

# **Middleware Support for Mobile Computing Applications**

**PhD First Year Report**

**Paul Grace**

Supervisor: Professor Gordon Blair

Computing Department,  
Lancaster University,

**September 2001**

## Table of Contents

---

<b>Chapter 1</b>	<b>Introduction</b>	<b>4</b>
1.1	Overview	4
1.2	Structure of the report	4
<b>Chapter 2</b>	<b>Background on Mobility and Mobile Computing</b>	<b>5</b>
2.1	Introduction	5
2.2	Technology supporting mobile computing	5
2.2.1	Mobile devices	5
2.2.2	Mobile networks	5
2.2.3	Network protocols for mobile hosts	7
2.3	Location based services and interaction	7
2.3.1	Jini	7
2.3.2	SLP	8
2.3.3	Other service discovery technologies	8
2.4	Mobile applications	9
2.4.1	M-Infotainment	9
2.4.2	M-Commerce	9
2.4.3	Healthcare and emergency services	10
2.4.4	Tourism	10
2.4.5	Context-aware applications	10
2.5	Analysis	10
2.5.1	Mobile technology analysis	10
2.5.2	Analysis of service discovery platforms	11
2.5.3	Analysis of mobile applications	11
2.6	Conclusions	11
<b>Chapter 3</b>	<b>Middleware for Mobile Computing</b>	<b>13</b>
3.1	Introduction	13
3.2	Base middleware platforms	13
3.2.1	Common Object Request Broker Architecture (CORBA)	13
3.2.2	CORBA in a mobile environment	14
3.2.3	Alternative base middleware platforms	15
3.2.4	Analysis of support for mobile computing	16
3.3	Reflective middleware	17
3.3.1	Introduction	17
3.3.2	Open ORB	17
3.3.3	DynamicTAO	18
3.3.4	Universally Interoperable Core™ (UIC)	19
3.3.5	Other reflective middleware technologies	21
3.3.6	Analysis of reflective middleware for mobile computing	22
3.4	Conclusions	22
<b>Chapter 4</b>	<b>Asynchronous Middleware for Mobile Computing</b>	<b>23</b>
4.1	Introduction	23
4.2	Tuple Spaces	23
4.2.1	Introduction	23
4.2.2	L <sup>2</sup> imbo	23
4.2.3	JavaSpaces	24
4.2.4	T-Spaces	24
4.3	Agents	24
4.3.1	Introduction	24
4.3.2	Tacoma and Tacoma Lite	24

4.3.3	Aglets	26
4.4	Message-orientated and event based middleware	26
4.4.1	Introduction	26
4.4.2	iBus (Message orientated)	27
4.4.3	Cambridge Event Architecture (CEA) – (event-based)	27
4.5	Other important systems	28
4.6	Analysis and conclusions	28
<b>Chapter 5</b>	<b>Other Architectural Support for Mobile Applications</b>	<b>30</b>
5.1	Introduction	30
5.2	OSA and MExe	30
5.3	Context aware middleware	31
5.4	Conclusions	32
<b>Chapter 6</b>	<b>Research Proposal</b>	<b>33</b>
6.1	Introduction	33
6.2	Statement of aims	33
6.2.1	Main aim	33
6.2.2	Specific objectives	34
6.2.3	Methodology and approach	35
6.2.4	Plan of work	36
<b>Chapter 7</b>	<b>Conclusions</b>	<b>38</b>
<b>References</b>		<b>39</b>

### 1.1 Overview

The rise in popularity of mobile computing has been fuelled in recent times by numerous advances in mobile device technology and improving wireless communication infrastructures. Subsequently, new types of mobile applications (as opposed to the standard e-mail and web browsing applications) have emerged. These include navigation systems, tourist guides, intelligent mobile services, mobile multimedia applications and many others, in which the user interacts with fixed, location-based services or other mobile users. However, developing these applications for a mobile environment is a complicated task for the application programmer; wireless environments present a number of problems including fluctuating levels of network service, periods of disconnection and constant environmental changes, e.g. device location change and discovering new local hosts and services. Furthermore, these applications normally execute on devices with limited resources such as memory, battery power and CPU speed.

To overcome the dynamic nature of mobile environments, a standard approach is to develop adaptive applications [Katz, 94]. These applications adapt their operation dependent on the current environment, to offer the best level of service to the user. However, this approach still requires the developer to discover and react to current environmental information and deal with the distribution problems of mobility. Therefore, middleware has been cited as an appropriate technology to provide the programmer with a complete support platform for the development of mobile applications [Friday, 96].

Middleware is simply defined as the layer of software residing between the operating system and application on every machine, whose purpose is to mask heterogeneity and provide a consistent and integrated distributed programming environment. It has proved a successful technology in the realm of wired distributed applications, especially in the domains of finance and banking. However, current research is examining how middleware can benefit the next generation of applications such as multimedia and mobility. It has been identified that existing middleware technologies (e.g. CORBA [OMG, 95] and DCOM [DCOM, 96]) present a fixed service to the user that is unsuitable for a mobile environment [Blair et al, 01], consisting of changing local service technologies, local users and environmental properties (e.g. network level of service). Therefore, the main aim of the proposed research is to develop a dynamically re-configurable middleware platform to support the interaction with newly discovered services and mobile users.

### 1.2 Structure of the report

The development of middleware platforms that support mobile applications is the underlying subject of this report and the PhD research proposal that is described within it. Chapter 2 first examines the topic of mobile computing, introducing the technologies that support it and the types of applications and services that mobile middleware must support. Subsequently, chapter 3 and chapter 4 look in detail at middleware platforms; the two chapters offer alternative methods to deal with mobility. Chapter 3 reports first on existing standard commercial platforms and then research-based, adaptable middleware technologies as potential mobile platforms. Chapter 4 then illustrates alternative asynchronous middleware technologies that claim to be suited to mobile computing. Following this, chapter 5 looks at other issues that need to be supported in mobile computing including context awareness and dynamic applications. Chapter 6 presents the proposed research for the next stages of the PhD project; the main ideas of the project including principle aims and objectives, methodology and the plan of work to be carried out are described. Finally, chapter 7 draws general conclusions from this report and the proposed project.

### 2.1 Introduction

Advances in wireless networking technology over the previous decade generated a new computing paradigm known as *mobile computing*, defined by users carrying portable devices that access shared infrastructures independent of their physical location; this allows flexible communication between people and continuous access to networked services [Forman & Zahorjan, 94]. The importance of mobile computing is escalating due to continued improvements in end system technologies generating smaller and more portable devices; notably, mobile phones, Personal Digital Assistants (PDAs) and handheld computers now account for a significant proportion of computer sales [Wade, 99].

To provide a summary of the properties of mobile computing this chapter first examines the key technologies that support it. Subsequently, the key issue of service discovery & interaction is investigated. Finally, an overview of past and current mobile applications is provided.

### 2.2 Technology supporting mobile computing

#### 2.2.1 Mobile devices

A key factor in the uptake of mobile computing lies with the mobile end system; this must be lightweight, conservative with power, and easy to use. Currently available mobile devices fall into a set of categories based on differences in physical size, battery life, screen size, system memory and processor speed. A selection of commercially available devices is compared in table 2.1 to illustrate this.

Table 2.1 identifies 4 types of mobile computers that are described as follows. *Laptop computers* provide the closest in terms of performance to desktop machines; however, they are also the largest in size and the most expensive. The boundary between the more portable *handheld PCs* and *PDAs* is now particularly blurred; previously they differed in performance (PDAs had slower CPUs, less memory and monochrome screens) and the application types they provided (PDAs initially offered applications of an organizational nature as opposed to desktop equivalents). However, there is now a striking convergence of the two, although handheld PCs may still provide a keyboard as standard for data entry. Alternatively, *Smartphones* are mobile phones that provide both cellular voice connectivity and PDA-type applications (organiser, address book etc.) and illustrate the concept of mobile phones as a multi-function mobile computing device.

In contrast to the available systems, *wearable computers* are being researched and developed; for example, IBM has developed a wristwatch that runs the Linux operating system [IBM, 00]. This and other devices such as head-mounted sets, buttons [iButton, 01] and other wearable components with computational capacity lead into the realm of *ubiquitous computing* [Weiser, 1991], but are important to consider as they may become the mobile devices of the future.

#### 2.2.2 Mobile networks

Advances in wireless networking technologies over the past decade have provided a range of connectivity types within both wide and local areas. Wireless LANs fall into two technology types: infrared and radio frequency. An example of an infrared LAN is IrDA [IrDA, 01]; its main use is the wireless connection of devices that would normally use cables. IrDA is a point-to-point, narrow-angle (30-deg), ad-hoc data-transmission standard designed to operate over a distance of 0 to 1 m and at speeds of 9.6 kb/s to 16 Mb/s. However, once an IrDA device is connected to the LAN, it must

remain relatively stationary to maintain the connection. Alternative radio frequency LAN technologies are Bluetooth, IEEE802.11b, and HomeRF. Bluetooth [Bluetooth, 99a] provides short-range, point-to-multipoint voice and data transfer and can transmit through solid, non-metal objects. The nominal range of a Bluetooth device is 10 cm to 10 m but can be extended to 100 m by increasing the transmit power. IEEE802.11b [Crow et al, 97] and HomeRF [HomeRF, 01] provide wireless coverage of larger areas such as buildings and provide capacity for greater volumes of traffic.

Category	Product	RAM (Mb)	CPU	MHz	Battery (type, hrs)	Weight (g)	Display (pixels)	Display (")	OS
Laptop	Range	32-256	Range	500-850	Lithium rec., 3-6	(to) 3500	(to)1024 x768	(to) 15	Range
Handheld PC	HP Jordana	32	Intel StrongARM sa-1110	206	Lithium rec., 9	510	640x 240	6.5	Win CE
PDA	Compaq iPaq 363	32	Intel StrongARM sa-1110	206	Lithium rec., 12	179	240x320	2.26x3.02	Win CE
PDA	Palm m505	8	Motorola dragonball	33	Lithium rec. (2wks)	139	160x160	2.25	Palm OS 4.0
PDA	Handspring Visor Edge	8	Motorola dragonball	33	Lithium rec., (2 mths)	136	160x160	2.375	Palm OS 3.5
Smartphone	Kyocera QCP6035	8	Motorola dragonball	33	Lithium rec., 5	198	160x160	1.8	Palm OS 3.5

**Table 2.1** Comparison of currently available mobile devices (July 2001)

Alternatively, wireless WANs can cover anything from a city to a continent. Furthermore, they are growing in increasing capacity with the change from circuit-switched technologies such as GSM [Rahnema, 93] to packet-based technologies such as UMTS [Muratore, 00] allowing higher data throughput more suited to mobile applications. Table 2.2 compares a selection of the currently available technologies.

The collection of wireless technologies presented allows a user to be continuously connected; that is, they may *roam* from location to location using whichever network technology is available. For example, the user may be connected to the network while within a building covered by an 802.11b network, when they then move outside of the building the device communicates by using a GSM network. This concept is known as *overlay networks* [Katz et al, 96].

Network	Type	WAN/LAN	Bandwidth
Bluetooth	Digital RF	LAN	57.6 Kb/s to 1Mb/s
GSM	Digital RF	WAN	2.4 to 9.6 Kb/s
GPRS	Digital RF	WAN	2 Mb/s
UMTS	Digital RF	WAN	32 Kb/s
802.11b	Digital RF	LAN	2 Mb/s
HomeRF	Digital RF	LAN	10 Mb/s
TETRA	Digital RF	WAN	9.6 to 19.2 Kb/s
RangeLan II	Digital RF	LAN	1.6 Mb/s
InfraLAN	Digital IR	LAN	9.6 kb/s to 16 Mb/s

**Table 2.2** Communication technologies and their available bandwidths [Wade, 99]

### 2.2.3 Network protocols for mobile hosts

A network protocol provides end-to-end delivery of datagrams from source to destination through an interconnected series of networks. Therefore, in order to maintain this level of transparency in the face of changing locations of mobile hosts, research into network protocols to support this has been undertaken. This section presents only the key Internet based approaches within this domain, i.e. Mobile IP and IP Version 6.

Each host throughout the Internet is assigned an Internet Protocol (IP) address, which uniquely identifies it and specifies its physical location within the network. That is, the IP address is fundamental to the routing of data, as well as identification. Therefore, when a host moves it cannot change its IP address without higher-level software being disrupted.

Mobile IP [Perkins, 96], a standard proposed by the IETF, was designed to solve this problem by allowing the mobile node to use two IP addresses: a fixed home address and a care-of address that changes at each new point of attachment. The *home address* is static and is used, for example, to identify TCP connections. The *care-of address* changes at each new point of attachment; it indicates the network number and consequently identifies the mobile node's point of attachment within the network topology. The home address makes it appear that the mobile node is continually able to receive data on its *home network*, where Mobile IP requires the existence of a network node known as the *home agent*. Whenever the mobile node is not attached to its home network the home agent gets all the packets destined for the mobile node and arranges to deliver them to the mobile node's current point of attachment. Whenever the mobile node moves, it *registers* its new care-of address with its home agent. To get a packet to a mobile node from its home network, the home agent delivers the packet from the home network to the care-of address.

Mobility support in IPv6 [Johnson & Perkins, 96], proposed by the Mobile IP working group, follows the design for Mobile IP. It retains the ideas of a home network, home agent, and the use of encapsulation to deliver packets from the home network to the mobile node's current point of attachment. While discovery of a care-of address is still required, a mobile node can configure its care-of address by using Stateless Address Auto-configuration and Neighbour Discovery. Thus, foreign agents are not required to support mobility in IPv6.

However, Mobile IP may face competition from alternative tunnelling protocols such as PPTP [Pal et al, 97] and L2TP [Palter et al, 97]. These other protocols, based on PPP, offer portability to mobile computers.

## 2.3 Location based services and interaction

An important element of mobile computing lies in the device's interaction with location-based services. That is, the ability to discover what services are available at a particular location and communicate with them. A simple example of this is a room that has a service available to control the light switch; a person who then enters the room with a PDA can discover the service and switch the lights on and off using their handheld. There are a number of existing service discovery technologies currently available, which are discussed as follows.

### 2.3.1 Jini

Jini is a Java based technology that provides an infrastructure for delivering services and creating spontaneous interaction between programs that use these services regardless of their hardware/software implementation. Services can be added or removed from the network, and new clients can find existing services without administration [Arnold et al, 99].

A service is defined by its programming API, declared as a Java programming language interface. When a service is added to a network, it advertises itself by publishing a Java object that implements the interface to a lookup service. Clients then locate nearby lookup services using the Jini discovery protocols. When a client finds an interface that matches its requirements it gets the service's published object (downloading any code it needs in order to talk to the service), thereby learning how to talk to the particular service implementation via the API. Also, the Jini architecture lets programs use services in a network without knowing anything about the wire protocol that the service uses; one implementation of a service might be RMI-based, and another CORBA-based.

Furthermore, Jini employs the concept of leasing. Each time a device joins the network and its services become available on the network, it registers itself only for a certain period of time, called a *lease*. This is especially useful for dynamic ad hoc network scenarios.

### 2.3.2 SLP

Service Location Protocol (SLP) [Veizades et al, 97] is being developed by the IETF and aims to provide a vendor independent standard for service discovery. It consists of three parts: *user agents* that perform discovery on behalf of the user or application, *service agents* that advertise the location and characteristics of a service and *directory agents* that collect service addresses from service agents and responds to user agents. The operation of these agents is shown in figure 2.1.



**Figure 2.1** Service discovery in SLP (a) using a discovery agent and (b) without using a discovery agent

SLP has a flexible and scalable architecture that is suitable for networks of different sizes i.e. it can be used in small ad-hoc networks without directory agents or in large enterprise networks with a directory agent as illustrated by figure 2.1.

### 2.3.3 Other service discovery technologies

Alternative service discovery technologies are also available. An open industry consortium ([www.salutation.org](http://www.salutation.org)) is developing the *Salutation* architecture [Salutation, 98]; this consists of Salutation Managers (SLMs) that have the functionality of service brokers. Services register their capabilities with an SLM, and clients query the SLM when they need a service. After discovering a service, clients can request to use the service through the SLM.

*Universal Plug and Play* (UPnP) [Microsoft, 01] is being developed by an industry consortium ([www.upnp.org](http://www.upnp.org)) lead by Microsoft. It provides peer-to-peer mechanisms for auto-configuration of

devices, service discovery, and control of services. In UPnP's current version there is no central service register, such as the DA in SLP or the lookup table in Jini. The Simple Service Discovery Protocol (SSDP) [Goland et al, 99] is used within UPnP to discover services.

Furthermore, the Bluetooth protocol stack contains the *Service Discovery Protocol* (SDP) [Bluetooth, 99b], which is used to locate services provided by or available via a Bluetooth device. It has been modified to suit the dynamic nature of ad-hoc communications and addresses service discovery specifically for this environment; thus, it focuses on discovering services, where it supports the following inquiries: search for services by service type, search for services by service attributes and service browsing without a priori knowledge of the service characteristics. SDP does not include functionality for accessing services. Once services are discovered with SDP, they can be selected, accessed, and used by mechanisms out of the scope of SDP.

## **2.4 Mobile applications**

[Wade, 99] categorises the range of applications that execute on mobile devices: stand-alone applications (e.g. a word processor) and existing simple distributed applications (e.g. e-mail & a WWW browser) that are similar to their wired counterparts. However, the main identification is that of advanced mobile applications that are linked primarily to the mobility of the user. These applications are based upon peer-to-peer and group interactions, collaboration between users, the transmission of multimedia data and the interaction with location based services.

The following sections examine a range of implemented mobile applications and the vertical domains that they fall into.

### **2.4.1 M-Infotainment**

Entertainment applications have proved popular on the wired network; moreover, applications like news and weather are particularly suited to the mobile domain by reporting the news based on your interests and context [Jacobsen & Johansen, 97]. Furthermore, information and entertainment applications rely heavily on the use of multimedia data; initial applications of this type include mobile video conferencing. Notably, many projects have attempted to create a complete suite of services for developing applications of this type for a mobile environment; a good example of this is MiLife [Lucent, 01] developed by Lucent Technologies that includes media streaming as one of its main functions. Additionally, interactive games are becoming increasingly popular in the wireless world and the ability to create ad-hoc communities of gamers offers a key mobile application.

### **2.4.2 M-Commerce**

Mobile commerce has presented a wide range of interesting application types: Reservation and ticketing systems that allow the user to book at nearby restaurants and taxi firms, and advertising applications allowing retailers to present their latest offers to customers close by. Notably, both applications take into account the user's current environment. However, alternative applications such as M-banking, M-shopping and auction systems that allow users to perform their wired counterparts while on the move remain less popular as they are not reliant on the user's mobility.

Automated tolling is another encouraging application area, for example the ROBIN Toll Collection System Architecture. Instead of the toll bars and bridges used in a conventional toll system, ROBIN uses the satellite-based GPS (Global Positioning System) to determine when a vehicle is on a toll road and charges the on-board device (this is similar to the range of existing navigational applications that are also currently available).

### **2.4.3 Healthcare and emergency services**

Mobile applications have been rigorously investigated within the healthcare domain to provide better levels of care especially in settings such as hospitals and surgeries. For example, [Mitchell et al, 00] present a scenario to provide doctors with information such as medical records and changes in patient status, whatever their location. Furthermore, the Remote Patient Monitoring System (RPMS) [Crumley et al, 99] allows vital information captured either at the scene or en route to hospital to be transmitted in real-time to a hospital emergency room. The information is monitored by a consulting doctor, who then diagnoses and transmits advice back to the ambulance.

### **2.4.4 Tourism**

Tourist guides provide examples of how content is suited to a user's particular context. Normally, tourists follow paper-based guides informing them of interesting information about the place that they are visiting. However, people tend to get lost easily and become unsure if the information is appropriate to their location. Furthermore, the information is also static and difficult to update. To improve upon this mobile devices present the latest information based on places of interest close by and direct them elsewhere. Examples of tourist guide applications are GUIDE [Davies et al, 99] and CyberGuide [Long et al, 96].

### **2.4.5 Context-aware applications**

This sub-section has identified that mobile computing has engendered new types of distributed applications that support the mobile user. However, the most important class of applications within mobile environments are *adaptive applications* [Katz, 94]; these operate in fluctuating environments and adapt themselves to provide the best level of service to the user. Hence, closely linked to mobile computing is the concept of *context-awareness*. Initial research identified that *context* was defined as: "where you are, who you are with and what resources are available to you" [Schilit et al., 94]. However, as mobile applications become more sophisticated the range of context they encompass will become greater. Therefore, a more general definition is: "context is any information which can characterize the situation of an entity, where an entity is a person place or object that is considered relevant to the interaction between a user and an application" [Dey & Abowd, 99]. Therefore, the application designer must be able to discover a range of context information allowing the application to adapt to any changes.

## **2.5 Analysis**

### **2.5.1 Mobile technology analysis**

Although often debated, there is no single technology (device or network) best suited to mobile computing. End users differ in the types of devices they use and in how they want to use the networks; for example, some users requires the cheapest solution available, while others want the best level of service for important data. When considering middleware for mobile applications the ability to incorporate all networks and mobile devices is fundamental. Mobile middleware must take into account that applications will execute in a heterogeneous environment on devices with varying levels of resources and different network types; in addition, the network type and device may change as the user roams. Therefore, middleware must support the application developer in coping with the changing characteristics of this heterogeneous environment.

### 2.5.2 Analysis of service discovery platforms

The presented discovery techniques all share similar architectures. However, they can be compared in how they support mobility. Jini's main strength is its environment for interaction, i.e. the ability to execute code discovered on the fly and communicate easily with services of different types. However, it is specific to the Java language and reliant on resource expensive virtual machines not always suited to portable devices; this leads to static proxy-based architectures that are inappropriate due to the dynamic nature of mobile environments. However, SLP overcomes these problems by providing a language independent solution that is also geared to smaller ad-hoc networks not carrying the expense of a central repository when not needed. Notably, Salutation extends this advantage as it is independent of both language and network platform (unlike for example UPnP, which is specific to IP).

The range of discovery technologies available presents a significant problem for mobile computing. That is, if a device uses one discovery technology it will only be able to find services registered using that technology and miss other locally available services. However, utilising all the technologies is infeasible due to the limited resources available.

Fundamentally, mobile middleware must provide an architectural service for the discovery of services that can locate and interact with any existing discovery technologies and also any newly available ones. This must also be parallel with the ability to interact with services of different types after they have been located.

### 2.5.3 Analysis of mobile applications

There are many additional application types not discussed; for example, applications to help in archaeology fieldwork [Ryan, 98] and office meeting tools, eg. ParcTab [Want et al, 95]. This demonstrates the diversity of mobile applications, which will only extend further as the popularity of mobile computing increases and visions of how it can improve current environments are identified. The key to mobile middleware is to provide support across these domains and help application designers overcome the problems of mobility in developing applications of this type.

The collection of applications illustrates three styles of interaction that are fundamental within mobile computing and as such must be supported by the middleware. Firstly, the mobile device must interact with fixed or location based services as they roam (e.g. tourist guides). Secondly, the user must interact with one or more local mobile users (e.g. mobile gaming) to create *virtual communities*. Finally, the user must be informed of new and relevant information independent of their mobile device (e.g. news sent to a computer or display board physically close to the user); therefore, the interaction with newly discovered devices not owned by (or specific to) a single user must be supported.

## 2.6 Conclusions

This chapter has illustrated that the two fundamental properties of mobile computing environments are heterogeneity and change. Mobile applications are developed upon a range of service discovery technologies and platform types. These in turn function upon different protocols, networks and mobile devices. Furthermore, the mobile environment is also subject to change, e.g. varying network bandwidth, network connectivity and device location. Therefore, mobile middleware must provide support to the application developer to overcome the level of heterogeneity and change that exists.

The survey of mobile applications and interactions within a mobile environment has presented three styles of interactions that must be supported by middleware to simplify the task of application programmers in developing mobile applications that utilise these types:

- ❑ Nomadic interaction – The user roams from location to location, utilising varying wireless networks to communicate with fixed, location-based services.
- ❑ Ad-hoc interaction – The user communicates directly with devices located physically close to them. This is extended to small groups of users forming *virtual communities*, e.g. mobile gaming.
- ❑ Presence (“Find me”) interaction – The user roams from location to location but is not linked to a specific mobile device; Information (e.g. video data) can be sent to the device they are currently using or close to them (video display, desktop machine etc.). For example, the user may wish to watch a video but are currently mobile. Whilst in the office they may wish to stream the video to their desktop PC, but while walking outside of the building they want to see the video on their PDA device and finally, as they get into a taxi they wish to watch the video on a display screen inside.

Furthermore, applications will encompass more than one of these interaction types to provide their functionality. Therefore, individual middleware technologies supporting only one style of interaction are unsuitable for mobile applications.

### 3.1 Introduction

Middleware is defined as a layer of software residing on every machine, sitting between the underlying operating system and the distributed applications, whose purpose is to mask heterogeneity and provide a simple, consistent and integrated distributed programming environment [Coulouris et al, 00]. Different middleware platforms support different programming models [Coulson, 01a]. The most popular is *object-based middleware* in which applications are structured of distributed objects that interact via location transparent method invocation, for example CORBA and DCOM. Other paradigms are *event-based middleware* and *message-orientated middleware*, both of which utilise “single shot” communications rather than request-reply.

Middleware has proved a successful technology in wired communications, especially in the banking and finance domains, for overcoming the problems of distribution and heterogeneity. However, mobile applications, as described in the previous chapter, must adapt to constantly changing environments and therefore provide additional distribution problems. Hence, the task of writing adaptive applications is a significantly complex problem, where middleware can provide support by masking selected aspects of mobility and informing the application about the environment [Friday, 96]. Therefore, rather than making the features of distribution transparent they must be made aware to the mobile application in order for it to adapt. This chapter examines the range of currently available standard and adaptable middleware technologies and analyses their importance in supporting mobile computing.

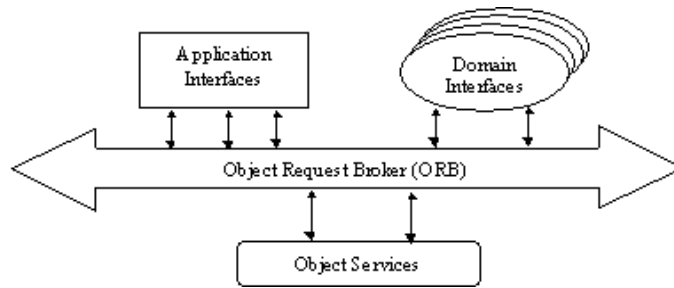
### 3.2 Base middleware platforms

#### 3.2.1 Common Object Request Broker Architecture (CORBA)

The Object Management Group ([www.omg.org](http://www.omg.org)) has defined a standard architecture called CORBA [OMG, 95] to address the problems of developing portable distributed applications for heterogeneous systems. CORBA provides an *object model* and a *relational model* to specify how distributed objects interact. Within the object model, an object is an entity that provides one or more services that can be requested by a client through well-defined interfaces. These are defined in IDL, the Interface Definition Language, which defines object interfaces in a manner independent of any programming language.

The relational model describes the categories of interfaces; figure 3.1 illustrates these categories that are conceptually linked by the Object Request Broker (ORB). *Object Services* are horizontally oriented interfaces used by many applications; for example, the OMG Naming Service provides references to objects that applications intend to use. *Domain interfaces* provide similar roles to object services, but are domain specific or vertically oriented e.g. healthcare applications. Finally, *application Interfaces* are developed specifically for newly created applications.

The main component of CORBA is the ORB (illustrated in figure 3.2); this allows clients written in one language to invoke the operations of remote objects written in another. The architecture allows both static and dynamic invocation of these requests. In the static approach, an IDL is translated into *stubs* and *skeletons* that are compiled into the application. A stub is a client side function that allows a remote invocation to be made via a local call. Similarly, the skeleton is a server side function that allows a request invocation to be received and dispatched to the appropriate object implementation. Dynamic invocation involves the construction of CORBA requests at run time, utilising the dynamic invocation and skeleton interfaces. The dynamic skeleton interface accepts requests for which it has no skeletons, inspects its contents and invokes the object and method it is targeted for.

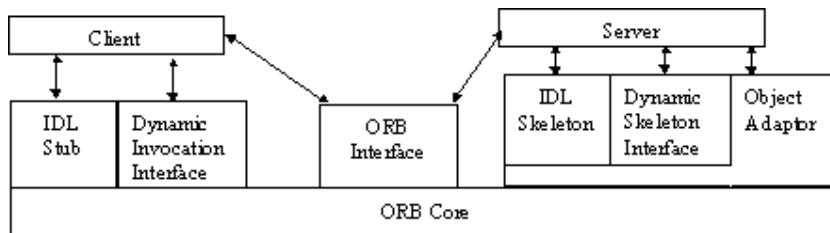


**Figure 3.1** Relation model specified by the overall CORBA architecture.

The OMG's newest standard (CORBA v3 [Siegel, 99]), includes three new categories:

- ❑ Improved Java and Internet integration, including a more interoperable name service.
- ❑ Quality of Service control, incorporating asynchronous messaging.
- ❑ A new component based architecture allowing applications to be composed of third party components.

These extensions have been made to cope with the increasing range of next generation application types such as multimedia.



**Figure 3.2** The Object Request Broker.

### 3.2.2 CORBA in a mobile environment

Employing CORBA in a mobile environment creates a number of problems due to hardware resources, hardware mobility and the characteristics of wireless networks [Haahr et al, 00]. Firstly, middleware implementations such as CORBA are too big to fit in devices with limited resources and they do not provide configuration tools that would allow adapting them to different architectures [Roman et al, 01]. Therefore, the large memory footprint size of CORBA is a fundamental obstacle. To overcome this, e\*ORB ([www.vertel.com](http://www.vertel.com)) and OrbacusE ([www.orbacus.com](http://www.orbacus.com)) are commercially available ORBs optimised for memory size and performance. However, they provide static configurations that cannot be changed at run-time. Whereas, more customisable ORBS e.g. UIC, which will be examined in detail later, optimise the memory footprint and also allow this to be changed over time.

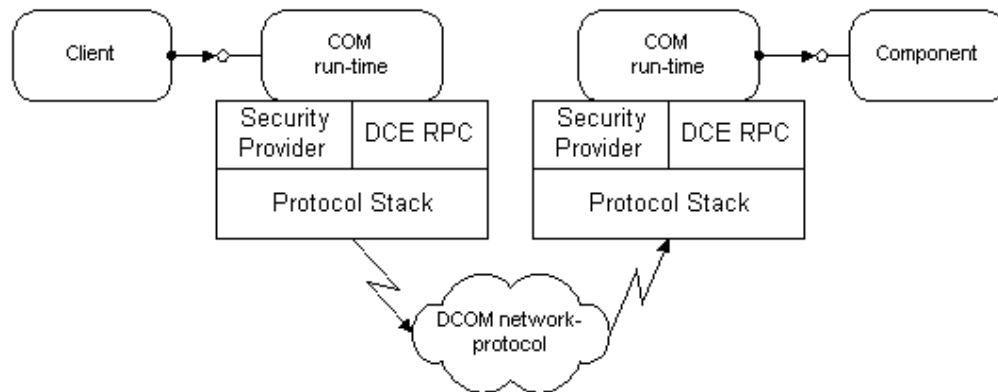
Furthermore, wireless networks also suffer from low bandwidth and periods of disconnection; whereas, CORBA specifies that a continuous connection is to be maintained throughout an invocation. There have been a number of research projects investigating how to improve CORBA implementations within a mobile environment presenting these properties. The Architecture for Location Independent CORBA Environments (ALICE) [Haahr et al, 00], defines a layered architecture

that takes into account movement of mobile hosts and ensures that client server connections remain established transparently. Alternatively, the DOLMEN project [Liljeberg et al, 97] defines a special Light-Weight Inter-ORB protocol (LW-IOP), which provides efficient message formats and compressed data representation for object communication over a wireless link. Notably, DOLMEN used CORBA bridging to deal with the introduction of the wireless link into the overall communications architecture. Furthermore, the RAPP system [Seitz et al, 98] presents a proxy-based architecture to allow proxies to be inserted between distributed CORBA objects and overcome the problems of disconnection and poor levels of network service. However, the impact of this critical problem is reduced by the availability of asynchronous messaging in the latest CORBA standard.

### 3.2.3 Alternative base middleware platforms

There are a number of other popular middleware platforms, which are used alternatively to CORBA. These will be examined briefly as follows.

DCOM [DCOM, 96] is an extension of the Component Object Model (COM) [COM, 95]. COM defines how components interact with their clients, whereby clients call methods in the component without any overhead. COM provides inter-process communication in a completely transparent fashion; it intercepts calls from the client and forwards them to the component in another process. When the client and the called component reside on different machines, DCOM simply replaces the interprocess communication with a network protocol. Figure 3.3 shows the overall DCOM architecture: The COM run-time provides object-oriented services to clients and components and uses RPC and the security provider to generate standard network packets that conform to the DCOM wire-protocol standard.



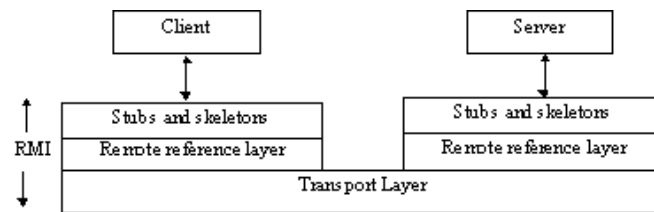
**Figure 3.3** DCOM: COM components on different machines

Initial work at the Technical University of Dresden developed a supporting architecture within DCE that enables applications to operate in a mobile environment. In the mobile DCE model [Schill et al, 95], a system is described in terms of a set of logical domains. Each domain consists of one or more sub-networks that contain mobile or fixed hosts. A domain manager in each domain accounts for all member hosts within the domain, provides brokerage services for the stations' services and resources and maintains a list of peer domains. Furthermore, domain managers periodically exchange hints about the services and resources offered by remote domains.

Java RMI [Sun, 97] provides a Java distributed object model that integrates into the programming language and local object model. Like CORBA, RMI is based on the separation of *definition of behaviour* (within a Java interface) and *implementation of that behaviour* (by a Java class). An advantage of RMI over CORBA is that RMI is based entirely on Java. This means that there is no need

to introduce a separate IDL. The overall implementation of RMI is shown in figure 3.4; it is a classical RPC style architecture, building on TCP as the transport protocol. Notably, Java 1.2 employs the concept of *reflection* to simplify the server side. It is used to implement a generic dispatcher at this end of the connection, replacing the need for skeletons.

Enterprise Java Beans [Monson-Haefel, 00] is a Java-based server side component architecture that enables and simplifies the process of building enterprise distributed object applications. A *software component* is code that implements a set of well-defined interfaces, cannot run alone and can be re-used in other applications to enable rapid application development. EJB provides a standardised way to build, manage and maintain these components, using *containers* that supply a run-time environment and a set of common services that most components need. EJB relies upon Java RMI as a communications method between components and their clients.



**Figure 3.4** Java RMI architecture

Alternatively, the JavaPod platform [Bruneton & Riveill, 00] is a middleware kernel for distributed component-based applications written entirely in Java. Its main goal is the transparent configuration of the non-functional properties of an application. The list of non-functional properties is not predefined (as in EJB) but is open and extensible. Therefore, it is more suitable than EJB to cope with the requirements of mobile applications and the varying non-functional properties they exhibit. Similarly, [Truyen et al, 00] present a reflective component architecture that allows ORBs to be built that can be dynamically configured to support different non-functional requirements on a per remote method invocation basis.

SOAP [Box et al, 00] is a lightweight, XML-based protocol for the exchange of information in a distributed environment. It consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. The latest specification adds flexibility that allows SOAP messages to be used for asynchronous and one-way message passing schemes. Notably, SOAP is simple and extensible; this means that several features of distributed object systems are not part of the specification. For example, distributed garbage collection, objects-by-reference and activation.

Fundamentally, SOAP provides a standard way of serializing the information needed to invoke remote services into a format that can be transported across the wire and interpreted by the remote service regardless of its platform (for example, avoiding problems of bridges needed to invoke CORBA object calls from a DCOM component). It also benefits from being able to traverse firewalls, as it is HTTP based.

### 3.2.4 Analysis of support for mobile computing

The major problems associated with mobility are location change, poor bandwidth, limited end-system resources, latency, changing environmental contexts and disconnection. Therefore, middleware must support adaptive mobile applications that can overcome these issues and create a new breed of applications that offer the best level of service to the user. The technologies described however, were

designed for fixed environments and provide little support for wireless environments. Research into CORBA has looked at overcoming these problems with support for wireless invocations and asynchronous communication, providing steps to solve a subset of the problems. However, CORBA's (and the other platforms') implementation remains too large for mobile devices and offers a high degree of transparency, providing no awareness of context to the application. Furthermore, Java RMI and EJB are language dependent (Java); however, within portable devices the virtual machine utilises a percentage of the available resources. Notably, SOAP provides the most lightweight solution for small clients; however, it doesn't provide distribution support, for example naming and locating objects.

However, one main reason that these platforms support mobile applications poorly is the black-box philosophy they utilise. In particular, a fixed service is offered to their users, and it is not possible to view or alter the implementation of this service. Inevitably, the architecture of this platform then represents a compromise design featuring, for example, general-purpose protocols and associated management strategies [Blair et al, 01]. Mobile applications exist in environments that are constantly changing, it is therefore difficult to provide a fixed middleware platform to support a range of adaptive applications and also overcome the problems of mobility within an implementation of suitable size.

### 3.3 Reflective Middleware

#### 3.3.1 Introduction

To overcome the problems of existing middleware platforms described in the previous section [Blair et al, 01] believe that future middleware platforms, for domains such as multimedia and mobile computing, should have the following properties:

- *configurable* to match the requirements of a given application domain,
- dynamically *reconfigurable* to enable the platform to respond to changes in its environment,
- they should support the *evolution* of the design of the platform as requirements change over time.

Recently, a group of *reflective middleware* technologies have emerged to meet these requirements. A reflective system is one that provides a representation of its own behaviour that is amenable to inspection and adaptation, and is *causally connected* to the underlying behaviour it describes. Causally connected means that changes made to the self-representation are immediately mirrored in the underlying system's actual state and behaviour, and vice-versa [Coulson, 01b]. The key to the approach is to offer a *meta-interface* supporting the inspection and adaptation of the underlying virtual machine. In terms of middleware, this implies that the meta-interface should support operations to discover the internal operation and structure of the middleware platform (e.g. protocols and management structures being deployed) and to make changes at run-time [Blair et al., 01].

The remainder of this section examines in detail three reflective middleware technologies and provides a brief overview of other similar techniques. These are later analysed and the benefits of adaptable middleware for supporting mobile computing are examined.

#### 3.3.2 Open ORB

Open ORB [Blair et al, 01] is a *reflective, component* based middleware platform developed at Lancaster University. They identify that existing middleware technologies (e.g. EJB) present a component based programming model to enhance configurability, re-configurability and re-use at the application level and extend this idea to the design of the middleware platform itself; where a component is defined as a "unit of composition with contractually specified interfaces, which can be deployed independently and is subject to third party creation" [Szyperki, 98]. Therefore, an instance

of Open ORB is a particular configuration of components, selected at build time and changed at run-time. The main features of the component model, which is designed to support multimedia programming, are:

- 1) Components are described by a set of required and provided interfaces.
- 2) Interfaces for continuous media interaction are supported.
- 3) Explicit bindings can be created between compatible interfaces i.e. where the defined required and provided interfaces match.
- 4) Components offer a built in event-notification facility.

Reflection is used to provide access to the underlying platform (component structure); every application-level component offers a *meta-interface* allowing access to an underlying *meta-space*, which is the support environment for that component. Meta-space is represented by four distinct meta-models. The *interface meta-model* provides access to the external representation of a component in terms of the set of provided and required interfaces. This allows a similar capability to the introspection facilities of the Java reflection API [Sun, 99], allowing a programmer to interact dynamically with a discovered component. It is not possible (as previously [Blair et al, 98]) to access the internal implementation of an interface (as a set of methods and attributes) in order to provide a clear separation between interface and implementation.

Furthermore, the *architecture meta-model* provides access to the implementation of the component as a software architecture, consisting of two elements: a *component graph* and an associated set of *architectural constraints*. The component graph is represented by a set of components connected by *local bindings*, where a local binding maps between a required and provided interface in a single address space. Distribution is added into the model by introducing (distributed) binding components into the graph. An extensible set of binding types can be supported offering interaction models such as remote invocation, publish/ subscribe, continuous media flows and group communication. Normally this structure is hidden from the user of a component. However, the architecture meta-model can be used to both discover and make changes to this structure at run-time.

Notably, Open ORB distinguishes between actions taking place in the system and the resources required to support the activity. The interception and resources meta-models represent these aspects respectively. The *interception meta-model* enables the dynamic insertion of *interceptors*. Such interceptors are associated with interfaces and enable the insertion of pre- and post- behaviour. This mechanism is useful, for example, to dynamically introduce monitoring or accounting into a running system [Wegdam, 00]. Interceptors can also be used to introduce additional non-functional behaviour, such as security checks or concurrency control. In contrast, the *resources meta-model* is unique to the Open ORB design, offering access to underlying resources and resource management [Duran, 00]. The resources meta-model is based around the abstractions of *resources* and *tasks*. Resources can be either primitive (e.g. raw memory or OS threads) or complex (e.g. buffers or user-level threads multiplexed on to kernel-level threads). Tasks are the logical unit of activity in the system; for example, there could be a single task dealing with the arrival, filtering and presentation of an incoming video stream. Therefore, tasks have a pool of resources that support their execution. There is a resources meta-model *per address space*, which provides access to a set of components representing resource management. As with other meta-models, it is then possible to either inspect or adapt activity associated with resources.

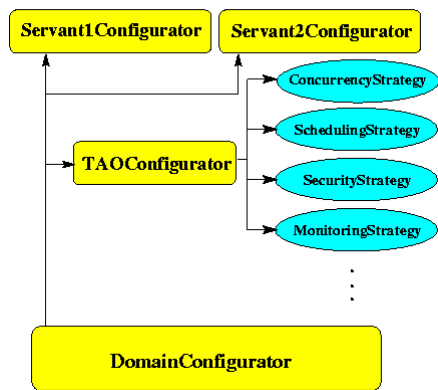
### 3.3.3. DynamicTAO

*DynamicTAO* [Kon et al, 00] is a reflective CORBA ORB built as an extension of TAO [Schmidt & Cleeland, 99]. Where TAO is a portable, flexible, extensible, and configurable ORB that conforms to the CORBA standard, which utilises a *Strategy* design pattern [Gamma et al, 95] to encapsulate different aspects of the ORB internal engine. TAO contains a configuration file that specifies the strategies the ORB uses to implement aspects like concurrency, request de-multiplexing, scheduling,

and connection management. When the ORB is initiated, the configuration file is parsed and the selected strategies are loaded. Notably, TAO is mainly used in static real-time applications.

However, *DynamicTAO* extends TAO to support on-the-fly reconfiguration; this is achieved by reifying *dynamicTAO*'s internal structure, i.e., keeping an explicit representation of the ORB internal components and of the dynamic interactions among them. The reification allows the ORB to change specific strategies without having to restart its execution. *DynamicTAO* is reflective because it allows inspection and reconfiguration of its internal engine; It achieves this by exporting an interface for (1) transferring components across the distributed system, (2) loading and unloading modules into the ORB runtime, and (3) inspecting and modifying the ORB configuration state.

Furthermore, reification in *dynamicTAO* is achieved through a collection of entities known as component configurators [Kon & Campbell, 00]. A component configurator maintains the dependencies between a component and other system components. Each process running the *dynamicTAO* ORB contains a component configurator instance called the *DomainConfigurator*, which is responsible for maintaining references to instances of the ORB and to servants running in that process. In addition, each instance of the ORB contains a customized component configurator called the *TAOConfigurator* that contains hooks to which implementations of *dynamicTAO* strategies are attached. Hooks work as "mounting points" where specific strategy implementations are made available to the ORB. Figure 3.5 illustrates the reification mechanism within a process containing a single instance of the ORB.



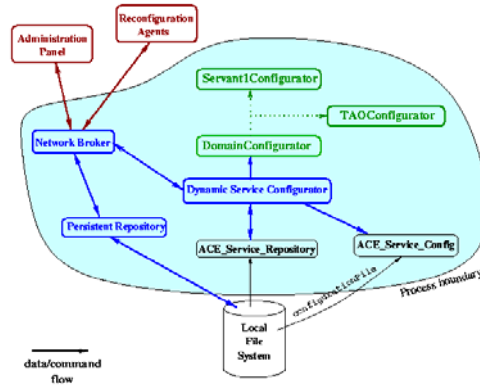
**Figure 3.5** Reifying the *dynamicTAO* structure [Roman et al, 01]

The *dynamicTAO* architectural framework is depicted in Figure 3.6. The *Persistent Repository* stores category implementations in the local file system. Once a component implementation is stored in the local repository, it can be dynamically loaded into the process runtime. A *Network Broker* receives reconfiguration requests from the network and forwards them to the *Dynamic Service Configurator*. The latter contains the *DomainConfigurator* (shown in Figure 3.5) and supplies common operations for dynamic configuration of components at runtime.

### 3.3.4 Universally Interoperable Core™ (UIC)

The Universally Interoperable Core (UIC) [Roman et al, 01] is a reflective middleware infrastructure designed to support ubiquitous computing environments, addressing the problems that other middleware platforms have in this area. The main scenario it supports is that of interaction with multiple service platforms from a mobile device, i.e. the ability to talk to remote light switches offered as CORBA services and other similar service based offerings implemented in Java RMI and SOAP.

UIC, like other reflective platforms, is implemented as a collection of components. Fundamentally, it provides a skeleton of abstract components that form the base of the platform. To enable the system to have the properties of particular middleware platforms (e.g. CORBA), devices and networks dynamically loadable components are added to specialise these abstract components.

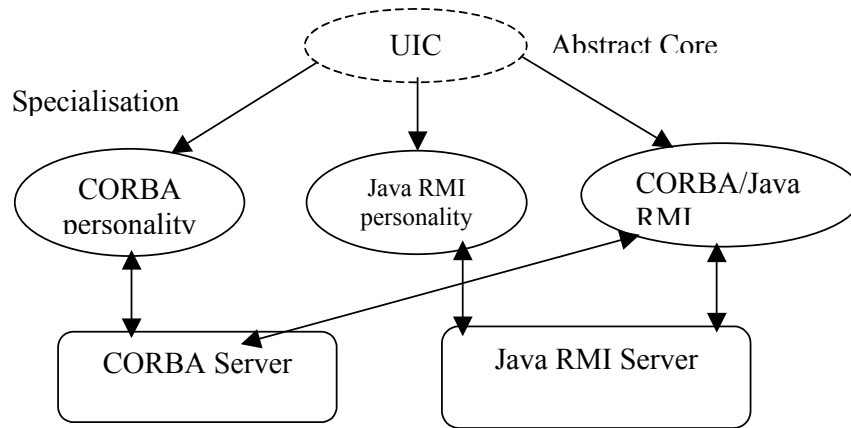


**Figure 3.6** dynamicTAO Components [Roman et al, 01]

The design of the platform is driven by the principle of *What You Need Is What You Get* (WYNIWYG) [Singhai et al, 97]; that is, only the required functionality is present in an instance of the UIC. The designer identifies that the majority of existing middleware platforms contain all possible functionality, even if the application only uses a subset; this is not suitable for devices with limited resources. Therefore, UIC provides only the minimum required functionality to guarantee interoperability with existing middleware platforms. Whenever more functionality is required, it is introduced either statically (rebuilding the UIC) or dynamically (adding the new functionality at run-time). While the UIC defines a standard skeleton structure targeted to object-oriented request brokers (CORBA, Java RMI, and DCOM), it also allows the application developer to change the structure to meet other requirements (e.g., non object-oriented RPC platforms and RTP streaming).

As defined, the UIC is an abstract entity that has to be specialised to meet the requirements of each application. This specialisation defines the behavior of the core and the type of entities with which it will be able to interact (e.g., CORBA servers or Java RMI servers). A UIC *personality* is a particular instance of the UIC obtained after the specialization as illustrated in figure 3.7. Personalities can be classified as client-side, server-side or both. The UIC can also be classified as single-personality or multi-personality. A single-personality interacts with a single middleware platform, while multi-personality UICs can interact with more than one platform at the same time. With multi-personality UICs, applications always use the same UIC instance and the same method invocation interface regardless of the type of the remote object; in this case, the UIC is responsible for automatically choosing the right personality.

UIC personalities can be configured either statically or dynamically. In static configurations, personalities are built at compile time by statically assembling all the components together. The result is a single component (the personality) that cannot be dynamically reconfigured. The main benefit of this configuration is the size of the personality. For example, a client-side CORBA personality for the Palm Operating System is 16K. In dynamic configurations, personalities are a collection of dynamically loadable libraries that can be fully reconfigured at run time. The main benefit of the dynamic configuration is the ability to modify the architecture of the personalities dynamically without affecting the applications. The main drawback of the dynamic personality is that the size of the core increases, because tools for loading and unloading components are required and each component of the core becomes an independent dynamically loadable library.



**Figure 3.7** UIC Personalities [Roman et al, 01]

### 3.3.5 Other Reflective Middleware Technologies

There exists a number of other configurable middleware platforms based upon the ideas of reflection; these include FlexiNet, K-ORB, OpenCorba and Multe-ORB, but this is not an exhaustive list. With the exception of K-ORB these platforms were not designed specifically for mobility but for wired application types and therefore, suffer from the problems of exhausting resources through their implementation. A brief overview of these platforms is given as follows.

The FlexiNet platform [Hayton et al, 98] is a Java based middleware platform providing a component-based approach to distributed application development, which emphasises the use of reflection within the protocol stack. There are four key elements of the FlexiNet architecture: software components, transparent component binding, policy definition, and automated deployment.

The component model is based upon bindings between components, so that a programmer may locate one from another. Components may pass references between each other in a transparent way; in these circumstances, FlexiNet associates the implicit binding request with the relevant policies and ensures that the constructed binding respects these policies. Notably, a reflective protocol stack is related to the binding to carry out the call process. FlexiNet provides a layered protocol stack, in which the layers can be viewed as reflective meta-objects that manipulate an invocation using Java Core Reflection [Sun, 99]. Reflection allows the component to have an open implementation; depending on what is required, the component can adapt itself by adding or removing sub-components that provide a degree of functionality. This means that rather than altering a stack of micro-protocols, the more complex layers of the FlexiNet architecture adapt themselves to changes in the environment.

K-ORB is the general term used to describe a family of customised and configurable ORBs that are based on the minimumCORBA specification [OMG, 98] being developed at Trinity College, Dublin. K-ORBs are instantiations of a minimumCORBA (a subset of OMG's CORBA 2.2 specification that is targeted at resource constrained environments) framework that allows developers to build ORBs for domains such as embedded systems, PDAs, intelligent devices and real-time systems. The K-ORB framework is an extension of the Mobile IIOP Engine developed in the Alice Project [Haahr et al, 00].

The K-ORB framework also allows developers to build ORBs where the environment and set of resources available to the ORB are subject to change at runtime. Mobile devices, for example, use the dynamic reconfiguration of the network protocol to select the most appropriate underlying network transport at runtime. For example, when a PDA with a GSM modem disconnects from the fixed network (an Ethernet or Wireless Ethernet connection), its transport protocol is dynamically

reconfigured to Mobile IIOP (TCP/IP over GSM) to enable CORBA clients and servers on the PDA to recommence communication with hosts on the fixed network. Similarly, when a PDA with a GSM modem connects from the fixed network, its transport protocol is dynamically reconfigured from Mobile IIOP to IIOP to provide higher bandwidth connections for CORBA clients and server on the PDA.

OpenCORBA [Ledoux, 99] is a reflective open broker that allows users to dynamically adapt the representation and execution policies of the software bus (ORB). It provides two mechanisms to do this: meta-classes and a protocol that enables the dynamic changing of meta-classes at run-time, making it possible to change the properties of a class. This allows the introduction of new semantics to the model such as security and replication.

MULTE-ORB [Kristensen & Plagemann, 00] is a reflective multimedia object request broker. The main programming models behind MULTE-ORB are explicit stream bindings, stream interfaces and flows. A binding type identifies the type of stream interfaces that participate in the binding, where a stream interface consists of source and sink media flows. Reflection is provided through reification of the composition of the binding, making it available for inspection and adaptation through meta-object protocols. Furthermore, Quality of Service management is supported by a set of components monitoring the behaviour of the system and binding controllers that encapsulate user policies for reconfiguration and adaptation.

### **3.3.6 Analysis of Reflective Middleware for Mobile Computing**

The use of reflection to create adaptable middleware platforms has found significant success in supporting multimedia applications. Furthermore, the identified benefits such as reconfiguration to maximise resource use and adaptation to support fluctuating levels of service are in accordance with the recognized needs of mobile computing. However, there is little evidence of the use of this technology within the domain to concretely illustrate these benefits. Only, UIC details attempts to fit itself to the mobile domain by examining the problems of service interaction. Questions still remain over the benefits of reconfiguration of the platform within mobile computing, where all aspects of the environment change and reconfiguration may be frequent.

## **3.4 Conclusions**

This section has presented the area of middleware and the currently available support required for the identified problems when developing mobile applications. The use of base middleware platforms such as CORBA and Java RMI are infeasible because they suffer from fixed, synchronous interaction and attempts to overcome this either via added asynchronous or other techniques still leave them hindered by their large static implementations that do not fit well with mobile devices in constantly changing environments. This has led to the proposal of adaptable middleware platforms to improve on size and support for environment change, based upon the technique of reflection.

Reflective middleware offers the most promising method of providing mobile support at the middleware level. The ability to provide a dynamic service is fundamental, as interaction with hosts and services of different types is the key feature of mobile computing. Presenting a fixed platform for this is infeasible due to the limited resources of mobile end systems. Furthermore, dynamic adaptation helps overcome the other issues of mobile computing such as poor levels of network service; that is, the platform can adapt to best support an application in the current context.

### 4.1 Introduction

The majority of middleware platforms described in the previous section provide a synchronous, RPC-style communication paradigm (e.g. CORBA, DCOM, DynamicTAO). This inherently requires that the client and server must co-exist in time in order for the complete process to be carried out. However, mobile networks are susceptible to bit error rates, network partitioning and periods of disconnection; therefore, the premise of synchronous invocations is not well suited to mobile environments. This led a number of researchers to attempt to overcome the problems of RPC using techniques such as modifying the RPC protocol [Davies et al, 94][Joseph et al, 95], transparent service rebinding [Davies et al, 94] and adaptation of the communication stream [Fitzpatrick et al, 98].

Although successful in overcoming the problems, solutions based around RPC are deemed inelegant and subject to significant adaptation [Wade, 99]. An alternative approach identified is to use the opposite of synchronous communication, i.e. asynchronous communication that allows the host to communicate while not coupled in time and space. A number of middleware platform exhibiting asynchronous properties have been produced based on paradigms such as tuple spaces, agents, messages and events. Key platforms within these paradigms are discussed as follows and their support for mobility is analysed.

### 4.2 Tuple spaces

#### 4.2.1 Introduction

The tuple space is a well-established asynchronous communication model, which is effectively a shared distributed memory spread across all participating hosts. To communicate, hosts submit *tuples* and *anti-tuples* to the tuple space. Tuples are typed data structures and are comparable to objects in languages like C++; to be altered they must be removed from the space, changed and then re-inserted. However, anti-tuples capture requests seeking to remove or copy data from a space; they contain a template against which to match tuples. [Wade, 99] simplifies this by identifying anti-tuples as questions and tuples as answers. Tuple spaces offer two key properties: temporal and spatial decoupling. Hosts can communicate through the space without being online at the same time or attached by an explicit binding.

The remainder of this section examines three tuple space platforms: L<sup>2</sup>imbo, JavaSpaces and T-Spaces. The properties they offer are detailed and their suitability for mobile computing discussed.

#### 4.2.2 L<sup>2</sup>imbo

The L<sup>2</sup>imbo platform [Davies et al, 98] is based upon the classic tuple space architecture but includes a number of extensions for operation within a mobile environment. It allows multiple tuple spaces to be created and used, removing the need for all operations to go through a central global tuple space on all machines; this is an important factor in an environment where communication links are unreliable. Furthermore, QoS attributes can be added to tuples, including delivery deadline so that mobile multimedia applications can be supported; this also allows the system to re-order to make best use of network connectivity. Finally, a number of agents for monitoring QoS and managing the creation of tuple spaces and propagation of tuples between tuple spaces are provided. The monitoring agents are injected into a management tuple space and watch characteristics such as connectivity, cost and power. A further advantage provided is that separate hosts can find out the QoS conditions of another host.

Incorporating the results of the monitoring  $L^2$ imbo provides a number of techniques for adaptation. The main one is the filtering agent, which controls the propagation of tuples between spaces. For example a filtering agent can act between two spaces dealing with MPEG video frames and only transmit I-frames, or by performing colour reduction on the I-frames.

### 4.2.3 JavaSpaces

A JavaSpace [Waldo, 98] is a tuple space embedded within the Java run time environment. JavaSpaces allows processes to co-ordinate their activities by exchanging objects through a shared space. A process can write new objects into the space (using `write` command), take objects from space (`take`) and copy objects in a space (`read`). When taking or reading objects, processes match based upon the values in the fields of a template and find the object that matters; if a match isn't found immediately then the process can wait until one arrives (`notify`).

### 4.2.4 T-Spaces

T-spaces [Wyckoff et al, 98] is a tuple space implementation developed by IBM; it is implemented in Java and is therefore portable to a range of platforms. Like  $L^2$ imbo it supports a multiple tuple space model in which each space exists as a single, centralised entity. It is further suited to mobile devices by providing a small memory footprint. Notably, it offers a range of services such as URL based file transfer, database access, event notification and group communication. Applications can also introduce operators that access the tuple space using new mechanism as the platform provides a dynamically extensible API.

## 4.3 Agents

### 4.3.1 Introduction

Agents can commonly be defined as “software beings” acting on behalf of a human user [Johansen et al, 95]. Different types of agents exist, i.e. a stationery agent that resides on the same host throughout its lifetime, while working on behalf of the user e.g. connecting to ftp sites or browsing through URLs. Other agents are mobile, transferring state from machine to machine; these are defined formally as *mobile agents* i.e. objects that have behaviour, state and location.

The agent paradigm presents a number of motivating factors for being suitable to supporting distributed applications within a mobile environment. Firstly, the client (or a broker on behalf of the client) can move to the server and perform all communication locally before returning; with bandwidth scarce this limits the interaction over the wireless link. Secondly, the reduction in communication across the wire limits the possibility of failure due to partition or disconnection; the agent may even monitor the status and control its mobility between hosts. Thirdly, mobile clients have limited end system resources; agents overcome this by moving the logic and communication processing to a more powerful server host.

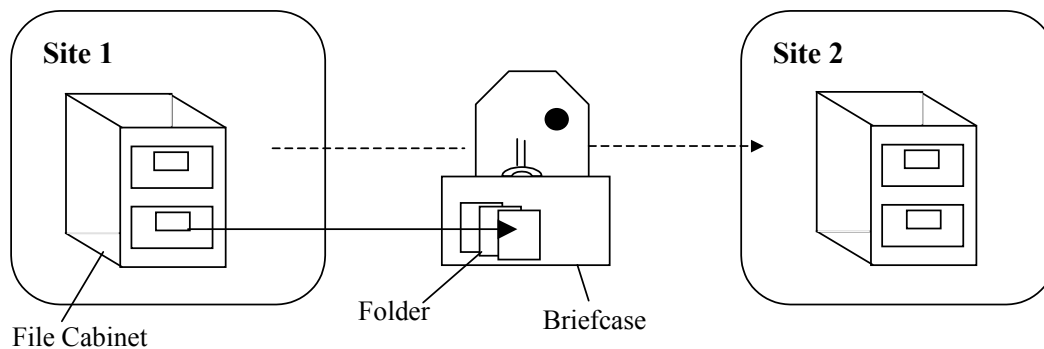
The remainder of this section identifies the properties of two agent platforms and examines how they support mobile applications.

### 4.3.2 Tacoma and Tacoma Lite

Tacoma [Johansen et al, 95] focuses on operating system support for agents and how agents can be used to solve problems addressed by other distributed computing paradigms. The Tacoma model makes no distinction between client, servers and agents; instead they are all identified as agents that

have a variety of properties. Some agents provide services, while others provide no service. Agents can remain static at a single node or move about. Alternatively, some agents act on the behalf of others. From a low level perspective the agent is just a process (code and state), with mobility as its primary characteristic.

In order to maintain state, agents must be able to manipulate data, i.e. leave data at a site and carry data when it moves. For example, an agent visiting multiple sites, with each site completing part of an overall computation needs to carry the sub results along with it when it leaves. To support this, Tacoma introduces the concept of *folders*, *briefcases* and *cabinets*, which are illustrated in figure 4.1. Folders contain data and code (including the source code of the agent) relevant to different computations. A collection of folders associated with an agent is known as a briefcase (the agent will carry a briefcase while moving about). Furthermore, stationary folders are needed for permanent data repository purposes; therefore, file cabinets hold folders at individual nodes.



**Figure 4.1** Maintaining state in the Tacoma model using folders, briefcases and file cabinets

The fundamental property of the architecture is the *meet* abstraction; agents do not communicate by exchanging messages they simply meet at the same location and exchange briefcases. For example, a client may request a service from a system agent by passing it a briefcase containing a service specification, the system agent then returns the service result in another folder. Using this model Tacoma agent applications are not limited to a single paradigm such as client-server; this is because the system can model other paradigms such as agent sequences, remote activation, client-server RPC, event driven programming and parallel processing [Johansen et al, 95].

In order to explore the properties of mobile agents within the domain of mobile computing, the Tacoma Lite system [Jacobsen & Johansen, 97] was developed to provide the properties of Tacoma for Windows CE based devices. To illustrate its capabilities a number of applications were developed, including a weather alarm (to alert user when certain conditions arose) and a stock ticker, which periodically checks stock prices for the user. Tacoma Lite adds an extra layer to the initial TACOMA architecture. This layer consists of an entity called the hostel that acts as a network proxy for the PDA. It is needed because of the requirements of the applications identified e.g. the weather alarm, which assumes the presence of a host that mobile agents could inquire in case the PDA is not connected.

Finally, this system identifies two key benefits of agent use in mobile computing. Firstly, the system can overcome periods of disconnection by the mobile device without loss of information. Secondly, the amount of data transferred to and from device is reduced by intelligent use of agents (rather than downloading all stock prices, the agent finds only those required).

### 4.3.3 Aglets

Aglets [Lange & Oshima, 98] are *Java-based autonomous software agents* developed by IBM that extend the model of network-mobile code as used by Java applets. Like an applet, the class files for an aglet can migrate across a network. But unlike applets, when an aglet migrates it also carries its state. An aglet is therefore a running Java program (code and state) that can move from one host to another on a network [Venners, 97]. A Java aglet is also similar to an applet in that it runs as a thread (or multiple threads) inside the context of a host Java application. A disadvantage however is that an aglet requires a host Java application, an "aglet host," to be running on a computer before it can visit that computer; therefore, making it unsuitable for ad-hoc interactions of mobile devices.

Aglets provide mechanisms for communicating with one another to help build complex agent applications. However, an aglet must go through a proxy object to interact with an aglet, even if both aglets are in the same aglet host. This proxy allows an aglet to send a message, either synchronously or asynchronously, to another aglet. A Message object is supplied for this purpose; it carries a String to indicate the kind of message plus one other optional piece of data, either a String or one of Java's primitive types.

The two presented agent platforms are two of the key technologies in this domain. However, many other agent architectures have been developed, usually based upon the mobile code properties of the Java programming language. These include Jumping Beans ([www.jumpingbeans.com](http://www.jumpingbeans.com)), Concordia [Wong et al, 97] and Voyager (<http://www.objectspace.com/products/voyager/>).

## 4.4 Message-orientated and event based middleware

### 4.4.1 Introduction

*Message Orientated middleware* is a specific class of middleware that supports the exchange of general-purpose messages in a distributed application environment. This encompasses a range of paradigms including publish/subscribe and message queuing systems. They support data exchange and request reply in a synchronous and asynchronous manner. Message delivery is ensured using reliable multicast or reliable queues.

The synchronous model of invocation is often inadequate in certain scenarios, where applications in which asynchronously occurring events need to trigger an immediate system response [Bacon et al, 00]. For example, a credit card cancellation operation by a banking service must invalidate a stolen card immediately and notify all affected services automatically if a thief is using that card. However, frequent polling to learn whether events have occurred overloads communications and infrequent polling delays the response to individual events so that users perceive the application as sluggish and inadequate, or insecure. Therefore, the best solution in this case is to use asynchronous *event notification*, which supports a range of application types from group communication to alarm systems.

Furthermore, event based paradigms are well suited to mobility. Mobile users and computers detach from networked systems and reattach later at other locations; the detection of a mobile user is an event that may require a system response. Systems supporting mobility also have to react to frequent changes in the environment due to movement of mobile devices and users.

This section first looks at the properties of Message orientated systems and looks at some available middleware platforms of this type. Subsequently, a single event-based platform is examined.

#### 4.4.2 iBus (Message orientated)

The iBus [Maffeis, 97] system from Softwired AG is a commercially available Java Messaging Middleware system based on the publish/subscribe communication paradigm. iBus resides between the operating system and the application software allowing the efficient asynchronous transfer of messages or Java objects between communicating parties.

iBus operates through an “information push” between producers and consumers of information. [Kruthoff, 99] identifies the three main components of the iBus system as:

- Producer: a component that feeds information (Java objects) into a channel.
- Consumer: a component that receives information from a channel.
- Channel: a component that moves information between the producer and consumer components.

An iBus *channelURL* specifies the channel; it is made up of a QoS String and a destination address. The QoS String defines a Quality of Service for the channel, which allows the use of reliable or unreliable IP multicast, TCP, HTTP or wireless protocols. Depending on the data being transmitted the QoS can be adjusted by the developer e.g. if low jitter is required in the case of video data, a less complex QoS would be defined.

Examples of other available Message Orientated Middleware (MOM) are xmlBlaster ([www.xmlblaster.org](http://www.xmlblaster.org)), which is CORBA/XML based for use in a variety of languages: Java, C, C++, Perl, Python and others. Alternatively, Gryphon [IBM, 98] is a *content-based* publish/subscribe system as opposed to the previous *subject-based* systems. In these subject-based systems, each message belongs to one of a fixed set of subjects (also known as groups, channels, or topics). For example, a subject-based publish/subscribe system for stock trading may define a group for each stock issue; publishers may post information to the appropriate group, and subscribers may subscribe to information regarding any issue. Alternatively, content-based systems support a number of information spaces, where subscribers may express a "query" against the content of messages published.

#### 4.4.3 Cambridge Event Architecture (CEA) – (event-based)

The Cambridge Event Architecture (CEA) [Bacon et al, 00] provides asynchronous operation by means of events, event classes, and event occurrences. CEA follows a publish-register-notify paradigm with event object classes and source-side filtering based on parameter templates. It incorporates standard platform technology: IDL for publishing events and automatic stub generation for event notification. Asynchronous notification allows a system to respond immediately to events; for example, the detection of a mobile user or the withdrawal of an individual’s access rights.

Toolkits that help manage event-based systems are normally developed on a system-specific basis. In JavaBeans, listeners register interest with an event raiser. However, JavaBeans is a proprietary, single-language system, and, when a distributed implementation is required, notification is built above synchronous Java RMI. In contrast, CEA is an asynchronous, language-independent, inter-application platform in an open distributed world.

In CEA, an object has a *register* method in its interface, and interested parties can register interest in any event class. Furthermore, access control is performed at event registration; the service does not allow a client without appropriate authority to register, and events that the service will notify are subject to restriction. When an event occurs, the service matches it against a stored template associated with each registration and each client whose template matches is notified of the event.

In addition, application developers can define intermediate services, called *event mediators*. A mediator can prevent a mobile user from missing events of interest while disconnected from the networked systems. The mediator registers interest with the required event sources on behalf of the mobile client and buffers the event notifications it receives from these sources. It also registers interest in the mobile client's location, and notification of an *attach* event (detecting the mobile user) triggers delivery of the accumulated events to the user at the new location.

CEA is one of a number of event based middleware platforms currently being researched. Other event-based platforms include Echo [Eisenhauer et al, 00] and JEDI [Bricconi et al, 99].

#### 4.5 Other important systems

Besides the described systems in the area of asynchronous middleware solutions for mobile computing, many others have been developed that concentrate on particular aspects of mobility. These are older platforms, but are documented as follows for completeness.

Rover [Joseph et al, 95] is a toolkit developed at MIT, which is designed to support the development of applications that execute on mobile computers. The toolkit provides queued remote procedure calls (QRPC) to applications. This allows applications to continue making remote procedure calls asynchronously while physically disconnected from the network. The RPCs are written to a stable log to await reconnection or for the cost of communication to fall below a threshold level before being executed.

Coda [Satyanarayanan et al, 90] is a file system for large-scale distributed computing environments. It overcomes server and network failures through two complementary mechanisms: server replication and disconnected operation. Server replication stores copies of a file at multiple servers, while disconnected operation offers caching sites that assume the role of a replication site. These properties are particularly useful for supporting mobile devices. Odyssey [Satyanarayanan, 96] is a successor to CODA that was designed to support mobile information access applications. It assumes these applications access and update data stored on remote servers.

Similarly, XMIDDLE [Mascola et al, 01] allows mobile hosts to share data when they are connected or replicate the data and perform operation on them when they are disconnected; reconciliation of data occurs when the hosts reconnect. XMIDDLE allows each device to store its data in a tree structure and communication is performed through the sharing of trees.

#### 4.6 Analysis and conclusions

The advent of a plethora of asynchronous middleware technologies has been clearly defined by the need for non-synchronous application areas that are non-critical. The described platforms identify that a number of researchers believe this is the most suitable technology for mobile computing.

Firstly, tuple spaces provide a method of both overcoming the limited network capabilities of mobile computing and limiting the adaptation that is inherent in synchronous communication over wireless links. However, the tuple space paradigm is not a well-known distributed programming model. Having its origins in the parallel computing community, many applications programmers would have to learn the model for it to be widely accepted as a mobile computing paradigm, though its emergence within the Java community will ease this problem. Further arguments against tuple spaces are that they do not support context awareness well; due to the flat data structures not allowing complex data organisation [Capra et al, 01]

Agents again provide methods to overcome the problems of synchronous communication and also provide natural techniques to create mobile applications. However, they suffer from requiring high

levels of security that add to resource use and are also not particularly suited to execution upon a mobile device limiting the effectiveness of the platforms for ad-hoc communities.

Systems designed in an event-based architectural style are particularly well suited for distributed environments without central control, to construct component oriented systems, and to support applications that must monitor or react to changes in the environment, information of interest or process status. Therefore, the event-based paradigm is especially suited to the mobile domain.

It can be concluded that these asynchronous technologies offer a number of important techniques in supporting mobile applications. Therefore, discounting their use is unwise; an important area of research is to examine the incorporation of their techniques into the middleware platform, when they are required by certain applications.

### 5.1 Introduction

The use of middleware from the classifications described in the previous sections is not the only method of supporting mobile applications. Notably, the telecommunications industry has developed standards for allowing access to network services to simplify the task of creating intelligent services for mobile applications. In addition, they have introduced environments for programmers to develop client side applications that execute on mobile devices. These technologies are described in the next section of this chapter; although they are not middleware-based solutions, they offer insight into platform improvements that can be made for the mobile world.

Furthermore, the issue of context is paramount to mobile computing. However, few of the previously described platforms employed techniques to support the full range of context (environment, service, hosts etc) that an application must react to. Therefore, the second section of this chapter identifies platforms that have incorporated context support at the platform level.

### 5.2 OSA and MExe

The principle behind Open Service Access (OSA), which is a standard from the “Third Generation Partnership Project” (<http://www.3gpp.org>), is that of *Open Networking*, i.e. to enable applications implementing services to make use of the network functionality provided by third generation networks types, such as UMTS, and create the next generation of intelligent services. An example application that illustrates the benefits of this technique is a vending machine that accepts payment from a mobile phone device (i.e. the charge is placed on the phone bill). The vending machine provides the intelligent service; after the user makes a selection, the service accesses location information about the user using OSA and finds out which device is in the vicinity of the machine. The network is then accessed to subtract the charge from the mobile phone’s account and after doing so the goods are dispensed.

OSA consists of three parts: applications, framework and service capability servers. Applications are location-based and implemented in one or more application servers. The framework provides applications with a mechanism that enables them to make use of the service capabilities in the network; example framework functions are authentication and discovery. Finally, service capability servers are functional entities that can be distributed across one or more physical nodes; these abstract the underlying network functionality from the application designer; examples of features provided by service capability servers are call-control and user location.

Other prominent initiatives in the area of open networks are Parlay [Parlay, 01], which is similar in features to OSA and JAIN (Java APIS for Integrated Networks), which is a standard Java environment for creating and executing services in the telecommunications domain.

Alternatively, the aim of MExe (Mobile Execution Environment), another standard specified by the 3GPP, is to provide a standardised execution environment within user equipment, allowing applications to be deployed (dynamically) independently of any user platform. A system of classification of device has been developed that recognises the diverse range of current and future wireless technology and that a one size fits all approach is unrealistic. *MExe classmark 1* is based upon WAP enabled devices requiring limited input and output facilities on the client side. *MExe classmark 2* is based upon Personal Java for devices with more resources and supports more applications and flexible user interfaces. Finally, *classmark 3* is based upon J2ME CLDC.

MExe provides a range of scenarios in which application interaction takes place. Firstly, services can execute on remote servers or web pages to which the MExe client establishes a connection and

receives content. Alternatively, the application is downloaded on the client device; this can then be used (as a local browser) to interact with the remotely provided service (cf. Jini). The third possible scenario is that the service is downloaded to the MExe device. The user downloads the service they require from a remote server and configures it on the device; these services execute directly on the handset without relying on a remote server to support the service (e.g. a game). Finally, MExe handsets may wish to establish connections with each other to provide, receive and use services (examples are interactive games and calendar sharing).

Another notable feature provided within the MExe environment is the ability for user interfaces and content to be tailored to the user's preferences. That is, newly downloaded applications and content are tailored (using user profiles to specify information of this type) specifically to each user and their device. For example, user interface buttons can be placed in the same place on screen or shortcuts can perform the same tasks across different applications.

The work in Open Service Access indicates that improvements at the server side, providing services that are intelligent and can best utilise the resources offered by the network is fundamental to mobile computing services. However, the approach is platform dependant and its use will be limited to the telecommunications domain. However, MExe offers a number of benefits to mobile applications: dynamic interaction through downloadable applications, configurable content and interfaces and multiple scenarios for all types of mobile applications. It is likely the features it offers will become prominent in general mobile computing and not just be restricted to the telecommunications sector.

### 5.3 Context aware middleware

Context aware computing [Shilit et al, 94] is fundamental to mobile computing. In order for a mobile application to provide the best level of service to the user it must be aware of the environment it executes within, so that it can adapt to any changes. Context information includes:

- ❑ Physical location - the user's or device's current position, discovered using a range of positioning technology e.g. GPS.
- ❑ Relative location – what devices, users, hosts, services, hardware are close by.
- ❑ Device – e.g. processing power, battery power, plug-in modules, and system memory.
- ❑ Physical environment – e.g. network bandwidth and noise level.

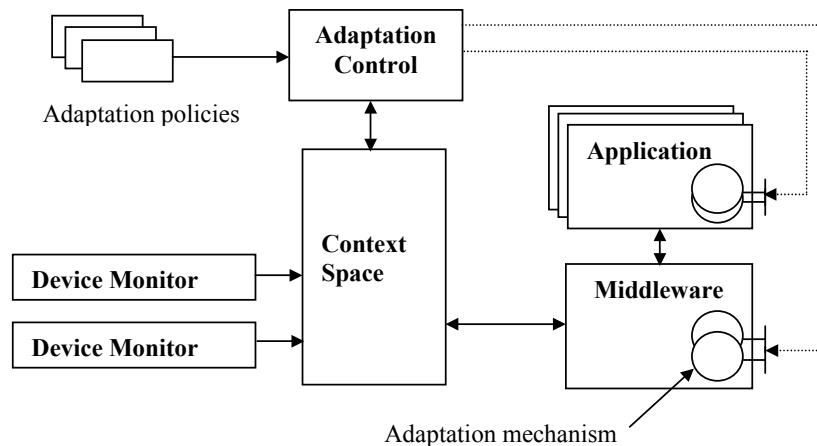
The Context Toolkit [Salber et al, 99] offered an initial method of supporting the development of adaptive context-aware applications. The Context Toolkit consists of context widgets and a distributed infrastructure that hosts the widgets. *Context widgets* are software components that provide applications with access to context information while hiding the details of context sensing. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns. The services of the Context Toolkit are: *encapsulation* of sensors, *access* to context data through a network API, *abstraction* of context data through interpreters, *sharing* of context data through a distributed infrastructure and *storage* of context data.

Similarly, initial mobile systems (applications) that collected context information mainly concentrated on location e.g. Cyberguide [Long et al, 96] and Guide [Davies et al, 99]. These interact with the underlying network operating system to obtain location information for the user. Therefore, they are not able to cope with heterogeneous position information. For example, a device may use a range of positioning technologies including GPS and radio frequency technology. To overcome this problem, a number of context-based (location) middleware systems have been developed that integrate a collection of positioning technologies with a common interface e.g. SignalSoft, [SignalSoft, 00] and Nexus [Fritsch et al, 00].

However, it has been identified that applications need to adapt to multiple contexts and this in turn presents additional problems: adaptive behaviour triggered by one attribute can cause side effects for

other attributes; these can in turn create conflicting actions e.g. a request to reduce power consumption enforces applications using the network to postpone their activities, as a consequence the network bandwidth increases and this could trigger a request to applications to utilise the spare bandwidth (i.e. the two are in conflict). Therefore, as well as supporting multiple context types co-ordination between adapting mechanisms is also needed [Efstratiou et al, 01].

Researchers at Lancaster University have presented an architecture [Efstratiou et al, 01] that provides a common space for co-ordinated system wide interaction between adaptive applications and a complete set of context attributes. The system they present decouples adaptation policies and mechanisms; figure 5.1 illustrates the proposed architecture. The *context space* acts as a repository for context information, storing information from the device monitors, applications and middleware for use by adaptation strategies. The *adaptation control module* is a key component of the architecture driven by a set of adaptation policies; it is responsible for co-ordinating adaptations and resolving potential conflicts. Furthermore, it is identified that the decoupling property of the architecture allows it to be integrated with a range of existing platforms such as the event-based, tuple space and object-based middleware described earlier.



**Figure 5.1** Architecture to support adaptive applications

It is clear that multiple context information must be supported for mobile applications and that the resolution of conflicts is an important research issue. However, the proposed architecture presents additional overhead that may not be suitable for all mobile devices. Devices that only execute a single application (e.g. Palm OS) do not need to resolve conflict; therefore, the architecture may waste valuable resources.

## 5.4 Conclusions

The vision provided by dynamic application environments such as MExe offer an insight into the future of mobile computing. It is unlikely that users will have a fixed set of applications on their device that are installed previous to motion. Instead users will obtain and interact with services & applications as they discover them. Therefore, future mobile middleware platforms must investigate how to provide support for the interaction properties of the dynamic applications and also to support the development of the dynamic environments themselves.

### 6.1 Introduction

This chapter presents the proposal for the research work to be undertaken by the author for the remaining two years of the PhD course. The ideas follow closely the background material presented in the previous chapters, concerning middleware platforms that support the development of mobile applications. To motivate the work to be carried out, a set of scenarios are presented that illustrate the problems of mobile computing that can be supported by an appropriate middleware platform.

This report has identified that a mobile environment is heterogeneous in nature. Mobile hosts present a range of hardware and operating systems, upon which applications are developed. Existing middleware technology has provided a method to overcome this level of heterogeneity and allow distributed applications to be developed across device types. However, the emergence of more than one middleware type presents new problems; applications developed upon differing middleware platforms cannot interact easily without the appropriate bridging technologies; for example, a CORBA based client cannot invoke the operations provided by a DCOM service. In static distributed systems, such as banking systems, this is not important as the whole system can be designed and implemented upon the same platform. However, within a mobile environment and for mobile applications this is infeasible, because the platforms that will be interacted with cannot be predicted.

Fundamentally, mobile applications will interact with location-based services; however, due to the roaming nature of the interaction, the range of services encountered will exist upon differing platform types (e.g. CORBA, SOAP, RMI etc.), be registered or discoverable by differing methods (e.g. Jini, SLP, Salutation) and exist upon fixed and mobile hosts. For example, a mobile application requiring to print a document at a nearby printer must locate the required functionality (printing service) independent of the underlying technologies and invoke its operation. Similarly, mobile applications will interact in an ad-hoc nature with other mobile hosts, which may also exist upon different platforms and discovery protocols. For example, a group of mobile users should be able to interact with each other using individual chat applications developed upon different middleware platforms, e.g. interaction with a SOAP-based chat tool.

The structure of the chapter is as follows: Section 6.2 describes the main aims of the project and the specific objectives that must be met. Section 6.3 presents the overall approach that will be followed to undertake the project. Section 6.4 highlights a programme of work to be carried out to meet the aims.

### 6.2 Statement of aims

#### 6.2.1 Main aim

Mobile applications must be able to *discover* and *interact* with a range of service types (CORBA, RMI, SOAP etc.) that may be registered using a variety of service discovery technologies (SLP, Jini, Salutation etc.). Therefore, the main aim of the project is to develop a lightweight, dynamically re-configurable middleware platform to support this.

Therefore, this platform will consist of a re-configurable, architectural middleware service for service discovery. This service will present a standard interface to integrate the features offered by service discovery techniques, irrespective of the underlying technology. Re-configurability will be provided by means of reflection. For example, the platform may be configured initially to discover Jini and/or SLP services but altered later to just find SDP (Bluetooth) services; for example, this could be caused by a change in network connection from 802.11b to Bluetooth as the user roams.

Taking this further, the application needs to interact with the service after it has been discovered; this may be in the form of a CORBA or RMI invocation, a message-based interaction or other middleware paradigm. Therefore, the underlying platform will be designed to configure itself (again using reflection) to support new interaction methods and multiple types at the same time.

This platform will therefore conform to the following criteria:

- ❑ The middleware architecture will provide the application programmer with the ability to create mobile applications based upon the recognised types of mobile distributed systems i.e. nomadic, ad-hoc and presence-based.
- ❑ The platform will be dynamically configurable to support changes in communication type and overcome mobility problems (disconnection, poor network QoS etc.), while limiting the end system resources utilised.
- ❑ The middleware platform will be aware of environmental, device and user context in order to control adaptation of the underlying platform to best suit the current situation. Furthermore, the platform will support the presentation of context information to the application as a means of controlling adaptation at the application level.
- ❑ The platform will meet recognised performance criteria in comparison with existing middleware platforms in terms of resource use, configuration time and the level of service provided.

## 6.2.2 Specific objectives

In order to meet the primary aim the following specific sub goals must be achieved:

1. A key sub goal is to design and implement a reflective, component based middleware infrastructure suitable for mobile devices that allows dynamic adaptation of the platform. It must be lightweight in nature to overcome the limited resources of mobile devices and provide a level of performance to meet the required service expected by the user. By utilising reflection, the range of middleware communication paradigms is unconstrained; the platform will support synchronous RMI types, asynchronous messaging and others.
2. The second key sub goal is to design and implement the dynamic, architectural service discovery service. This will be built upon the reflective platform described above to allow the service to be re-configured to meet differing discovery technologies. These two key sub-goals will allow the application designer to implement applications that can interact with mobile services independent of the middleware platforms they employ.
3. A further objective is to support applications based upon ad-hoc distributed systems. That is, the platform must offer the ability to discover hosts physically close, be aware of their system properties, be aware of the corresponding user preferences and provide peer-to-peer, group-based interaction. Therefore, the aim is to extend the design of the “discover and interact platform” to support mobile host discovery, peer-to-peer interaction and group interaction irrespective of the underlying communication technology.
4. The final style of interaction, i.e. presence-based, which was described previously must be supported. When creating presence-based systems the application needs to be aware of a user’s current context rather than a device’s and be able to interact with the device currently most suitable to the user. The underlying architecture will be extended to provide the support at the middleware level.

The first two objectives illustrate the key goals of the project to create a middleware platform to support mobile applications. The next two objectives outline the secondary goals of extending the platform to be suitable for varying styles of mobile applications. However, in order to meet these goals and add further value to the project the following four objectives are identified:

5. The description of the intended platform has presented a scenario where independence of middleware is a key factor. However, this leads to the question of how does the application programmer interact with the platform to obtain the required functionality from unknown platform types. Therefore, rather than just providing RPC or publish/subscribe programming techniques the platform will offer a more descriptive solution. The objective is to investigate the development of an XML based language that allows the programmer to describe their platform and service requirements. To illustrate this a scenario is presented as follows. The application programmer needs to call a remote printing service, which prints a specified document to the nearest printer to the user. To do this, the programmer describes their requirements within the designed language. The platform then maps these requirements to first locate an appropriate service and then interact with it.
6. To implement the previous aims, the platform must be aware of context information to support *adaptation*. For example, the platform may be adapted when the network bandwidth decreases. Furthermore, an application must also be able to discover context information about the environment and platform. The reflective model will provide full awareness of the platform. However a method for additional context discovery is needed. Therefore, a further aim of the project is to integrate a context architecture to support the underlying middleware platform and act as an information provider to the application.
7. The architecture of the middleware platform will be component based. However, it is envisaged that a number of components may not always be needed (or are not active in the platform); for example, it is unlikely all of the components for all of the service discovery types will be needed at the same time. These components take up significant resources and it is infeasible to currently store them on the device. Therefore, the project will investigate the use of dynamic downloading of components to overcome this, presenting an architecture to support this. Furthermore, it is envisioned that future mobile applications will be available to download dynamically; therefore, the proposed architecture will be extended to deal with this and an investigation of how the overall middleware platform fits with applications of this type will be carried out.
8. Mobile applications execute in environments with limited resources. It is therefore essential that they make best use of the available resources. For example, battery power and network bandwidth may be scarce and the application does not want to waste them. Therefore, the project will also investigate *resource management* strategies for mobile computing and design an architecture to allow the user to control these depending upon their preferences.

These final objectives are not central to the goal of the project, rather they act to support the key issues.

### **6.3 Methodology and approach**

The methodology that will be adopted to complete this project will follow an incremental approach. The initial reflective middleware architecture will be designed and implemented to produce a prototype, which will be tested to guarantee it meets the required characteristics and performance levels. Furthermore, it will be ensured to be suitable to real mobile applications by utilising its support within examples of these. For example, this could take the form of a “follow-me sport news” application presenting the latest sport news information to the user independent of their location and mobile device; this would identify the nomadic and presence support offered by the platform. Furthermore, a mobile game application could illustrate the ad-hoc support that the middleware provides.

Following from the successful implementation of the initial prototype the design of the platform will be extended to include the reflective, architectural service discovery service. The prototype will then be extended to include the implementation of this. After successful testing of the combined properties, which create the “discover and interact” platform, the project cycle will continue in the incremental style to incorporate the remaining described features into the system e.g. context, resource management and dynamic component downloading.

## 6.4 Programme of Work

This section outlines the main tasks and sub-tasks required to meet the aims of the project. A timetable describing the completion of these tasks is given in figure 6.1. The work described begins at the start of the second year of the PhD (October, 2001) and describes the development process over the following 15 months.

The following list describes the individual tasks that must be undertaken in order to complete the project. There is a description of what the task involves, what the deliverable is and how long it will take.

**Task 1:** Design the underlying reflective component-based middleware for the mobile platform.  
**Deliverable:** The architectural design and documentation for the platform, which will be capable of dynamically re-configuring itself for interaction with a range of service types and paradigms.  
**Time:** 6 months.

**Task 2:** Development of a prototype implementing the design created in task 1.  
**Deliverable:** The implemented prototype capable of communicating with a range of service types and dynamically adapt between these. To illustrate this, the prototype will implement Java RMI invocation, SOAP invocation, publish/subscribe and agent interaction.  
**Time:** 4 months.

**Task 3:** Testing of the prototype implemented in task 2.  
**Deliverable:** The test results for the prototype, ensuring the correct functionality and level of service required by the platform are met. To further test the system, a mobile news application will be developed upon the prototype platform.  
**Time:** 2 months.

**Task 4:** Design reflective, architectural middleware service for platform independent service discovery.  
**Deliverable:** The extended architectural design and documentation of the middleware platform  
**Time:** 2 months.

**Task 5:** Develop service discovery middleware service.  
**Deliverable:** The extended prototype with an integrated middleware service, creating a platform that can discover and interact independently of the underlying platforms. Initial behavioural properties of the service will incorporate SLP, SDP and Jini functionality, although this may be extended if time permits.  
**Time:** 4 months.

**Task 6:** Testing of the new prototype.  
**Deliverable:** The test results for the prototype, which illustrate that the required functionality and level of service is provided within a suitably heterogeneous environment. Newly designed applications will be developed in order to utilise the platform facilities and identify its suitability for nomadic, ad-hoc and presence-based interactions.

**Time:** 2 months.

**Task 7:** Investigation and design of language for application programmer to use for independent platform interaction.

**Deliverable:** A designed and documented language, that is fully integrated within the underlying platform.

**Time:** 6 months

**Task 8:** Development of a context architecture to support adaptation.

**Deliverable:** The design documentation, prototype implementation and test results for a context architecture that supports the use of context at the application level to control the adaptation of the underlying middleware platform.

**Time:** 1 month.

**Task 9:** Development of dynamic component downloading support within the architecture.

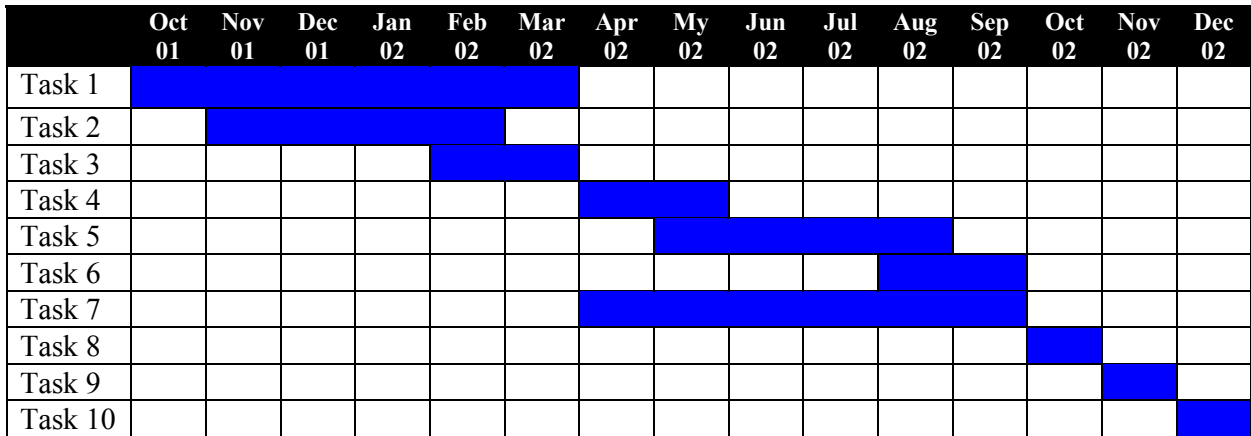
**Deliverable:** The design documentation and prototype implementation of extension to the middleware platform to allow components to be downloaded from remote hosts (fixed or mobile) during adaptation.

**Time:** 1 month.

**Task 10:** Development of resource management strategies for the middleware platform

**Deliverable:** An investigation into extending the prototype to control resources of limited mobile systems e.g. battery power and network bandwidth.

**Time:** 1 month.



**Figure 6.1** Programme of work

## Chapter 7 Conclusions

---

This report has presented the current state of the art in middleware for mobile computing. The problems with existing black-box middleware solutions has been described and the creation of adaptive and asynchronous technologies has illustrated possible methods to overcome these problems.

Furthermore, a number of middleware paradigms have been presented e.g. object based, tuple spaces, event based, agents and service discovery. However, limiting a platform to a single paradigm is unsuitable for mobile computing due to the range of application types available. This is illustrated by the adding of the event service and asynchronous messaging in the CORBA platform. The application designer must be allowed to utilise and integrate middleware paradigms to best support the application they are developing.

Finally, in the realms of mobile computing, context is a fundamental factor. If an application or middleware is to adapt then it must be aware of its current execution environment in order to provide the best level of service to the user. Furthermore, the emergence of new types of mobile applications based upon both traditional wireless services and ad-hoc device interaction require applications to be able to discover and interact dynamically with services, hosts and users within their current context. Therefore, mobile middleware must offer architectural support to simplify this task.

## References

---

- [Arnold et al, 99] Arnold, K., O'Sullivan, B., Scheifler, R., Waldo, J., and Wollrath, A. "The Jini Specification". Addison Wesley, 1999.
- [Bacon et al, 00] Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O. and Spiteri, M. "Generic Support for Distributed Applications", IEEE Computer, pp 68-76, March 2000.
- [Blair et al, 98] Blair, G., Coulson, G., Robin, P. and Papatomas, M. "An architecture for Next Generation Middleware", In Proceedings of Middleware '98, pp. 191-206, Springer Verlag, 1998.
- [Blair et al, 01] Blair, G.S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., Saikoski, K., "The Design and Implementation of OpenORB v2", To appear in IEEE DS Online, Special Issue on Reflective Middleware, 2001.
- [Bluetooth, 99a] "The Bluetooth specification", <http://www.bluetooth.com/developer/specification/specification.asp>, 1999.
- [Bluetooth 99b] Bluetooth Specification Part E. "Service Discovery Protocol (SDP)", <http://www.bluetooth.com/>, 1999.
- [Bricconi et al, 00] Bricconi, G., Di Nitto, E., Fuggetta, A. and Tracanella, E. "Analyzing the behaviour of event dispatching systems through simulation", In proceedings of 7th International Conference on High Performance Computing (HiPC 2000), India, December 2000.
- [Bruneton & Reveill, 00] Bruneton, E. and Riveill, M. "Javapod: an adaptable and extensible component platform", Workshop on Reflective Middleware, New York, USA, April 2000.
- [Box et al, 00] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S. and Winer, D. "Simple Object Access Protocol (SOAP) 1.1. Technical Report", <http://www.w3.org/TR/SOAP>, May 2000.
- [Capra et al, 01] Capra, L., Emmerich, W. and Mascolo, C. "Reflective Middleware Solutions for Context-Aware Applications", In Proc. of REFLECTION 2001- The Third International Conference on Meta-level Architectures and Separation of Crosscutting Concerns, September 2001.
- [COM, 95] Microsoft Corporation. "The Component Object Model Specification, Version 0.9", <http://www.microsoft.com/Com/resources/comdocs.asp>, October 1995.
- [Coulouris et al, 00] Coulouris, G., Jean Dollimore, and Tim Kindberg, "Distributed Systems, Concepts and Design", Addison-Wesley (3<sup>rd</sup> Edition), 2000.
- [Coulson, 01a] Coulson, G. "Introduction to Middleware", IEEE Distributed Systems Online, <http://computer.org/dsonline/middleware/index.htm>, 2001.
- [Coulson, 01b] Coulson, G. "What is Reflective Middleware?", IEEE Distributed Systems Online, <http://boole.computer.org/dsonline/middleware/RMarticle1.htm>, 2001.
- [Crow et al, 97] Crow B. P., Widjaja I., Kim J. G. and Sakai P. T., "IEEE 802.11 wireless local area networks," IEEE Communication Magazine, pp. 116-126, September 1997.
- [Crumley et al, 99] Crumley, G.C., Evans, N.E., Burns, J.B. and Trouton, T.G. "On the design and assessment of a 2.45 GHz radio telecommand system for remote patient monitoring", Medical Engineering and Physics, 20(10), pp750-755, 1999
- [Davies et al, 94] Davies, N., Blair, G. S., Cheverst, K. and Friday, A. "Supporting Adaptive Services in a Heterogeneous Mobile Environment", Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '94), Santa Cruz, California, U.S., IEEE Computer Society Press, pp 153-157, December 1994.

[Davies et al, 98] Davies, N., Friday, A., Wade, S. and Blair, G. S. "*L<sup>2</sup>imbo: A Distributed Systems Platform for Mobile Computing*", ACM Mobile Networks and Applications (MONET) - Special Issue on Protocols and Software Paradigms of Mobile Networks, 3(2), pp 143-156, August 1998.

[Davies et al, 99] Davies, N., Cheverst, K., Mitchell, K. and Friday, A. "*Caches in the Air: Disseminating Information in the Guide System*". Proceedings of the 2<sup>nd</sup> Workshop on Mobile Computing Systems and Applications (WMCSA '99), 1999.

[DCOM, 96] Microsoft Corporation. "*Distributed Component Object Model Protocol-DCOM/1.0, draft*", <http://www.microsoft.com/Com/resources/comdocs.asp>, November 1996.

[Dey & Abowd, 99] Dey, A.K. and Abowd, G.D. "*Toward a better understanding of context and context-awareness*". Gvu Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology. 1999. <<ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf>>

[Duran, 00] Duran, H. and Blair, G. "*A Resource Management Framework for Adaptive Middleware*", In 3<sup>rd</sup> IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'2K), Newport Beach, California, USA, March 2000.

[Eisenhauer et al, 00], Eisenhauer, G., Bustamente, F. and Schwan, K. "*A Middleware Toolkit for Client-Initiated Service Specialization*", Proceedings of the PODC Middleware Symposium, July 2000.

[Efstratiou et al, 01] Efstratiou, C., Cheverst, K., Davies, N. and Friday, A. "*An Architecture for the Support of Adaptive Context-Aware Applications*", Proceedings of Mobile Data Management (MDM 2001), Hong Kong, January 2001

[Fitzpatrick et al, 98] Fitzpatrick, T., Blair, G. S., Coulson, G., Davies, N. and Robin, P. "*Supporting Adaptive Multimedia Applications through Open Bindings*", Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDs '98), Annapolis, Maryland, U.S., May 1998.

[Forman & Zahorjan, 94] Forman, G.H and Zahorjan J. "*The Challenges of Mobile Computing*", *IEEE Computer*, 27(4), pp. 38-47, 1994.

[Friday, 96] Friday, A.J. "*Infrastructure Support for Adaptive Mobile Applications*", Ph.D Thesis, Computing Department, Lancaster University, 1996.

[Fritsch et al, 00] Fritsch, D., Klinec, D. and Volz, S. "*Nexus positioning and data management concepts for location aware applications*", In Proceedings of the 2<sup>nd</sup> International Symposium on Telegeoprocessing, pp 171-184, 2000.

[Gamma et al, 95] Gamma, E., Johnson, R., Helm, R. and Vlissides, J. "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley, 1995.

[Goland et al, 99] Goland, Y., Cai, T., Leach, P., Gu, Y., and Albright, S. "*Simple Service Discovery Protocol*", Internet Draft, draft-cai-ssdp-v103.txt, 1999.

[Haahr et al, 00] Haahr, M., Cunningham, R. and Cahill, V. "*Towards a Generic Architecture for Mobile Object-Oriented Applications*", SerP 2000: Workshop on Service Portability. San Francisco, December 2000.

[Hayton et al, 98] Hayton, R., Herbert, A. and Donaldson, D. "*Flexinet: a flexible, component oriented middleware system*", Proceedings of the 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications, Sintra, 1998.

[HomeRF, 01] "*HomeRF Working Group*". <http://www.homerf.org/>. 2001

[iButton, 01] "*iButton Homepage*". <http://www.ibutton.com/>. 2001.

[IBM, 98] IBM research. "*Gryphon: An Information Flow Based Approach to Message Brokering*", <http://researchweb.watson.ibm.com/gryphon/home.html>, 1998.

- [**IBM, 00**] IBM research. "Linux on a wristwatch". <http://www.research.ibm.com/WearableComputing/factsheet.html>. 2000.
- [**IrDA, 01**] "IrDA Serial Infrared Data Link Standard Specifications". <http://www.irda.org/>. 2001.
- [**Jacobsen & Johansen, 97**] Jacobsen, k. and Johansen, D. "Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code", In, Proceedings of the Symposium on Applied Computing (ACM SAC'99), February 1999.
- [**Johansen et al, 95**] Johansen, D., van Renesse, R. and Schneider, F. "An introduction to the TACOMA distributed system, version 1.0," Technical Report 95-23, University of Tromso, 1995.
- [**Johnson & Perkins, 96**] Johnson, D. and Perkins, C. "Mobility Support in IPv6". *ACM Mobicom 96*, ACM, pp. 2737, November 1996.
- [**Joseph et al, 95**] Joseph, A., deLespinasse, A., Tauber, J., Gifford, D. and Kaashoek, M. "Rover: A Toolkit for Mobile Information Access", Proceedings of the 15th Symposium on Operating Systems Principles (SOSP '95), Colorado, U.S., pp 156-171, December 1995.
- [**Katz, 94**] Katz, R.H. "Adaption and Mobility in Wireless Information Systems", IEEE Personal Communications, 1(1), pp. 6-17, 1994.
- [**Katz et al, 96**] Katz, R., Brewer, E., Amir, E., Balakrishnan, H., et al. "The Bay Area Research Wireless Access Network (BARWAN)", Proceedings of the Spring COMPCON Conference, 1996.
- [**Kon et al, 00**] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., and Campbell, R. "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", in Proceedings of the Middleware 2000 Conference, ACM/IFIP, April 2000.
- [**Kon & Campbell, 00**] Kon, F. and Campbell, R. "Dependence Management in Component-Based Distributed Systems", IEEE Concurrency, 8(1), pp.26-36, 2000.
- [**Kristensen & Plagemann, 00**] Kristensen, T. and Plagemann, T. "Enabling Flexible QoS Support in the Object Request Broker COOL", *Proceedings of International Workshop on Distributed Real-Time Systems (IWDRS 2000)*, April 2000.
- [**Kruthoff, 99**] Kruthoff, A. "Jini and Software bus systems". IFI, University of Zurich Technical Report. 1999.
- [**Lange & Oshima, 98**] Lange, D, Oshima, M., "Programming and Deploying Java Mobile Agents with Aglets", Addison-Wesley, 1998.
- [**Ledoux, 99**] Ledoux, T. "OpenCorba: a Reflective Open Broker", In 2<sup>nd</sup> International Conference on Reflection and Meta-level Architectures, St. Malo, France, July 1999.
- [**Liljeberg et al, 97**] Liljeberg, M., Raatikainen, K., Evans, M., Furnell, S., Maumon, K., Veldkamp, E., Wind, B., Trigila, S. "Using CORBA to Support Terminal Mobility", in: Proc. TINA '97, 1997.
- [**Long et al, 96**] Long, S., et al. "Rapid Prototyping of Mobile Context-aware Applications: The Cyberguide Case Study", 2nd ACM International Conference on Mobile Computing and Networking (MobiCom'96), 1996.
- [**Lucent, 01**] "MiLife Press release". <http://www.lucent.com/press/0301/010313.nsa.html>. 2001
- [**Maffeis, 97**] Maffeis, S. "iBus - the Java Intranet Software Bus", Technical Report, SoftWired AG, February 1997.
- [**Mascolo et al, 01**] Mascolo, C., Capra, L., Zachariadis, S. and Emmerich, W. "'XMIDDLE: A Data-Sharing Middleware for Mobile Computing", To appear in Personal and Wireless Communications.

- [**Microsoft, 01**] Microsoft Corporation. “*Universal plug and play device architecture*”, <http://www.upnp.org/>. 2001
- [**Mitchell et al, 00**] Mitchell, S., Spiteri, M. D., Bates, J. and Coulouris, G. “*Context-Aware Multimedia Computing in the Intelligent Hospital*”, In Proc. SIGOPS EW2000, the Ninth ACM SIGOPS European Workshop, Kolding, Denmark, September 2000.
- [**Monson-Haefel, 00**] Monson-Haefel, R. “*Enterprise Javabeans*”, O'Reilly UK (2<sup>nd</sup> Edition), 2000.
- [**Muratore, 00**] Muratore, F. (Ed) “UMTS: Mobile Communications for the Future”. John Wiley & Sons. 2000
- [**OMG, 95**] Object Management Group. “*The common object request broker: Architecture and specification*”, Tech. Rep. Version 2.0, July 1995
- [**OMG, 98**] Object Management Group, “*Minimum CORBA - Joint Revised Submission*”, OMG Document orbos/98-08-04 ed., August 1998.
- [**Pal et al, 97**] Pall, G. et al, "Point-to-Point Tunnelling Protocol--PPT," <ftp://ftp.ietf.org/internet-drafts/draft-ietf-pppext-pptp-02.txt>, July 1997.
- [**Palter et al, 97**] Palter, W. et al, "Layer Two Tunnelling Protocol 'L2TP,'" <ftp://ftp.ietf.org/internet-drafts/draft-ietf-pppext-l2tp-08.txt>, November 1997.
- [**Parlay, 01**] “*The Parlay Group*”. <http://www.parlay.org>.
- [**Perkins, 96**] "IP Mobility Support," C. Perkins, ed., IETF RFC 2002, Oct. 1996.
- [**Rahnema, 93**] Rahnema M. "Overview of the GSM System and Protocol Architecture", IEEE Communication Magazine, 31(4), pp. 92-100, April 1993.
- [**Roman et al, 01**] Roman, M., Kon, F. and Campbell, R. H. “*Reflective Middleware: From Your Desk to Your Hand*”, IEEE DS Online, Special Issue on Reflective Middleware, 2001.
- [**Ryan, 98**] Ryan, N.S., Pascoe, J. and Morse, D.R., “*Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant*”, in V. Gaffney, M. van Leusen and S. Exxon (eds.) Computer Applications in Archaeology, 1998.
- [**Salber et al, 99**] Salber, D., Dey, A. K. and Abowd, G. D. “*The Context Toolkit: Aiding the Development of Context-Enabled Applications*”, In: the Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), pp. 434-441, Pittsburgh, PA, May 1999.
- [**Salutation, 98**] Salutation Consortium. “*White Paper: Salutation Architecture Overview*”, <http://www.salutation.org/whitepaper/originalwp.pdf>, 1998.
- [**Satyanarayanan et al, 90**] Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E. and Steere, D. “*Coda: A highly available File System for a Distributed Workstation Environment*”, IEEE Transactions on Computers, 39(4), pp. 447-459, 1990.
- [**Satyanarayanan, 96**] Satyanarayanan, M. “*Mobile Information Access*”, IEEE Personal Communications, 3(1), pp. 26-33, 1996.
- [**Schilit et al, 94**] Schilit, B.N., Adams, N.I. and Want, R. “*Context-Aware Computing Applications*”, Proceedings of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, pp. 85-90. 1994
- [**Schill et al, 95**] Schill, A., Bellmann, B., Bohmak, W. and Kummel, S. “*System Support for Mobile Distributed Applications*”, Proc. 2nd International Workshop on Services in Distributed and Networked Environments (SDNE), Whistler, British Columbia, IEEE Computer Society Press, pp 124-131, June 1995.

- [Schmidt & Cleland, 99] Schmidt, D. and Cleland, C. "Applying Patterns to Develop Extensible ORB Middleware. IEEE Communications Magazine Special Issue on Design Patterns", 37(4), pp. 54-63, 1999.
- [Seitz et al, 98] Seitz, J., Davies, N., Ebner, M. and Friday, A. "A CORBA-based Proxy Architecture for Mobile Multimedia Applications", Proc. 2nd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS '98), Versailles, France, November 1998.
- [Siegel, 99] Siegel, J. "CORBA 3 Fundamentals and Programming", John Wiley and Sons (2nd Ed), 1999.
- [SignalSoft, 00] Wireless Location Services. <http://www.signalsoftcorp.com>. 2001
- [Singhai et al, 97] Singhai, A., Sane, A., Campbell, R., "Reflective ORBs: Supporting Robust, Time-critical Distribution", Proc. ECOOP'97 Workshop on Reflective Real-Time Object-Oriented Programming and Systems, Jyväskylä, Finland, 1997.
- [Sun, 97] Sun Microsystems Corporation, "Java RMI Specification", <ftp://ftp.javasoft.com/docs/jdk1.1/rmi-spec.pdf>, 1997.
- [Sun, 99] Sun Microsystems. "Java Core Reflection", <http://java.sun.com/products/jdk/1.1/docs/guide/reflection>
- [Szyperski, 98] Szyperski, C. "Component Software, Beyond Object-Oriented Programming", ACM Press/Addison-Wesley, 1998.
- [Truyen et al, 00] Truyen, E., Jrgensen, B. N. and Joosen, W. "Customization of Component-Based Object Request Brokers through Dynamic Configuration", in Proceedings of TOOLS Europe'2000, pp. 181-194, June 2000.
- [Veizades et al, 97] Veizades, J., Guttman, E., Perkins, C., and Kaplan, S. "Service Location Protocol (SLP)", Internet RFC 2165, 1997.
- [Venners, 97] Venners, B. "Under the hood: The architecture of aglets", Java World, 2, Issue.4, April 1997.
- [Wade, 99] Wade, S. "An Investigation into the use of the Tuple Space Paradigm in Mobile Computing Environments," Ph.D. Thesis, Computing Department, Lancaster University, 1999.
- [Waldo, 98] Waldo, J. "Javaspace specification 1.0", Sun Microsystems Technical report, March 1998.
- [Want et al, 95] Want, R., Schilit, B. N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J. and Weiser, M. "An overview of the ParcTab ubiquitous computing experiment". IEEE Personal Communications Magazine, 2(6), pp. 28-43, 1995.
- [Wegdam, 00] Wegdam, A. "Experiences with CORBA interceptors", position paper for the Workshop on Reflective Middleware, co-located with Middleware 2000, New York, USA, April 2000.
- [Weiser, 91] Weiser, M. "The Computer for the 21st Century" Scientific American", pp. 94-104, September 1991.
- [Wong et al, 97] Wong, D., Paciorek, N., Walsh, T., DiCelie, J., Young, M. and Peet, B. "Concordia: An Infrastructure for Collaborating Mobile Agents", Mobile Agents: First International Workshop, Lectures Notes in Computer Science, Vol. 1219, pp 86-97, 1997.
- [Wyckoff et al, 98] Wyckoff, P., McLaughry, S., Lehman, T. and Ford, D. "Tspaces". IBM Systems Journal, 37(3), pp 454--474, 1998.