handled by the network infrastructure; assumptions that are not always valid with other types of mobility as is shown later on.

Base Station Mobility

This type of mobility describes users that connect to a fixed network (such as the Internet) while still being physically mobile within a region where there is coverage. Exploiting wireless links to connect to a fixed infrastructure network, base station mobility offers an error prone, low-bandwidth and continuous (provided there is coverage) connection. Examples include using GSM/GPRS mobile phones or modems, with PDAs or laptop computers. This type of connection is usually expensive: users pay either for the time they are connected (e.g. GSM) or for the the data that they exchange (e.g. GSM/GPRS).

The characteristic of this type of mobility is that it allows the user to be constantly connected to a fixed infrastructure network while being mobile. The wireless communication link is usually low bandwidth, expensive, and error prone. Routing is handled by the network's infrastructure.

Ad-Hoc Networking

The final type of mobility considered is Ad-Hoc networking. It usually has no fixed infrastructure and hosts (mostly) employ wireless connections to allow a host to connect to other devices and hosts which are within the range of the wireless networking hardware (*in reach*). In order to contact a host which is not currently in reach, other hosts need to act as routers; otherwise, no routing is necessary and communication is always single hop. The networks formed are thus very dynamic and rapidly changing. The communication link is usually of relatively low speed and is also error prone. However, as there is no centralised provider, connectivity is essentially free, costing mobile users only in terms of battery life.

Examples of networking hardware that supports Ad-Hoc networking include 802.11b com-

pliant cards, Bluetooth chips and infrared ports. Ad-Hoc networks are currently actively being researched on and appear to be particularly useful in disaster scenarios, where they might even be the only feasible networking choice. However they are also frequently used in scenarios where peers want to communicate with each other without accessing any sort of centralised provider or pay-per-use network.

The characteristic of mobility based on ad hoc networking is that it allows devices to spontaneously form completely unstructured highly dynamic and short-lived networks with other peers that are in reach, using inexpensive wireless links offering error prone and low bandwidth connectivity. Routing is either abolished or performed by the nodes of the network themselves.

We feel that given the current trend in mobile technologies, most mobile devices will end up using a number of mobile networking paradigms, depending on the situation. This is certainly feasible using the networking hardware that devices already ship with and is also the logical conclusion for users that wish to access a centralised network (the Internet) when being on the move (base station mobility), want to have cheap connectivity with their peers (ad-hoc) and also need connect to the centralised network while being static (nomadic computing).

We believe that the use of logical mobility primitives naturally lends itself to supporting self-organisation as it allows for adaptability. In particular:

- Logical mobility allows applications to update their codebase/components, thus acquiring new abilities;
- Logical mobility permits interoperability with remote applications and environments, which were not envisioned at design time;
- Logical mobility achieves the efficient use of peer resources, as computationally expensive calculations can be offloaded to the environment;
- Logical mobility facilitates the efficient use of local resources, as infrequently used functionality can be removed to free some of the limited memory that mobile devices feature, to be retrieved later when needed.

2.3 Component-Based Systems

This section provides a very brief introduction to component based systems. For more details we refer the reader to [Emm00, OMG95, Gri97, RMI98].

The Corba Object Model

Corba is an open standard, that allows applications to be distributed between multiple hosts. Corba's core component is the Object Request Broker or ORB. The ORB is responsible for delivering the request of a client object to the server object and then returning the result back to the client object.

Distributed Component Object Model

Microsoft's Distributed Component Object Model (DCOM) [EE98], is an extension to the Component Object Model (COM), which adds a security model and allows for applications to be built using multiple COM objects that reside in multiple hosts.

Java Remote Method Invocation

Java Remote Method Invocation (RMI) enables an object that resides in one Java virtual machine to invoke a method from a remote object, residing in a different virtual machine. RMI relies in a registry, registering objects which can be accessed remotely.

Universal Interoperable Core

UIC [RKC01], another is a generic request broker that defines a skeleton of abstract components which have to be specialised to the particular properties of each middleware platform the device wishes to interact with.

ReMMoC

ReMMoC [P. 03] is 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.

2.4 Motivating Examples

2.4.1 Case Study: Industry State-of-the-Art mobile application development

This section presents how applications are developed on the most popular portable platform, the PalmOS [pal].

The PalmOS platform is the most popular operating system for PDAs. It runs on more than 30 million devices worldwide, including mobile phones, GPS receivers, PDAs and sub notebooks. The latest devices at the time of writing have 64MB of RAM (which is

used both as storage and heap), Bluetooth, 802.11b, Infrared and wired serial networking interfaces, as well as a 400MHz ARM processor.

The current version of PalmOS allows for the creation of event-driven, single-threaded applications. All files (applications and data) are stored in main memory. Each file is a database, or a collection of records, as the devices lack a filesystem. Developers compile an application into a single Palm Resource File (PRC) and application data can be stored in Palm Databases (PDBs). The operating system allows for limited use of libraries. Applications are identified by a unique 4 byte identifier, the Creator ID, which is stored at a central database. As such, any developer must create a unique Creator ID for each individual application and register it with the company that develops the operating system. All PRCs and PDBs include the Creator ID, which is used by the OS to identify applications and the data associated with them.

A PalmOS device usually ships with personal information management (PIM) applications installed. Installing new applications requires either locating a desktop computer and performing the installation there or having the application sent by another device directly, a procedure which is not automated. Statistics show that users don't usually install any 3rd party applications, even though there is a plethora available.

There are various disadvantages to this model: There's very little code sharing between applications running on the same device. There is no interoperability framework for applications running on different devices. Applications are monolithic, being composed of a single PRC, which makes it impossible to update part of an application. The fact that users do not install any 3rd party applications has been attributed to the fact that it is difficult to do so.

Devices such as these have the ability to be deployed in both a nomadic and ad-hoc networking setting. The potential for interaction with their environment is great, however PalmOS does not provide any mechanism to do so. As such, these devices remain largely underutilised, simply because they execute legacy applications that do not take advantage of their hosts' mobility.

2.4.2 Case Study: Application Deployment and Update in a Mobile Environment

This section presents a realistic scenario, against which we plan to evaluate our approach. In this case study, mobile devices are equipped with two different networking interfaces, one allowing them to form *ad-hoc networks* with peers in reach and another allowing them to *nomadically* connect to a backbone network. Consider Bluetooth or 802.11b as an example of the first interface and a GSM modem as an example of the second. Given the current market penetration of mobile phones and PDAs, this is a very common

Figure 2.1: (a) Deploying applications (ring tones & games) to mobile phones. (b) Deploying & maintaining applications from peers: Dotted lines represent certificate downloads. Solid lines represent update downloads.

scenario. Statistics show that most users rarely install any new applications to those devices. Moreover, current mobile applications tend to feature very little interaction with each other and not to react to events and changes to their context. There have been some approaches that promote interoperability between applications running on mobile phones allowing for data synchronisation, such as SyncML-based approaches [Syn00], Microsoft activesync, or PalmOS HotSync [pal]; however, such approaches only tackle limited aspects of interoperability, as they only target synchronisation of information between mupltiple devices.

Installing new applications and updating existing ones is currently difficult. As a matter of fact, the only popular update to mobile phones is the download of ring tones and games. The source of the download is usually the network operator, and the cellular bandwidth, which is very expensive for both the user and the operator, is used for the transfer. Figure 2.1(a) shows how this is done. Even though the phones could communicate with each other directly using Bluetooth, they instead all download the application from the network operator using the cellular network. This approach is very limiting and the infrastructure used cannot be expanded to tackle the heterogeneity and context awareness. Thus, the rich resources and ad-hoc connectivity that these devices are equipped with are not exploited and users experience very little context interaction. Figure 2.1(b) shows the self-organisation approach to the same problem: The devices download the updates from each other, when feasible, and only need to interact with the network operator to get a certificate of authenticity.

We believe the mobile application market to be vastly untapped in this sense, because of the difficulty of installing and maintaining new applications. Moreover, the connectivity and interaction potential of these devices, that could allow a whole new class of self-organising applications to be built, is largely underutilised.

An self-organising approach to application deployment and maintenance would offer significant advantages:

- Discovery and retrieval of new functionality that is needed by an application. For example, a media player would be able to download a codec when needed to play a

particular file.

- Transparent discovery and installation of new updates from peers. A trusted host from the centralised network could be used to verify the authenticity of the updates.
- Installation of new applications from peers, with the trusted host again verifying the authenticity of the applications.
- Removal of functionality when infrequently used. The functionality could be transparently retrieved from peers or the centralised host when again needed.

2.5 Limitations of Related Work

Despite the evident suitability of logical mobility to the dynamicity of a mobile computing environment, its use to support self-organisation has been very limited. Current efforts can be grouped into two categories: Approaches that use logical mobility 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 specific paradigms of logical mobility to provide particular functionality to applications. Examples of the first category include ReMMoC [P. 03], 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 [RKC01], 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 [MPR01], PeerWare [CP02] and Jini [AOS+99]. 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 logical mobility 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 logical mobility to download code allowing them to utilise services that are already being offered.

The problem we find with approaches in both categories is that their use of logical mobility is limited to solving specific problems of a limited scope, such as middleware reconfiguration, or data-sharing.

This work investigates the use of logical mobility primitives for self-organising mobile applications, a class of applications which we believe offer substantial advantages over traditional ones.

Chapter 3

Primitives for Self-Organisation in Mobile Systems

-Work in Progress-

This chapter identifies the primitives we propose for self-organisation in Mobile systems.

3.1 Component Model

3.1.1 The necessity of componentisation

We consider component systems to be a key prerequisite for self-organisation and for the use of logical mobility in a mobile environment. Componentisation is primarily a result of object orientation techniques and replaces traditional monolithic software, with a collection of interoperable components, each of which performs a specific function. Component based systems are the opposite of monolithic systems: Whereas a monolithic system is effectively one large component, a component-based one splits the functionality into a series of potentially reusable components. We consider componentisation necessary for the following reasons:

- It solves monolithism by promoting componentisation of applications.
- It encourages reusability, which is particularly important in the resource constrained setting of mobile devices. For example, a component that implements a compression algorithm can be reused by multiple applications.
- It splits the system into distinct components: Considering that we are targeting a

distributed environment where we specifically aim to send and receive parts of an application, component based systems naturally lend themselves as a coarse-grained guide to structuring the units that are sent and received.

- It allows for replacing a set of components of a system, seamlessly changing or *adapting* the overall behaviour of the parts of the system that use these components[1]. As such, provided that we can dynamically add, remove and replace modules, componentisation can allow, in principle, for the adaptation or self-organisation of a software system, by replacing its constituent modules.

Essentially, componentisation provides a structured and systematic way to split systems into distinct components: This behaviour is required, because it provides us with a way to tackle monolithism and it can be used as a basis to define what parts of an application can be sent and received around the network.

3.1.2 Requirements

We identify the following requirements for a mobile component system, that will be used with logical mobility in mind:

- **Middleware System Componentisation.** Middleware componentisation refers to the design and representation of a middleware system as a collection of components. As such, functionality which is considered part of the middleware, such as lookup and communications are components which can be interchanged, allowing the middleware to be reconfigured. This strategy has been followed by various systems, such as ReMMoC [P. 03] and UIC [RKC01]. This allows the middleware to reconfigure itself when in the vicinity of a particular system: For example, when encountering a Jini [AOS+99] system, the middleware can be reconfigured to use the Jini lookup service. Middleware componentisation is a mechanism that is particularly useful in tackling middleware heterogeneity.
- **Application Componentisation.** Similarly, application componentisation refers to the design and representation of an application as a collection of modules, each of which encapsulates specific functionality. Application componentisation aims for reusability of components, which can range from a GUI widget to a compression library, by different applications. The principle of componentisation can lead to coarse-grained application adaptability, by allowing for the exchange of one module with another, or the dynamic acquisition of another module. For example, desktop media players can acquire new codecs dynamically, when faced with a file they cannot decode.

[1] This assumes that the component to be changed uses the same programming interface with its replacement and that the latter functions correctly

We thus require both the middleware system used and the applications to be represented as components.

- **Identification.** We require each component to be uniquely identifiable. This allows us to distinguish different components in the heterogeneous distributed environment that we target, by using the appropriate identifier in any advertising and lookup mechanism. This mechanism also potentially allows for an application using a remotely received component without the component having advertised its programming interface. This assumes that the identifying mechanism can classify modules according to a functionality. Considering discovery modules, we can assume that a String-based identification mechanism will have a `DISC:` prefix, and that all advertising modules will have a `lookup()` method.
- **Dependencies.** Building on the identification mechanism as described above, we require each module to have a list of its dependencies. This allows us to build a dependency graph for each module and to be able to compute dynamically whether a host will be able to use a component.
- **Versioning.** We require each component to have a version. We allow for depending on a particular version of a component, as well as for having multiple versions of a component installed in a system.

We consider existing component models and distributed object middleware systems, as presented in Section 2.3 to be too heavy and unsuitable for our work, for the following reasons:

- We do not require location information that other systems provide. Our main target is not the active migration of components which potentially requires location information. It is also very difficult to provide location information for components running on a mobile distributed system, particularly because of the scale and very dynamic structure of the system, as well as the heterogeneity of the networking links used to form it.
- We do not require any specific communication primitives that other approaches provide.
- We do not wish to be tied down to a specific middleware system. We wish to provide an approach that allows for interoperability with such middleware systems.
- Most of these systems are very heavyweight to be used in a mobile device. For example, Orbix [TD98], a lightweight implementation of Corba uses 40 bytes for each reference.
- We cannot use a centralised server to register components as some of these approaches provide.

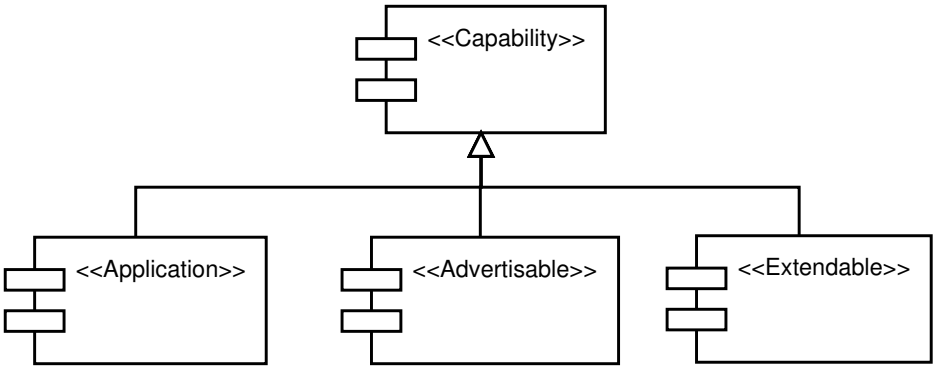

Figure 3.1: An illustration of our component archetypes and their relationship.

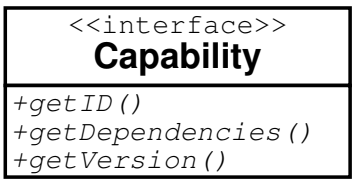

Figure 3.2: The basic interface of a capability

3.1.3 A Lightweight Component Model

This section describes our component model. To illustrate it better, we describe the design of a media player application in terms of our component model.

The basic component in our model is a *capability* (see figure 3.2). A capability includes metadata, including a version, an identifier and, using that identifier, a list of dependencies. The metadata set can be extended. Our media player for example, would be a collection of capabilities. At a coarse level, those would be the user interface, a codec and a codec repository. Notice that, even at this level, the componentisation of the application allows for reusability: Other applications can use the codecs that are installed. Moreover, in order to update the application, only the capability (the UI) would need to be replaced.

Using the basic concept of a capability, we define a limited meta-model composed essentially of a basic and very high level set of *capability subclasses*:

- **Application Capabilities.** An application capability flags a capability as an application to the middleware system. It includes methods to run, suspend and resume the application. The user interface of the media player would be the application component.
- **Extendable Capabilities.** An extendable capability is a capability that can receive logical mobility execution units from other hosts. The codec repository for example would be an extendable capability, provided that it can receive codecs from other

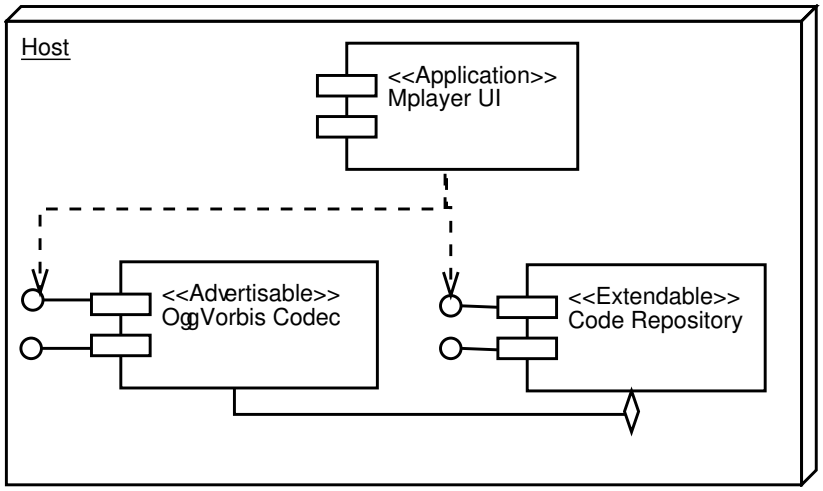

Figure 3.3: Deployment of a media player in a mobile node, using our component model.

hosts.

- **Advertisable Capabilities.** An advertisable capability is a capability the presence of which we want to advertise to other hosts. A service can be represented by an advertisable capability. In our media player example, a codec capability would be advertisable: We want to allow other hosts to retrieve the codec from the local host if it is needed.

An illustration of these classes of capabilities can be found in figure 3.1. Figure 3.3 illustrates a node with the media player deployed.

Our model allows for multiple inheritance: A component can be both an extendable and an advertisable capability for example. Moreover, it is extensible, as more types of capability stereotypes can be defined.

This model has various advantages, given our target setting, over those described in section 2.3.

- It is much more simple and lightweight [2], making it more suitable for resource constrained devices.
- It is extensible, unlike other models
- It does not require location information for each component (although it can be added).
- It does not require centralised registration of components.

[2]This is going to be evaluated in the future

- It does not define fixed communication primitives
- We do not need an interface description for each component - Thus, server and client side stubs are not needed.

3.2 Logical Mobility

When employing logical mobility techniques in support of self-organisation, we have identified a number of requirements that must be taken into account by any solution:

- The functionality provided by the middleware must be generic enough to be usable by any application and should not be tied down to a specific one. We cannot predict which application will require self-organisation, and it would be of no use to provide a specific infrastructure for self-organisation on a per-application basis. As such, the middleware should allow the use of any logical mobility paradigm by any application.
- The infrastructure provided should be built on the assumption that the mobile device will be able to access different types of networks and have different connectivity patterns. The device can also potentially encounter total lack of connectivity.
- In order for an application to self-organise in a given computational context, it should be able to probe it to discover what is currently available and to advertise the device's presence, functionalities and abilities. This allows interaction between applications running on different devices and reaction to changes in context. For example, if a service exists which streams a video file using a particular codec, a media player application running on a mobile device should be able to detect it and download the codec used if it is not installed on the device.
- The flexible use of logical mobility requires the ability to use any logical mobility paradigm described above, by any application. This implies the encapsulation of modern programming language abstractions such as objects, classes and RPCs into a container or execution unit, as defined in section 2.2.1. The container should also be able to optionally store the following metadata:
  - A digital signature which would allow hosts to verify its validity.
  - A class that would be able to deploy the contents to the recipient as it might not know how to utilise it.
  - Information on its size and an estimate on the size of its contents when deployed.
  - The dependencies of its contents.
  - The source and target of the items

  - A mechanism that would allow to add and retrieve arbitrary metadata to the container, to cater for applications that may need it.
  - An identifier, which would allow an application to uniquely identify each container. This is similar to the Corba [OMG95] object identifier concept.

  Thus, considering the different logical mobility paradigms again, client-server is represented as sending a request in a container, code on demand becomes requesting a particular container, remote evaluation is represented as sending a container to another host and mobile agents are essentially a container encapsulating a thread.

- The use of logical mobility as provided by a middleware system must be symmetrical; a host should be able to both provide send and receive containers of logical mobility elements. This allows for the creation of a large peer-to-peer network of mobile devices, which can use new resources and update their functionality by downloading them from each other.

  A number of related approaches, assume that mobile devices are thin-clients, with servers providing most functionality. Jini [Wal99] and its mobile implementation [HK01] assume that devices will be using services offered by servers, using COD and assume a centralised discovery infrastructure. FarGo-DA [WBS02] the version of FarGo [HBSG99] that is geared for mobile devices, allows for disconnected operations to services offered by remove servers by using CS and COD to replicate their functionality locally or send the requests to the servers when connectivity is restored.

- Applying logical mobility in a mobile computing environment is different from applying it to traditional distributed systems. In the latter case, 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 most suited, as mobile connectivity may potentially be fluctuating, expensive and error-prone. As such, any approach developed should cater for the extra dimension of asynchronous communication.

Chapter 4

SATIN: Design and Implementation

-Work in Progress-

This section describes how the principles outlined above are implemented in our prototype middleware system, SATIN.

4.1 Component Model Extension

We extend the model presented in Section 3.1.3, to include the following capability subclasses:

- **Advertiser Capabilities.** Advertiser capabilities implement advertising techniques.
- **Discovery Capabilities.** Similarly, discovery Capabilities implement discovery techniques.

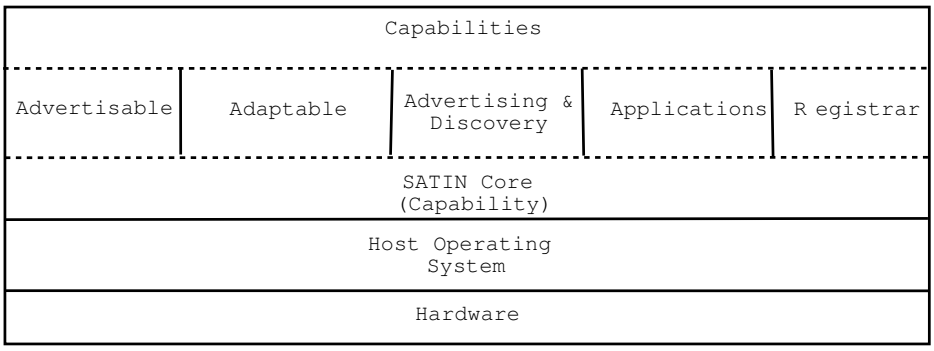

Figure 4.1: A high level view of a SATIN host. It is composed of capabilities, registered with a Core.

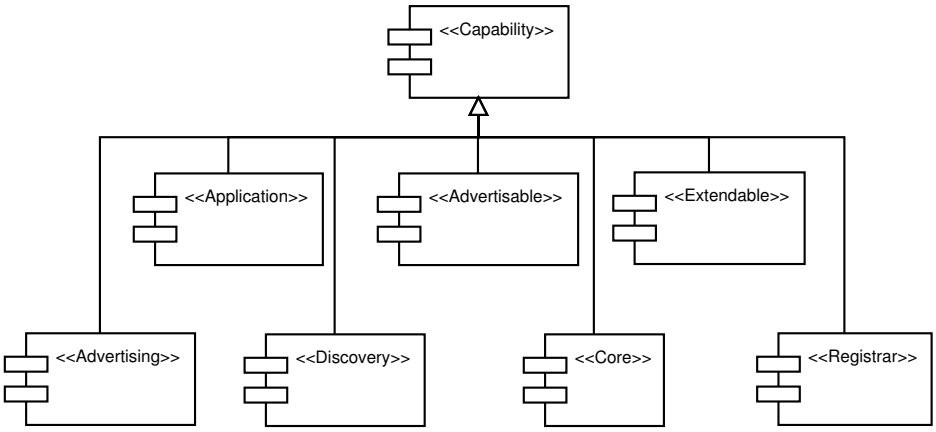

Figure 4.2: The complete SATIN metamodel.

- **Core Capabilities.** A Core capability is a registry of capabilities: Each host running SATIN, must have a Core.
- **Registrar Capabilities.** A Registrar capability is responsible for registering new capabilities with the Core.

Figure 4.2 shows the complete SATIN metamodel, including the extensions identified above.

All capabilities that are available to a particular host are registered with the host registry, or *Core*, which is also a capability. A reference to 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 is also the unique identification for every 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 [pal] applications have (i.e., a *globally* unique identifier). 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. The Core can also be an extendable capability, if the host is allowed to acquire new components at runtime.

4.1.1 Advertising & Discovery

An advertisable capability defines a text message that will be used to describe it. The message is encoded in XML. For example, let us assume a STREAM capability, which provides a video stream server facility that wants to be advertised. In this scenario, the capability message might be formatted as `<udp>4662</udp><codec>theora</codec>`, denoting that the server is waiting for requests at port 4662 and that the video codec is called

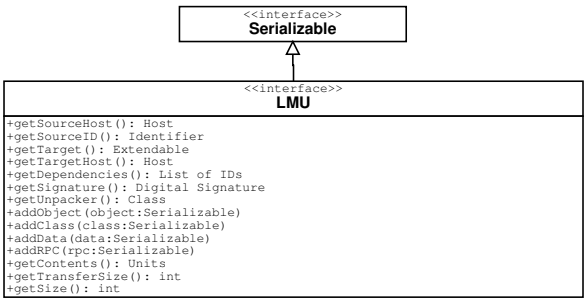

Figure 4.3: A SATIN Logical Mobility Unit.

theora. The importance of global identifiers is evident considering that if the STREAM capability is registered and has a global identifier, hosts which receive this message can decode its meaning, without requiring that STREAM also advertises a programming API.

This approach decouples the advertising message from the advertising and discovery mechanisms. Advertisable Capabilities can decide which advertiser is allowed to advertise them. An advertiser which is allowed to advertise a capability, adds some information to the advertising message regarding the advertisable capability itself and sends it over the network. For example the STREAM message given above would become `<capability id=''STREAM'' version=''0''> <udp>4662</udp> <codec>theora</codec> </capability>`. 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.

4.2 SATIN & Logical Mobility

The container used for encapsulating parts of an application is called a logical mobility unit (LMU) (see Figure 4.3). It allows for an arbitrary number of the programming constructs mentioned above to be encapsulated in a single unit, which can be digitally signed. An LMU can be sent by any capability that is registered with the local Core. Recipients of LMUs can range from the Core to any other capability present. Capabilities that can receive LMUs and are therefore *extendable* (see Figure 4.1). An LMU also encapsulates various metadata, such as which host and capability it was sent from, which host and capability is its target, the size the LMU takes when serialised, the size it takes when installed and the combined dependencies for the data encapsulated. The LMU can also include a special class called the *unpacker*, which is responsible for deploying the LMU in the target host, if the latter does not know how to operate it. The unpacker can also be

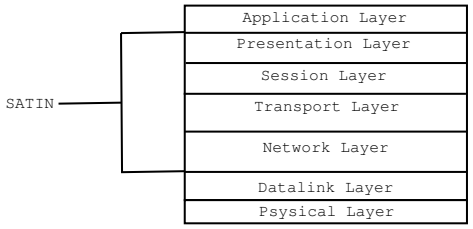

Figure 4.4: Our architecture in the context of the ISO/OSI networking model.

used to transfer threads, by resuming their execution.

A capability cannot send an LMU directly. The responsibility of sending, receiving and deploying capabilities is abstracted and handled by the *logical mobility Deployment Capability* (LMDC). The LMDC is a SATIN capability that manages requesting, creating, sending, receiving and deploying LMUs to the appropriate extensions. The LMDC offers both synchronous and asynchronous transfer of LMUs, and allows extensions to query an LMU that is targeted at them before accepting it and possibly rejecting it. The LMDC is directly accessible to any capability through the Core.

SATIN is well-suited for the construction of LMUs, because it promotes the decoupling of applications into discrete modules of specified functionality, which we can identify and build dependency trees for. We have the flexibility of sending any part of an application to appropriate recipients, allowing for identification and security. We also allow the inspection and rejection of LMUs by extensible capabilities, as well as their utilisation even if the recipient does not know how to. Using the concept of the LMU and the LMDC, we can simulate and implement any LM paradigm and allow any application to use it. Using the LMUs and the LMDC, we also allow for the updating and installation of new capabilities; we simply have to specify the Core as the destination target.

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 dynamic instance of SATIN must have an LMDC.

A prototype of SATIN has been implemented in Java (J2SE) and currently takes 64KB. This excludes KXML2 [kxm] (24KB), the XML parser for messages, and MuCode [Pic98] (40KB) which is used to transfer code around the network. The prototype includes an LMDC, multicast advertising and discovery capabilities, centralised publish/subscribe advertising and discovery capabilities and single-hop communication capabilities, with multi-hop planned for future releases.

Figure 4.4 shows the middleware in the context of the ISO/OSI networking model. Although traditionally middleware implements the presentation and session layers of the model, SATIN also implements the transport and network levels. The aim of this is to encapsulate network and transport protocols (our prototype uses TCP and UDP over IP),

to cater for the heterogeneous networking environment that SATIN is exposed to.

The componentisation of SATIN allows for great customisation: We can have many different interoperable instances of SATIN, specifically tailored to suit specific devices.

Chapter 5

Evaluation

-Work in Progress-

5.1 Applying SATIN: The Dynamic Program Launcher

Inspired by the scenario presented in Section 2.4.2, we have designed and implemented a prototypical application using SATIN, to test our claims.

The application implemented is a Dynamic Program Manager or Launcher for mobile devices, similar to the PalmOS Launcher in that its basic purpose is to display and launch application capabilities that are registered with the Core. The applications installed are shown as buttons, with the capability identifiers as labels. The Launcher also manages and controls all capabilities installed, enabling and disabling them. Figure 5.1 shows the manager displaying all the capabilities that are installed, including their type (advertisable, application, extensible etc) and version. The Launcher supports installing capabilities from many sources, including local storage, as demonstrated in Figure 5.2, as well as using any installed discovery services to find a capability and retrieve it using the LMDC.

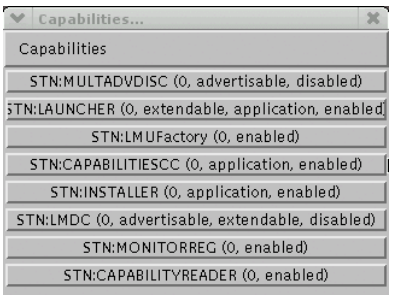

Figure 5.1: Managing installed Capabilities.

Figure 5.2: (a) Loading a Capability from disk. (b) Displaying the loaded Capability.

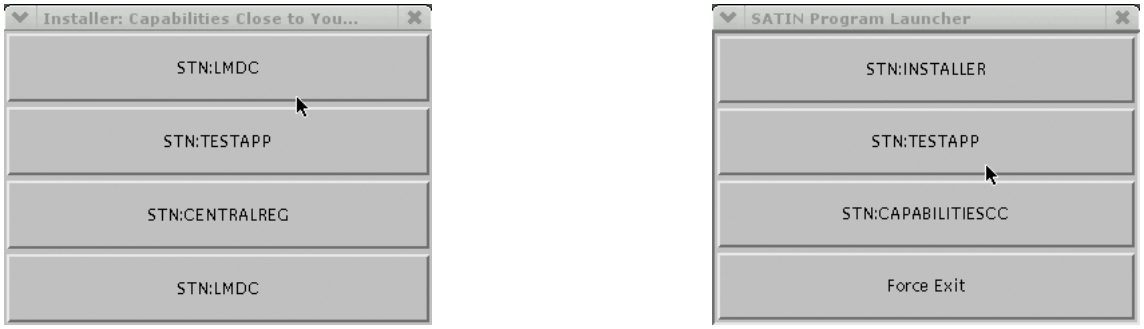

Figure 5.3: (a) Showing what capabilities are advertised on all networks, including those of the local host. (b) Capability "STN:TESTAPP" was installed from a remote host and is displayed by the Launcher.

The Launcher is dynamic, as it has the ability to search for new versions of capabilities that are available in any networks that the device is currently connected to, download and install them, either transparently, or as a result of a user command. It can also verify the authenticity of those downloads via a centralised server. We deployed an implementation of the Core, that monitors the usage of the capabilities installed: If the device running the Launcher runs out of memory, it can delete unused capabilities based on their frequency of use.

The application caters for the scenario presented in Section 2.4.2: Mobile devices roam through a dynamic context, able to transparently update their libraries and install new applications available in their current environment.

The SATIN Launcher is an application capability that depends on a number of other capabilities, as shown on Figure 5.4. As all SATIN applications, it is essentially a collection of capabilities.

Capability Identifier	Capability Type	Description	Size	Depends On
STN:LAUNCHER	Application	The SATIN Application Launcher	8	STN:MONITORCORE
STN:MONITORCORE	Extensible	An implementation of the Core that monitors frequency of usage of capabilities	12	MONITOR:REGISTRAR
STN:MONITORREGISTRAR	-	Registers new Capabilities with the Core and Monitors their usage frequency	8	STN:MONITORCORE
STN:CAPABILITIESCC	Application	Manages Installed Capabilities	8	STN:CAPABILITYREADER, STN:UPDATER
STN:CAPABILITYREADER	-	Loads Capabilities from disk and registers them with the Core.	4	STN:MONITORCORE
STN:INSTALLER	Application	Uses the LMDC and any discovery services to request and install capabilities remotely	4	STN:BASICLMDC
STN:BASICLMDC	Advertisable, Extensible	An implementation of the LMDC based on MuCode.	16	STN:LMUFACTORY
STN:LMUFACTORY	-	Used by the LMDC to construct LMUs	12	STN:MONITORCORE
STN:TESTAPP	Application, Advertisable	A Test application which we use to install and update from the network.	4	STN:MONITORCORE
STN:MULTICASTDISCADV	Advertisable, Advertiser, Discovery	An advertiser and discovery service, using Multicast.	32	STN:MONITORCORE
STN:CENTRALREGISTRY	Advertisable	A registry which can be used by hosts to advertise their capabilities and query about those of others.	12	STN:MONITORCORE
STN:CENTRALDISCADV	Advertiser, Discovery	An advertising and discovery service using the registry above.	8	STN:MONITORCORE
STN:UPDATER	-	The Capability that is used to update the system.	12	STN:MONITORCORE

Figure 5.4: The SATIN Launcher as a collection of Capabilities. Sizes are in KB (uncompressed).

5.1.1 Testing

We have tested the application with three devices: a PDA running Linux equipped with an 802.11b card in ad-hoc mode, a laptop equipped with a 802.11b card (again in ad-hoc mode) and an Ethernet card, and a server with an Ethernet card. As such, the laptop could communicate with both the server and the PDA, whereas the PDA and the server could only communicate with the laptop. All three machines were running Linux. The PDA, specifically, was running a beta version of JDK-1.3, with no JIT compilation. The server was running a static instance of SATIN, whereas the other devices were running a

Figure 5.5: Our test configuration: A laptop with two networking interfaces (Ethernet, 801.11b), a PDA with a wireless interface and a server with Ethernet.

dynamic instance. In their initial configuration, the machines had the following capabilities installed[1]:

Server	STN:BASICLMDC, STN:TESTAPP (version 2), STN:CENTRALREGISTRY
Laptop	STN:LAUNCHER, STN:BASICLMDC, STN:MULTICASTDISCADV, STN:CENTRALDISCADV, STN:UPDATER
PDA	STN:LAUNCHER, STN:TESTAPP (version 1), STN:MULTICASTDISCADV, STN:UPDATER

The laptop and PDA used the multicast advertising and discovery service to communicate over 802.11b, whereas the laptop and the server used the centralised registry over Ethernet. In our tests, the server was advertising the availability of version 2 of a capability with identifier STN:TESTAPP, version 1 of which was installed on the PDA. The laptop installed version 1 of the capability from the PDA, updated it to version 2 from the server and then the PDA updated its copy from the laptop. Figure 5.6 shows the test sequence. The table below shows the Java heap memory usage and the startup time for the Launcher on the PDA, the time it took for STN:TESTAPP to be installed from the PDA to the Laptop, the time it took for the laptop to update STN:TESTAPP to version 2 from the server and the time it took for the PDA to update to version 2 from the laptop.

Startup Time on PDA	21 seconds
Memory Usage on PDA	1155KB
Installation time of Capability from PDA to Laptop:	1998ms
Update time from Server to Laptop	1452ms
Update time on from Laptop to PDA	2063 ms

The results obtained above show that the middleware implementation is lightweight and that it satisfies the goals and principles outlined in chapter 3. Please note that SATIN is not optimised and is still in a prototype state that we used as a proof of concept. We attribute the large time difference between the tests when the PDA was involved (installation time from the PDA to the laptop and update time from the laptop to the PDA) and when it was not (update time on from the server to the Laptop) to the fact that the PDA runs a beta version of an interpreted JVM and to the nature of the wireless network that was used. We attribute the time difference between installing from the PDA to the laptop and updating from the laptop to the PDA to the fact that the PDA discovery capability had

[1]Please note that some have been omitted for clarity. See Figure 5.4 for more details and descriptions.

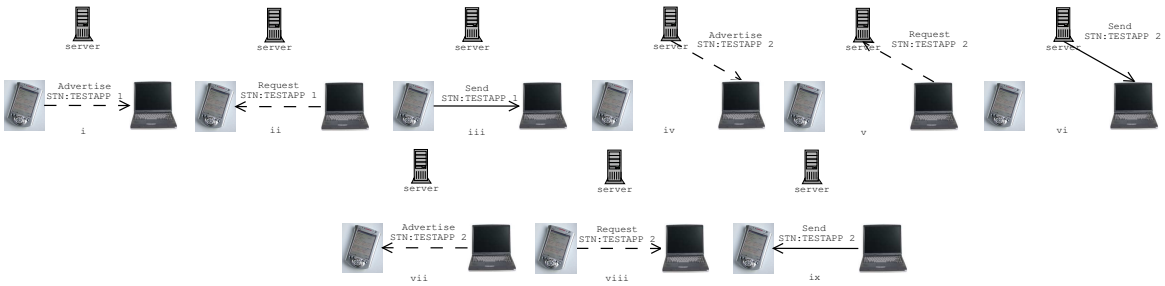

Figure 5.6: The testing sequence. Solid lines represent capability download, dotted lines represent requests and advertising.

to recognise that there was an updated version of the capability in reach.

5.2 Discussion

The advantages that this dynamic approach offers compared to the static approach of other launchers (including the PalmOS launcher) are the followings:

- Flexible application installation and update. We allow for the installation and updating of specific parts (capabilities) of applications and libraries. We also cater for versioning and dependencies. The updates are downloaded from any source in the any network to which the device is connected. On the other hand, the PalmOS launcher supports the update of complete applications only, which are monolithically developed and stored on the handheld as a single file. Moreover, the user has to explicitly connect the device with the source of the update (usually a desktop computer) to perform the update. Finally, PalmOS has no concept of dependencies or versioning.
- Self-Healing. Closely tied with the point above, this approach allows for self-healing applications, by allowing for their transparent customisation, reconfiguration and update based on the context. The Palm Launcher on the other hand, does not support application adaptation.
- Ease of use. The updating process is automated and transparent to the user. The installation process can also be automated. To install or update an application using the PalmOS launcher on the other hand, the user has to explicitly know and request it.
- Better use of resources. We send and receive code from peers in reach when possible, using inexpensive or free network connectivity. We are also able to remove capabilities to conserve resources. The PalmOS launcher on the other hand does not support such concepts.

The implementation of the program launcher and the testing have confirmed that SATIN is reasonably lightweight, despite the offered features tackling heterogeneity and the added flexibility. The heterogeneity of our target environment is handled by SATIN as follows: The use of advertising, discovery and communication capabilities handles network heterogeneity. The decoupling of applications into capabilities, which are versioned and uniquely identified, tackles software heterogeneity and interaction. And the combined use of LM, advertising and capabilities, allows for reaction to changes in the environment and caters for hardware heterogeneity. The ability of the middleware to reconfigure also tackles middleware heterogeneity.

It should be noted that the Launcher we developed only uses client-server interactions and code on demand; other applications could use any paradigm. For example, future versions of the Launcher could use mobile agents to roam the network searching for new versions of capabilities. Also, upon closer inspection of our work, it is evident that our design does not inherently support updating of capabilities by "patching", or changing individual bytes. There are two reasons for this: First, this can be implemented on top of SATIN at a higher level, as a capability itself, using the LMDC to transfer LMUs with the patch. Second, the modularisation of applications alleviates the need for "patching" as we allow for the update of individual capabilities.

Moreover, our implementation does not support the transfer of execution state with the code. This is partially because our implementation is based on Java and an unmodified Java Virtual Machine, which does not support the transfer of the bytecode execution state. Future implementations in other languages may well do. When developing SATIN we decided to prefix the identifiers of all capabilities that we provided with STN:. There is no reason for the prefix other that to signify that the capabilities were provided by us.

The most significant speed bottleneck in SATIN is MuCode [Pic98], which we used to implement logical mobility. This is because MuCode includes a lot of infrastructure which is not needed in our middleware, including it's own encapsulation mechanism, its support for resuming threads etc. We plan to replace this module in the future with something more efficient.

SATIN is not the first project in which we use logical mobility techniques in a mobile environment. An earlier approach was used in XMIDDLE [MCZE02] a mobile middleware, which allows for the reconciliation of mobile data. In the development of XMIDDLE, we realised that it would be advantageous to be able to chose at runtime which protocol to use to perform the reconciliation of changes and implemented an architecture which allowed for deciding upon, retrieving and using a reconciliation protocol at runtime. Moreover, in other previous work [ZME02, ZME03], we have identified a number of examples showing that logical mobility can bring tangible benefits to mobile applications. Our efforts in designing SATIN are certainly based on that experience. For example, the concept of modularising everything, including the middleware itself, came from the desire to be able

to create a superset of previous approaches, in that SATIN can support and implement the solutions that previous works have offered, but its use is not strictly limited by them. Thus the modularisation of SATIN allows us to adapt these previous approaches, like XMIDDLE or Lime, on our middleware system. The usage of XML and choice of XML parser was also influenced by our previous work on XMIDDLE, where we had found the parser we had chosen to be our biggest bottleneck.

Compared to the related work, our approach does not limit how applications use logical mobility techniques; as such, it can support solutions that have been given by previous approaches, but its use and applicability is more adaptive and general. Existing middleware systems such as Lime can be implemented on top of SATIN, thus giving SATIN applications interoperability with hosts running Lime. The way in which ReMMoc tackles heterogeneity through discovery and adaptation to different services can also be emulated with SATIN. The general adaptability and flexibility through logical mobility allows SATIN-based applications to heal and mutate according to context, making them extremely suitable for mobile computing. Moreover, our approach to advertising and discovery, as well as the modularisation that we encourage, propose and support through infrastructure, make SATIN demonstrably suitable for roaming.

Chapter 6

Conclusion

The main contribution of this thesis is the investigation of the use of Logical Mobility techniques and componentisation for mobile self-organisation, through a mobile middleware system. We have designed and implemented SATIN, a generic platform that offers self-organisation through logical mobility and that can be used to tackle the heterogeneity inherent in a mobile environment. SATIN, unlike other approaches, allows applications to use any logical mobility paradigm. The prototype application built demonstrates this functionality, as it allows applications to be dynamically deployed and updated.

There are two main directions of research that may follow this work:

1. **Security and Proof Carrying Code.** Our current design and implementation offers basic security, by allowing for a Logical Mobility Unit to be digitally signed. This, however, assumes that there is a trusted third party that is issuing the signatures. In a large peer to peer network of mobile devices, this is not always possible or desirable. Further research could investigate into the usage of Proof Carrying Code [Nec97] (PCC). PCC is a technique that can guarantee that code produced by an untrusted party will adhere to a set of security rules, or a *security policy*.

   Moreover the security provided by the middleware could be further increased by providing secure channel capabilities.

2. **Performance Analysis of the Mobile Use of Logical Mobility Paradigms.** Our work can be used as a basis for modelling the performance of individual logical mobility paradigms based on the current context and to select which paradigm to use based on the results of this modelling, at the design stage. There is already some work [GM02] being undertaken from different groups on this matter.

Appendix A

Thesis Table of Contents

1. Introduction
    (a) Hypothesis Statement
    (b) Scope of this Work
    (c) Research Contribution
        i. Component Model for Self Organisation
        ii. Logical Mobility Primitives for Mobile Self Organisation
        iii. System Design and Implementation
        iv. Evaluation of Results
2. Background and Motivation
    (a) Introduction to Self-Organisation & Distributed Systems
    (b) Introduction to Logical Mobility
        i. Logical Mobility and Code Mobility
        ii. Paradigms
            A. Client - Server
            B. Remote Evaluation
            C. Code on Demand
            D. Mobile Agents
    (c) Introduction to Physical Mobility
        i. Types of Mobility
            A. Nomadic Computing
            B. Base Station Mobility
            C. Ad-Hoc Networking
    (d) Component-Based Systems

(e) Motivation
    i. Case Study: Industry State-of-the-Art Mobile Application Development
    ii. Case Study: Application Deployment and Update in a Mobile Environment
    iii. Limitations of Related Work

3. Component Model for Self-Organisation
    (a) Requirements
    (b) Design
        i. Component Based Middleware
        ii. Application Componentisation

4. Logical Mobility Primitives For Mobile Computing
    (a) Requirements
    (b) Paradigm Representation
    (c) Language Abstraction
    (d) Mobile Deployment

5. The SATIN Mobile Computing Middleware System
    (a) System Design
        i. Capabilities
            A. Application Capabilities
            B. Advertising & Discovery Capabilities
            C. Advertisable Capabilities
            D. Extendable Capabilities
            E. Application Capabilities
        ii. Use of Logical Mobility
            A. Logical Mobility Units
            B. The Logical Mobility Deployment Capability
    (b) System Implementation
        i. Communication
        ii. Message Encoding
        iii. Logical Mobility
        iv. Programming Interface

6. Evaluation
    (a) Component Model Evaluation
    (b) Evaluation of the Effectiveness of the use Logical Mobility Primitives

Appendix B

Plan for Completion

This research project has been active for just over two years. Middleware for Mobile computing was an active field of research even at the start of our approach. However, research on the use of Logical Mobility primitives in mobile middleware systems was and still is, quite limited. Having had the privilege of publishing and receiving constructive and encouraging feedback at conferences and workshops, we believe that there is general acceptance for the relevance, merits and originality of our work.

Given this, we are confident that we will be able to submit the thesis by June 2004. The core task that still needs to be investigated before the submition date is the evaluation. More specifically:

- Extending the component model introduction. (Completed by December 2003)
- More Evaluation of SATIN: Scalability issues, more applications and metrics. (Completed by January 2004). We plan to do this by running comprehensive tests with more hosts, which transfer variable sizes of logical mobility units. We are also planning to replace the use of MuCode [Pic98] with an in-house mobile code toolkit.
- Evaluation of our component model. This will include metrics to show its efficiency. (Completed by February 2004). We plan to do this by testing the size of each type of reference and the size of applications represented as sets of capabilities, versus monolithic applications. We will also show how the size varies with the included metadata.
- Evaluation of the use of Logical Mobility techniques for self-organisation: We need to show that mobile applications that use logical mobility can bring significant advantages than those that don't and that new types of applications are possible. (Completed by March 2004). We plan to do this by showing new applications that are possible using our model but impossible or infeasible using traditional ones. We

are also going to show application functionality which is infeasible to implement otherwise. We are also considering implementing XMIDDLE [MCZE02] on top of SATIN, demonstrating how the latter handles middleware heterogeneity.

This gives us three months to do the writeup. This document is written in the basic structure of our thesis. Sections which need to be expanded have been clearly marked. We believe that the weak sections are the evaluation and the related work, regarding component model systems.

Appendix C

List of Publications

- S. Zachariadis, C. Mascolo, and W. Emmerich, "Self-Organising Mobile Systems: Use of Logical Mobility Primitives in Mobile Computing Middleware " Submitted for publication (MOBISYS 04).
- S. Zachariadis and C. Mascolo, "Adaptable mobile applications through SATIN: Exploiting logical mobility in mobile computing middleware," in *1st UK-UbiNet Workshop*, September 2003.
- S. Zachariadis, C. Mascolo, and W. Emmerich. Adaptable Mobile Applications: Exploiting Logical Mobility in Mobile Computing . In *In Proceeding of 5th Int. Workshop on Mobile Agents for Telecommunication Applications (MATA03)* , LNCS. Springer, pages 170-179, October 2003.
- S. Zachariadis, C. Mascolo, and W. Emmerich. Exploiting logical mobility in mobile computing middleware. In *Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, pages 385-386, July 2002.
- S. Zachariadis, L. Capra, C. Mascolo, and W. Emmerich, "XMIDDLE: Information sharing middleware for a mobile environment," in *Proceedings of the 24th International Conference of Software Engineering (ICSE 2002), Demo Session*, Orlando, Florida, May 2002, p. 712, ACM Press.
- C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich, "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," *Int. Journal on Personal and Wireless Communications*, vol. 21, no. 1, pages 77-103 April 2002.
- L. Capra, C. Mascolo, S. Zachariadis, and W. Emmerich. Towards a Mobile Computing Middleware: a Synergy of Reflection and Mobile Code Techniques. In *In Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, pages 148–154, Bologna, Italy, October 2001.

Bibliography

[AAH+97] G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *Baltzer/ACM Wireless Networks*, 3(5):421–433, 1997.

[AOS+99] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini[tm] Specification*. Addison-Wesley, 1999.

[BCB+02] G. S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, Rui Moreira, and Nikos Parlavantzas. Reflection, self-awareness and self-healing in openorb. In *Proceedings of the first workshop on Self-healing systems*, pages 9–14. ACM Press, November 2002.

[BGP97] M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. In *Proceedings of the First International Workshop on Mobile Agents*, pages 13–26, Berlin, Germany, April 1997.

[BPW98] A. Bieszczad, B. Pagurek, and T. White. Mobile agents for network management. *IEEE Communications Surveys*, 1(1):2–9, 1998.

[CBM+02] L. Capra, G. S. Blair, C. Mascolo, W. Emmerich, and P. Grace. Exploiting Reflection in Mobile Computing Middleware. *ACM SIGMOBILE Computing and Communications Review*, 6(4):34–44, October 2002.

[CEM01] L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proc. of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *LNCS*, pages 126–133, Kyoto, Japan, September 2001.

[CLP00] M. Childs, P. Lomax, and R. Petrusha. *VBScript in a Nutshell*. O'Reilly, May 2000.

[CMZE01] L. Capra, C. Mascolo, S. Zachariadis, and W. Emmerich. Towards a Mobile Computing Middleware: a Synergy of Reflection and Mobile Code Techniques.

In *In Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, pages 148–154, Bologna, Italy, October 2001.

[CP02] G. Cugola and G. Picco. Peer-to-peer for collaborative applications. In *Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, pages 359–364, July 2002.

[dis] Distributed.net. http://www.distributed.net.

[EE98] G. Eddon and H. Eddon. *Inside Distributed COM*. Microsoft Programming Series. Microsoft Press, Redmond, WA, 1998.

[Emm00] W. Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, April 2000.

[FH02] S. Fickas and R. J. Hall. Self-healing open systems. In *Proceedings of the first workshop on Self-healing systems*, pages 99–101. ACM Press, November 2002.

[FPV98] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[FS01] A. Finkelstein and A. Savigni. A Framework for Requirements Engineering for Context-Aware Services. In *Proc. of* 1^{st} *International Workshop From Software Requirements to Architectures (STRAW 01)*, Toronto, Canada, May 2001.

[GGGO99] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony. Using mobile agents for distributed network performance management. In Sahin Albayrak, editor, *Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA-99)*, volume 1699 of *LNAI*, pages 96–112, Berlin, August 9-10 1999. Springer.

[GJK02] I. Georgiadis, Magee J, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM Press, November 2002.

[GM02] Vincenzo Grassi and Raffaela Mirandola. PRIMAmob-UML: a methodology for performance analysis of mobile software architectures. In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP-02)*, pages 262–274, New York, July 2002. ACM Press.

[Gri97] R. Grimes. *DCOM Programming*. Wrox, 1997.

[HBSG99] O. Holder, I. Ben-Shaul, and H. Gazit. Dynamic layout of distributed applications in FarGo. In *Proceedings of International Conference on Software Engineering*, pages 163–173, May 1999.

[HK01] S. Hashman and S Knudsen. The application of jini technology to enchance the delivery of mobile services. http://www.sun.com/jini/whitepapers/PsiNapticMIDs.pdf, December 2001.

[KRL+00] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L.C. Magalh aes, and R.H. Cambpell. Monitoring, Security, and Dynamic Configuration with the *dynamicTAO* Reflective ORB. In *International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, pages 121–143, New York, April 2000. ACM/IFIP.

[kxm] The kxml2 xml parser. http://kxml.enhydra.org/.

[LKAA96] S. Long, R. Kooper, G.D. Abowd, and C.G. Atkenson. Rapid prototyping of mobile context-aware applications: the Cyberguide case study. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking*, pages 97–107, White Plains, NY, November 1996. ACM Press.

[LO98] D. B. Lange and M. Oshima. *Programming and Deploying Java^{TM} Mobile Agents with Aglets^{TM}*. Addison-Wesley, 1998.

[MCE02] C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In E. Gregori and G. Anastasi and S. Basagni, editor, *Advanced Lectures on Networking*, number 2497 in Lecture Notes in Computer Science, pages 20–58. Springer, 2002.

[MCZE02] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Int. Journal on Personal and Wireless Communications*, 21(1):77–103, April 2002.

[Met99] R. Mettala. Bluetooth Protocol Architecture. http://www.bluetooth.com/developer/whitepaper/, August 1999.

[MPR01] A. Murphy, G. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-01)*, pages 524–536, Los Alamitos, CA, April 16–19 2001. IEEE Computer Society.

[MRS02] P. Mathieu, J. C. Routier, and Y. Secq. Dynamic organization of multi-agent systems. In Maria Gini, Toru Ishida, Cristiano Castelfranchi, and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 451–452. ACM Press, July 2002.

[Nec97] G. C. Necula. Proof-carrying code. In *The 24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 106–119. ACM SIGACT and SIGPLAN, ACM Press, January 1997.

[OMG95] *The Common Object Request Broker: Architecture and Specification Revision 2.0.* 492 Old Connecticut Path, Framingham, MA 01701, USA, July 1995.

[P. 03] P. Grace and G. S. Blair and Sam Samuel. Middleware Awareness in Mobile Computing. In *Proceedings of First IEEE International Workshop on Mobile Computing Middleware (MCM03) (co-located with ICDCS03)*, pages 382–387, May 2003.

[pal] Palmsource developers program. http://www.palmsource.com/developers/.

[PB01] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, pages 124–130. ACM Press, May 2001.

[Pic98] G. P. Picco. μCODE: A Lightweight and Flexible Mobile Code Toolkit. In Kurt Rothermel and Fritz Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, Lecture Notes in Computer Science, pages 160–171, Berlin, Germany, September 1998. Springer-Verlag.

[Pow90] J. Power. Distributed systems and self-organization. In *Proceedings of the 1990 ACM annual conference on Cooperation*, pages 379–384. ACM Press, 1990.

[RKC01] 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.

[RMI98] Sun Microsystems. *Java Remote Method Invocation Specification*, Revision 1.50, JDK 1.2 edition, October 1998.

[set] Seti@home, the search for extraterrestrial intelligence. http://setiathome.ssl.berkeley.edu.

[Sha02] M. Shaw. Self-healing: Softening precision to avoid brittleness: Position paper for woss '02: Workshop on self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 111–114. ACM Press, November 2002.

[Sun95] Sun Microsystems. *The Java Language Specification*, Oct 1995.

[Sun98] Sun Microsystems. *Java Object Serialization Specification*, 1998.

[Syn00] SyncML. Building an Industry-Wide Mobile Data Synchronization Protocol. http://www.syncml.org/technical.htm, 2000.

[TD98] IONA Technologies and Dublin. *Orbix IIOP engine.* 1998.

[V. 96] S. Kerry V. Hayes. IEEE P802.11. Specification, Institute of Electrical and Electronics Engineers, Inc. (IEEE), 1996.

[Wal99] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[WBS02] Y. Weinsberg and I. Ben-Shaul. A programming model and system support for disconnected-aware applications on resource-constrained devices. In *Proceedings of the 24th International Conference on Software Engineering*, pages 374–384, May 2002.

[WPM99] D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. *Communications of the ACM*, 42(3):92–102, 1999.

[ZME02] S. Zachariadis, C. Mascolo, and W. Emmerich. Exploiting logical mobility in mobile computing middleware. In *Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, pages 385–386, July 2002.

[ZME03] S. Zachariadis, C. Mascolo, and W. Emmerich. Adaptable mobile applications: Exploiting logical mobility in mobile computing. In *5th Int. Workshop on Mobile Agents for Telecommunication Applications (MATA03)*, pages 170–179. LNCS, Springer, October 2003.