Deploying A Middleware Architecture for Next Generation Mobile Systems

N Amanquah and J Dunlop
University of Strathclyde -Department of Electronic and Electrical Engineering, Glasgow G1 1XW, Scotland
Phone: +44 141 5482081, Fax:+44 141 5524968,
e-mail: {nathan.amanquah, j.dunlop}@strath.ac.uk

Abstract: **Although 2G systems quite adequately cater for voice communications, the demand today is for high-speed access to data centric applications and multimedia. Future networks have been designed to provide higher rates for data transmission, but this will be complemented by higher speed access to services via hotspots using secondary wireless interfaces such as Bluetooth or WLAN. With a wide range of applications that may be developed, a growing number of short range wireless interfaces that may be deployed, and with mobile terminals of different capabilities, a means to integrate all these variables in order to facilitate provision of services is desirable. This paper describes an architecture involving the use of middleware that makes software development independent of the specific wireless platform. The advantages of this approach are also presented. Sample applications for use in next generation systems were implemented on a testbed and features of these have been described. The method described is a standardised approach for service execution in mobile systems, independent of radio interface involved.**

1. Introduction

Next generation networks are expected to feature data and multimedia application sessions that require transmission rates much higher than currently achievable using the GSM interface. To complement this drive towards the delivery of high-speed services, hotspots have commenced to be deployed, which feature high speed short-range [1] wireless connections. These are aimed at high data rate applications, primarily access to the Internet, email, and sometimes to remote corporate data-based applications, and ultimately to multimedia content. Several wireless technologies exist for providing such connectivity. One notable example is the Bluetooth [2] wireless interface, which has a small form factor, and lower power consumption, and hence is targeted towards small hand held mobile devices. Although originally billed as a cable replacement technology, it is also capable of supporting such data sessions, as well as audio. Other wireless interfaces include the IEEE 802.11 series of standards, HiperLAN, and InfraRed.

Short range wireless has a significant role to play in the delivery of high speed services in next generation mobile communications. It is evident that to develop next generation services for more than one wireless interface will require knowledge of the workings of the particular target platform, and will make an application less portable. Nevertheless, that is not a sufficient requirement. It is necessary to be able to locate or discover what services are on offer, and to be able to access them, without requiring significant effort on the part of a user. With respect to service providers, an open framework where services can be developed and deployed without requiring knowledge of a specific wireless platform would be particularly useful. Furthermore, standardised procedures for interaction between distributed objects and an architecture that can integrate disparate objects in order to deliver a more useful service would be desirable. This is a role filled by middleware in conjunction with service discovery protocols.

In this work, Bluetooth has been selected as a sample wireless interface in a testbed. A middleware based architecture is described and demonstrated on the testbed, but may well be implemented on other wireless platforms. This serves to illustrate some aspects of next generation networks and services. It will be a win-win situation both for application developers (including third party developers), hardware vendors and users alike. Although Bluetooth has a short transmission range, for most of the services described or expected, this distance is adequate. [6] describes a successful deployment of specific services over Bluetooth, but only provided access to local services, and was for the Bluetooth interface only. This paper in contrast examines an architecture for deploying services over *any* short range wireless platform, Bluetooth has only been used as a sample for demonstration. First, an overview of next generation services is given, followed by a description of the testbed, and the proposed architecture. A sample deployment, together with the software is examined, and lessons therefrom are outlined in the conclusion. It is shown that middleware can be employed to abstract the wireless interface, while integrating distributed service components.

2. Overview of Services

Several types of services can be envisioned. One useful primary service is having access to the Internet, and to email from any location at a reasonable data transmission rate. Clients will prefer to access such high speed services via access points at wireless enabled information kiosks, public phone booths acting as access points, and from similar hotspots in airport terminals, train stations and banking halls –indeed wherever people aggregate and tend to have to wait for some other service. Such situations are ideal because clients remain almost stationary for long enough to establish a session to obtain the information required- for the entire life cycle of a session; a complete transaction. Hefty data or multimedia files may be downloaded faster and more cheaply during such sessions than over the GSM interface. For others, the desired service will be a location-based service. It may often take the form “where am I” and “where is the

nearest XYZ". Yet another category of services requires interaction between a client application and a remote server/service. This runs the gamut of services to satisfy a variety of client needs comprising services such as booking or checking flight times, activating an alarm system at a user's home, monitoring the location of a child, or updating a corporate database. All such scenarios can make good use of an access point conveniently located, with its own power supply and connection to the Internet or other communications network. Such services may be classified broadly as infrastructure network dependent services.

At the other end of the spectrum, infrastructureless networks can be established and some applications can still make use of these. The wireless network interfaces are capable of forming a network without the presence of a wired access point. Chat applications and file sharing schemes can make use of such peer to ad hoc peer networks. Users sitting in a bus on a journey for example can exchange information or chat by seeking out and establishing sessions with willing participants.

3. Middleware Testbed Description

Middleware Technology [4][8][9] is the architecture and specification for creating, distributing, and managing distributed program objects in a network. Distributed object computing extends an object-oriented programming system by allowing objects to be distributed across a heterogeneous network, so that each of these distributed object components inter-operate as a unified whole. These objects may be distributed on different computers throughout a network, living within their own address space outside of an application, and yet appear as though they were local to an application. The distributed objects may be components of the service, or objects to facilitate the management of qualitative and quantitative aspects of QoS. In order to make use of services available at an access point, the service must first be located after a wireless connection has been established. Such actions require the use of a service discovery protocol [7][10]. Furthermore, provision should exist for a user to download a client application if one is not available for the desired service.

For this testbed, Java RMI (remote method invocation) is the preferred middleware platform. Jini is employed as the service discovery protocol. Jini is itself based on Java RMI, a veritable middleware platform. Another reason for using Jini, in this proof of concept deployment on the testbed, is its tight integration with the Java programming language. It is quite straightforward to write Jini enabled Java applications. Furthermore, the compile once run everywhere paradigm of Java makes it convenient for client terminals to download a single copy of a client application, which will run on their hardware. Besides, with Java enabled mobile devices and software already on the market, it is positioned as a potential favourite platform for application development. Jini and RMI require the definition of an interface (a set of functions) between the client and the service. By separating the service interface from the implementation, different implementations can be provided for the same interface to suit the client' s capabilities. This is desirable in order to provide context aware QoS. The same server side code will run but be presented differently according to the capabilities of the device. Jini' s real strength is the ability to distribute platform independent executable code to the client that requires a service. It must be emphasised however that this work is not about mapping Java services to Bluetooth [11], but rather abstracting the application from whatever wireless interface is used.

4. Middleware Architecture

The middleware testbed incorporates a number of individual components that work together to provide a Jini service over a wireless interface (in this case Bluetooth). The interactions are depicted in Figure 1 The components are described in the subsequent sections. Middleware provides the isolating layer of software that shields an application from the complexities of a heterogeneous environment at the network layer and below.

The middleware testbed makes use of a distributed framework that provides extensions to the Java RMI middleware, and makes use of the Jini architecture for service discovery. It makes extensions to it to facilitate delivery of services in a uniform way, and to aid interoperability between services, and leverage devices in ways that they were not originally envisaged. This functionality is provided as a Java Class. Attributes required by all services are also specified in one class. The advantage of using a standard class is that it can be extended to provide further attributes desired by a service provider, but the basic class will remain the same. Any descendant class will still have the class members needed for the proper operation of middleware. The Middleware Toolkit has being implemented predominantly as a collection of Java classes and a set of ancillary functions to enable connectivity, and other desirable functionality.

The architecture proposed will allow third party developers to create applications independently, without significant regard for a specific target wireless interface by making use of the middleware toolkit. Support of applications created by third party developers is one factor that made Japan's iMODE a success and is a desirable feature to be supported. Middleware comes between the client application and the underlying hardware or wireless interface. A developer will not need to know the intricacies of the operation of any specific wireless platform, or its protocol stack. That nevertheless does not prevent a client application from making direct calls to the wireless interface if such is its design, although such an application will be rendered less portable between different wireless interfaces. Both types of applications may co-exist on a wireless terminal.

With this architecture, client applications will interact uniformly with middleware on the client-to-middleware interface uniformly, irrespective of the underlying interface. On the middleware-to-wireless API interface, the implementation of middleware calls

would be platform specific. For example, calls to establish a session in Bluetooth will differ from that for wireless LAN or other wireless interface, but will have the same API definition and achieve the same effect. In fact, for this testbed, work began before the completion of JSR-82 specification [5] for Bluetooth-Java interaction. Accordingly, a proprietary Bluetooth-Java interface has been defined. To make use of the JSR-82 definitions, or to replace the Bluetooth interface with HomeRf or Wireless LAN, would simply involve changing the implementation of the middleware-to-wireless interface, without affecting any of the applications previously written. Applications will operate correctly provided they have not been written to bypass the middleware. A bypass is not a desirable attribute, although it is permitted for flexibility. This does not make for portability. The role of middleware is to effectively mask the underlying hardware from the application developer.

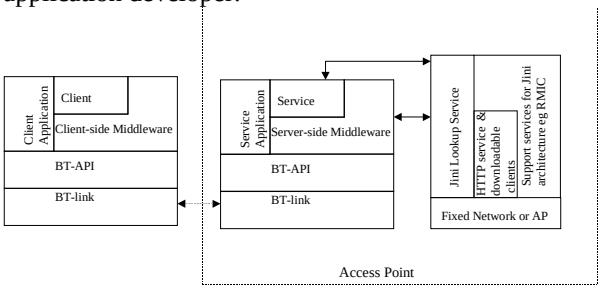

Figure 1 Middleware architecture to enable independence between wireless platform and applications

The middleware toolkit includes such functionality common to all wireless platforms and to next generation applications such as methods to establish a connection, or to tear it down, functions to determine what capabilities a mobile terminal has, in order to present a service in the proper context, and procedures to return location information if required. There are also methods to return discovered services and their attributes or to search attributes of services, as well as functions to located nearby wireless terminals. The toolkit also facilitates retrieval of the client executable from a URL. This is just a sample of available functionality.

5. Middleware Testbed: Software Components

The present work makes use of a proprietary Java-Bluetooth API and a middleware toolkit, above which client applications or services sit. It demonstrates the interaction between client and service applications and the middleware. Interaction between middleware and the wireless interface is also demonstrated. Furthermore, most Jini client or Jini service applications would share a number of common functions. This boiler plate functionality has been put together into "wrapper" applications, for clients and for services, making it faster to create other applications for the testbed. Two email clients and a service have been created to illustrate these as in Figures 2 to 4.

The email clients and service demonstrates certain features of next generation services available via access points:

The client is thin and is agnostic to the underlying connection between it and the service. It will run well on both a wired and wireless interface. Because the target environment would be wireless, data passed between the client and service is kept to the minimum possible, to give the user a responsive experience.

Figure 2 Sample Email Service

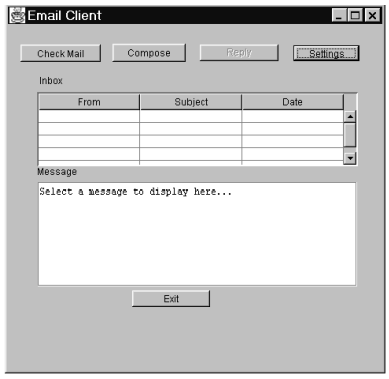

Figure 3 Graphical User Interface for Email Client

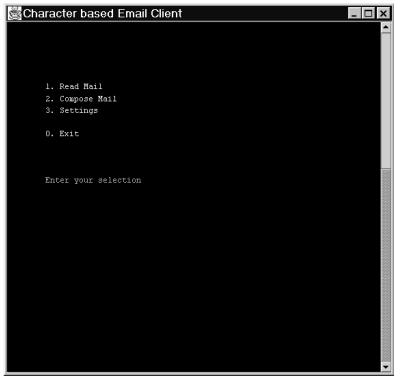

Figure 4 Text-based User Interface for Email Client

The Jini service however is not constrained by size. It has all the functionality required to provide the service. It can be a much bigger application than the client, including all the supporting libraries. For example, the email service on this test bed is in effect an email client application. This requires knowledge of how to interact with POP and SMTP servers, and how to handle mail received generally. This functionality is incorporated in the program by making use of JavaMail API, and JavaBeans Activation Framework- both extensions to the Java 2 Platform, Standard Edition

(J2SE) [3]. The JavaMail API provides a set of abstract classes that model a mail system. These extra libraries are altogether only about 600KB in this case, but for other types of services or applications, could be much bigger. It could be much bigger than a typical mobile device would handle. However, this presents no problem because all these libraries would be hosted on an access point. All a client has to be able to do is to know how to exchange messages with the Jini email service using Java RMI. The client need not know how to contact a normal mail server, nor associated protocols. This will be carried out on its behalf by the Jini mail service. The client must simply display the information returned, or pass new messages to be sent on its behalf.

A simple interface between the Jini mail service and client is defined, incorporating a few functions to provide interaction between the two. It is possible to create applications for clients targeted at devices with different capabilities, while preserving the interface between the client and the service. The client can thus be a text based one or an enhanced graphical interface. The client implementation would depend to a large degree on the capabilities of the target mobile device. Indeed, multiple implementations may be provided by the service provider, and made available for downloading to client terminals by an HTTP/FTP server. A client terminal that locates the Jini lookup service can determine what services are available. The user can then decide to download an appropriate client application, depending on the capabilities of the terminal. All the clients would use the same object interface. Alternatively, using the Jini paradigm, the user interfaces can be incorporated in the service, and the appropriate one is downloaded to a client at run time, depending on its capability. This testbed illustrates both a Graphical user interface and character based clients that access the same services, but targeted towards devices of different capabilities.

All access points are started up first, followed by all software services. The Jini Lookup service and supporting services are started. Next a utility application based on the middleware toolkit is started, followed by any application services that users may want to use. Jini services look for and register with the lookup service, so that any clients that find the lookup service can interrogate it for a proxy of a desired service. With a proxy in hand, a client can then interact with the service directly. One role of middleware in this instance is also to retrieve information about services registered with the Jini lookup service and register it with the Bluetooth's native service discovery protocol (SDP). It serves as a bridge. In principle, this could have been another Service discovery protocol service, in which case middleware will fill the role of a translation and mapping service between Jini and the other service discovery protocol. In the latter case, it would be possible for a client using a different service discovery to identify and make use of a Jini service.

A set of parameters have been defined, which provide basic information on all services, and is useful for providing context, location and quality of service information. It is expected that all applications implement these properties. These parameters form one Java class, the SDPEntries, which is registered with the Jini lookup service as an attribute of the service. On creation of the service, the service provider supplies values for these properties. At runtime, the values can be specified by a user interface. These values are retrieved during service discovery, and provide useful information on the hierarchical categorisation of registered services. It makes for easy browsing of services. Information is provided on where to obtain a downloadable client, as well as where this access point is located, thus facilitating the presentation of location based information. These basic attributes are given in Table 1 along with a brief description of what each parameter represents. Figure 5 also illustrates a sample implementation of this class for the Email Service.

These attributes have a two fold purpose. Service providers implement this class with their applications, which get registered with the look up service so that all relevant data is supplied to the lookup service. Then the server-side middleware can extract this information and package it for the client side middleware, to facilitate speedy service discovery. If a second service discovery protocol is in use, the server side middleware is responsible for registering these values with it.

Fields of SDP Entries	**Description**
Class-Hierarchy	Provider defined hierarchy in relation to other services.
ServiceName	Name of the service
Service-Description	A brief description of the service
ProviderName	Name of service provider
Location	Geographical Location of this service
Category	Class of QoS that characterises this service.
ServiceID	Unique ID for this service.
Lookup-ServerIP	IP address of Lookup Service – May be local, optional
AccessPointIP	IP address of access point. May be local, optional
ClientExecut-ableURL	Downloadable executable for this service
IconURL	An Icon for use by Bluetooth for this service

Table 1 SDPEntries Class attributes

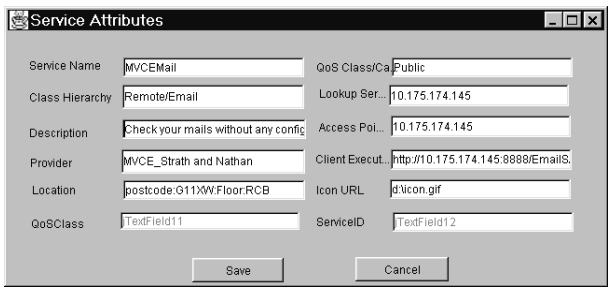

Figure 5 Sample Implementation of SDPEntries object by Email Server

6. Client terminal

The testbed presently demonstrates that the client application can be written entirely independently of the underlying wireless interface. Furthermore, being a Java-based application, this will run on any platform with a Java virtual machine. Certainly, consideration would have to be given to using what Java functionality is available in limited versions of Jini such as the Java 2 Micro Edition (J2ME) for small mobile terminals. It is assumed that when developing applications for such an environment, the appropriate substitute objects for what is not available in the limited versions would be employed.

On the client terminal is also located a utility application based on the middleware toolkit, that interacts with applications on one hand, and the underlying wireless interface on the other (in this case Bluetooth). This client-side middleware is responsible for locating other nearby wireless devices or access points, determining what services are available to the remote device selected, and establishing a suitable session with the AP. The user interface of the client side middleware based utility makes it possible to browse the services on the remote device. Based on the services available at this access point the relevant client application can be started. One important piece of information returned during service discovery is the IP address of the Jini Lookup server from which a 'registrar' or proxy for the desired service can be obtained. The IP address is useful for setting up a Jini session because using the known IP address, the client application can make a unicast request to the Jini lookup service for the desired service's proxy, in order to be able to contact the service desired. This client side middleware toolkit based utility works almost as an extension to the operating system. It goes between the client application and the underlying wireless interface. It is also used to download the client application from the remote device or AP if it is unavailable locally. Nevertheless, a client application can bypass the client-side middleware altogether, and make use of the Java-Bluetooth API directly.

7. The Service

The service is written independently of the Bluetooth interface. The service only is required to implement the SDPEntries object with attributes, which would be registered with the Jini lookup service. This object describes various features of the service. The server-side middleware interrogates the Jini lookup server and identifies all objects with these attributes, and can make this information available to a second service discovery protocol. In this case, the server-side middleware makes use of a Java Bluetooth API object to write to the Bluetooth Service discovery protocol database in a format that can be read by other Bluetooth devices (even if such Bluetooth devices had no client-side Jini middleware available on them). The service can be of any size and complexity required to accomplish the task and can be updated at any time without modification to the clients, as long as the interface between it and the client is preserved.

8. Supporting Services

A number of services, including a Jini Lookup Server are deployed to create the Jini framework, as well as to provide a means to download a client when it is not available locally on the client device. These services may run on the same host as the access point or on another host. There may also be interaction between the Jini service and other remotely located services as need be, in order to fulfil its service requirements.

9. What is demonstrated:

- Clients deployed on a mobile terminal can be made very thin, and can be tailored to meet the capabilities of the terminal. Any clients developed to the same service interface can interact with the bulky service to obtain the same desired results.
- Location independent client operation: The email client on this testbed for example needs no configuration changes when it establishes a connection with an Access Point. The only configuration made once for all time is specifying what the user's usual email settings are. On change of network, other factors like IP address that usually affect network access, no longer require user intervention. Also any access point may be used to access the service, without worrying about the new network's configuration (as long as there is a route from the new AP offering the mail service to the target mail server desired). An identical scheme can be employed for a web browsing service.
- There is a separation between application development (client and service) and protocols for the wireless interface.
- The use of multiple client user interfaces for the same service, reflecting the context/capabilities of the client terminal is supported.
- Service discovery is carried out in the Bluetooth sphere (second service discovery protocol) and in the Jini environment, as well as by purely RMI based methods.
- Location dependent service have been demonstrated on the testbed by extracting the location in a building or post code of the access point, and passing this on to a location service, which returns relevant maps or information for this location.
- Although this is geared towards infrastructure networks, a chat application has been deployed on the tested. The application can be downloaded to a client which lacks it, and clients can engage in peer to peer communications, by making use of middleware to determine what other clients are within range, and if they would be willing participants in this type of communications.

The middleware toolkit has been defined as an extensible Java class. No user interface has been predefined for use with the middleware toolkit. The service provider will only need to make calls to its member functions in order to obtain the effect desired.

The interface will remain the same for different wireless interfaces, but the actual implementation of each function can be overridden or extended to obtain the desired effect. Thus it is possible to extend the middleware toolkit to incorporate platform specific quality of service management routines.

One role of the middleware toolkit includes retrieving information about services registered with the Jini lookup service and registering it with a second service discovery protocol if present. (in this case Bluetooth's native SDP).

Middleware functionality can be extended to provide other qualitative and quantitative QoS support, without affecting the basic functionality desired. The middleware toolkit enables an application to be oblivious of QoS management. Independence from the underlying wireless platform is also provided by middleware. Applications will only fail on a new wireless platform when they have been written to bypass the middleware architecture. It is recommended that for IP based services, RMI be used for interactions between the client application and the service. This paper has presented a standardised methodology for service execution in mobile systems which is independent of the radio interface involved.

10. Acknowledgement

The work reported in this paper has formed part of the WA1 area of the Core 2 Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.co.uk, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.

11. References:

[1] D. Leeper, "A long term-view of Short Range Wireless", IEEE Computer, p39-44, June 2001

[2] Bluetooth Special Interest Group, "The Bluetooth Specifications", www.bluetooth.com

[3] Sun MicroSystems, "Java 2 Platform, Standard Edition (J2SE)", http://java.sun.com

[4] Campbell AT, Coulson G, Kounavis ME, "Managing Complexity: Middleware Explained", IEEE IT Pro Magazine, September/October 1999 p22-28

[5] Java Community Process, "JSR 82 Java APIs for Bluetooth", Java Specification Requests, http://www.jcp.org/en/jsr/detail?id=82

[6] R Kraemer, "Bluetooth Based Wireless Internet Applications for Indoor Hot Spots: Experience of a Successful Experiment during CeBIT 2001", 26th Annual IEEE Conference on Local Computer Networks, Tampa, USA, November 14 - 16, 2001

[7] R. E. McGrath, "Discovery and its discontents: Discovery protocols for ubiquitous computing", Ubiquitous Computing, Department of Computer Science University of Illinois, Urbana-Champaign, April 2000.

[8] M. B. Juric, I. Rozman, and M. Hericko, "Performance Comparison Of Corba And Rmi", Information and Software Technology Journal, Elsevier Science, vol. 42, pp. 915{933, October 2000.

[9] G. S. Raj, "A detailed comparison of corba, dcom and java/rmi", http://my.execpc.com/~gopalan/misc/ compare.html, September 1998.

[10] G G Richard, "Service Advertisement and Discovery: Enabling Universal Device Cooperation", IEEE Internet Computing, Sept-Oct 2000, p18-26

[11] J Dunlop, N Amanquah, "Mapping of Services on Bluetooth Radio Networks", In proceedings of European Wireless 2002 Conference (Florence),Vol 1,p69-74