Middleware Awareness in Mobile Computing

Paul Grace
Computing Department,
Lancaster University,
Lancaster, LA1 4YR, UK
p.grace@lancaster.ac.uk

Gordon S. Blair
Computing Department,
Lancaster University,
Lancaster, LA1 4YR, UK
gordon@comp.lancs.ac.uk

Sam Samuel
Bell Laboratories, Lucent
Technologies, Stonehill
Green, Swindon, SN5 7DJ
lsamuel@lucent.com

Abstract

The mobile computing domain presents major new challenges for middleware to overcome. In particular the mobile environment is characterised by frequent changes and often poor network QoS. Therefore, a number of middleware platforms and paradigms have been put forward to solve these issues. This in turn though has generated a problem i.e. middleware heterogeneity exists within this domain. As a consequence, mobile client applications developed upon one type of middleware are unable to interoperate and utilise services implemented on an alternative. In this paper, we examine the issue of middleware heterogeneity and propose a configurable and dynamically reconfigurable middleware platform, named ReMMoC (Reflective Middleware for Mobile Computing), which allows mobile client applications to be developed independently of the underlying middleware technology.

1. Introduction

Research into middleware for mobile computing (where mobile hosts physically change locations) has concentrated on overcoming problems related to the characteristics of wireless networks. Extensions to well-established platforms for fixed networks have been proposed [1, 2, 3]. Asynchronous middleware paradigms have been argued to be the preferred style of communication in an environment where network connection is unpredictable [4, 5]. Finally, middleware that utilises context information to react to the frequent changes of the mobile environment have been developed to better support mobile applications [6, 7]. Therefore, mobile application services can be implemented across many styles of middleware, including paradigms such as: remote method invocation, publish-subscribe, message-oriented and tuple spaces. However, this introduces *middleware heterogeneity* to the mobile computing domain [8]; a client application implemented upon publish-subscribe middleware cannot interoperate with an RMI-based service of the same application type. Likewise, a SOAP client cannot interact with a CORBA service. Therefore, clients developed on a specific type cannot interoperate with services developed upon another.

Similarly, application services at different locations are advertised using one of the contrasting service discovery protocols available (e.g. Jini, SLP, UPnP and Salutation). Utilising only one of these technologies to discover services will mean that services advertised by the other types will be missed. This problem is likely to become significantly worse in the future with the advent of ubiquitous computing, a field with its own specialized discovery and interaction protocols, e.g. JESA [9], Centaurus [10] and the Bluetooth Service Discovery Protocol.

Given the heterogeneity that exists across middleware and service discovery protocols we argue that dynamic awareness of middleware technologies in the current environment is required, in order to allow mobile client applications to continue their operation (the types of middleware interactions required for this is unknown until run-time). We therefore present ReMMoC, a configurable and dynamically reconfigurable middleware for mobile computing. The fundamental mechanism of ReMMoC is to discover the services available at a location, irrespective of the advertising discovery protocol and then dynamically adapt itself to allow interoperation across the obtained underlying middleware type or types. Therefore, a client application can be developed independently from specific middleware types, allowing interoperation with a range of discovered heterogeneous services.

The paper is structured as follows. Section two presents a motivating mobile scenario illustrating the requirement for mobile middleware to be aware of and overcome heterogeneity of middleware. Section 3 examines current mobile middleware and their support

for overcoming heterogeneity. Section 4 presents a description of the ReMMoC platform. Finally, section 5 draws conclusions about the effectiveness of this work.

2. Illustrating middleware heterogeneity

In this section we present a typical mobile computing scenario to illustrate the middleware heterogeneity that exists in the mobile domain. It consists of three applications that may be encountered and utilised at home and in the office. Application 1 prints documents to a nearby printer, which can be carried out at both home and the office. Application 2 is a music playing system; the user can select available songs to play through a given music player in the house. Finally, application 3 is an office bulletin service; the latest news is sent to every subscriber.

Figure 1 illustrates a mobile user switching between home and office, using a PDA connected via 802.11b wireless networks to use the available services. At home, the printer can be used through a SOAP service and the music player using a CORBA service. At work, the print service is implemented as a CORBA service and the office bulletin utilises a publish-subscribe service. This illustrates heterogeneous middleware types existing within the mobile domain. If the print application was implemented as a SOAP client it would not function in the office.

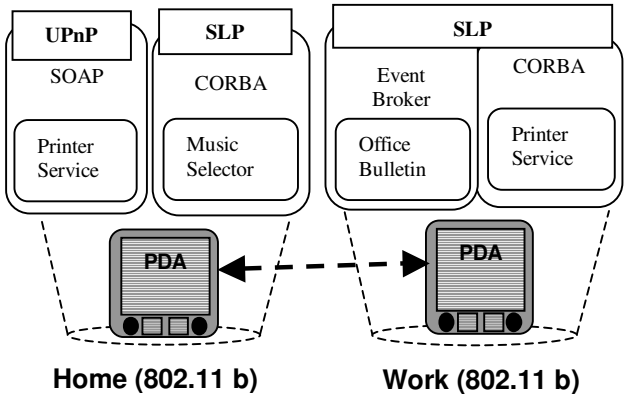

Figure 1. Mobile computing scenario.

However, this is not the only level of heterogeneity within this scenario; the services themselves are advertised by different service discovery protocols. Services in the home are advertised using UPnP & SLP and services in the workplace use just SLP. If only one protocol is used then some of the available resources cannot be found, for example if only SLP is used on the device then the printer service at home cannot be used.

Given scenarios of this type we argue that to support mobile client application development, middleware for mobile computing must be aware of the service discovery protocols in use in the environment and also the middleware used to develop these services upon i.e. the middleware on the mobile device must configure its operation to match the environment. In turn, this will allow mobile applications to be developed for use across different middleware types in locations and domains unknown to the application developer.

3. Middleware for mobile computing

Research into middleware for mobile computing has generated a set of technologies, which provide good mechanisms for overcoming some of the major problems inherent in the domain of mobile computing (e.g. weak connection and poor network QoS). In this section we examine some of these key platforms, however further work is still needed to deal with the heterogeneity problem that has emerged within this domain.

3.1 Asynchronous mobile middleware

Frequent disconnections from the network, high error rates and lost packets, as the mobile host physically moves, have proven a driving factor in the initial development of middleware platforms for mobile computing. For example, the Rover platform [11] was one of the very first to address this issue; the toolkit provides queued remote procedure calls (together with relocatable dynamic objects) that allow an application to continue making invocations asynchronously while disconnected from the network. Other asynchronous styles include publish-subscribe systems and tuple spaces. Within a publish-subscribe system, consumers register for the events they are interested in and are informed when they occur. Examples of these are Elvin [12] and Gryphon [13]. However, these platforms were designed for fixed networks and do not take into account the dynamic connection of mobile hosts. This has enforced the emergence of some preliminary solutions. For example, Elvin has been extended to incorporate proxy servers to support the persistency of events, so that clients who disconnect repeatedly do not lose events. JEDI [14] includes a dynamic tree of dispatchers (the client can reconnect to any) for ensuring publish-subscribe information is retained as members connect and reconnect. It is interesting to note though that both of these rely on centralised entities holding event information, which cannot be guaranteed within ad-hoc wireless networks. In response to this, STEAM [5] is a scalable, publish-subscribe system designed to operate in ad-hoc networks; the platform is based upon the concepts of group communication with publishers and subscribers belonging to the same group. The communication is

scaled by the proximity of publisher to subscriber; any subscribers out of range do not receive the events.

The tuple space is an alternative asynchronous communication model that is effectively a shared distributed memory spread across all participating hosts that processes can concurrently access; hence communication is decoupled in time and space. The L^{2}imbo platform [15] is based upon the classic tuple space architecture but includes a number of extensions for operation within a mobile environment. Multiple tuple spaces can be created and used, removing the need for all operations to go through a central global tuple; this is an important factor in an environment where communication links are unreliable. Lime [4] is similar to L^{2}imbo in this aspect and is an important middleware in this domain. However, L^{2}imbo allows QoS attributes to be added to a tuple, including delivery deadline allowing the system to re-order to make the best use of network connectivity.

3.2 Adaptive middleware

Established middleware technologies and those described in the previous section offer a fixed black-box implementation whose underlying structure and behaviour is hidden from the programmer and cannot be altered at run-time to cope with changes that occur in the mobile environment. Therefore, future middleware platforms, for domains such as multimedia and mobile computing, should be *configurable* to match the requirements of a given application domain and dynamically *reconfigurable* to enable the platform to respond to changes in its environment [6].

Recently, a group of *reflective middleware* technologies have emerged to meet these requirements: OpenORB [6], DynamicTAO [16] and OpenCORBA [17]. However, these existing systems are built for application domains, such as multimedia and real-time; they do not address the issue of middleware heterogeneity in mobile computing. Consequently, the Universal Interoperable Core (UIC) [8] has been developed; this reflective middleware is loosely based on the reconfiguration techniques of DynamicTAO. The platform can change between different middleware *personalities* e.g. a SOAP client, a CORBA server and a SOAP server. The implementation of UIC concentrates on synchronous middleware styles and does not implement asynchronous paradigms, nor does it address the issue of heterogeneous discovery protocols.

Furthermore, middleware and applications need to be aware of context information to support adaptation. CHARISMA [7] examines the use of reflection in managing a repository of application meta-data that stores each application's requirements for adaptation. Reflection is used to inspect and adapt this, so that behaviour of the middleware can be altered dynamically.

3.3 Service discovery protocols

Currently, there is a set of established service discovery protocols and emerging technologies to better support the mobile domain. Jini is a Java-based discovery protocol based around central lookup tables, to which service lookup requests are sent. Its use of centralized stores of services makes it less suited for mobile computing across ad-hoc networks. However, it offers the ability to download a proxy object to interact with the service, providing a method for overcoming heterogeneity. Alternatively, Universal Plug and Play (UPnP) allows SOAP services to be presented using the Simple Service Discovery Protocol and supports the use of events to notify service subscribers of changes to service state. Service Location Protocol (SLP) is an IETF standard for service discovery, which can be used in different network types e.g. it uses a central discovery point in fixed networks and multicast based lookup in ad-hoc networks. Other established service discovery techniques include Salutation and SDP within the Bluetooth protocol suite.

It has been identified that these service discovery protocols do not suit the mobile domain due to network structures and protocol efficiency; therefore some improved solutions are emerging. Examples of these are JESA [9] and Centaurus [10].

3.4 Others

Other projects have extended traditional platforms to make them effective over wireless networks. For example, ALICE [1] presents a layered architecture for managing the movement of mobile hosts and ensures that CORBA connections remain established transparently. Alternatively, RAPP [2] allows proxies to be inserted between distributed CORBA objects to manage poor levels of network service and disconnection. Finally, [3] implements a session layer that allows CORBA invocations to be made over the Wireless Application Protocol.

3.5 Summary

In this section we have identified the advances of middleware technologies for mobile computing. Though in turn these generate the major problem of escalating heterogeneity. Only UIC and Jini attempt to address this issue. Jini uses a service side approach based upon mobile code to allow clients to interoperate with a discovered service (which may be a Java RMI service, CORBA

service or other). Developing clients in this style relies on all services offering a mobile code proxy, which cannot be guaranteed. Alternatively, UIC uses a reflective technique to change the client's middleware in order to communicate with the service; however, it does not address heterogeneous discovery. We therefore, propose a dynamic client based approach to change between different service discovery personalities and different middleware platforms to allow discovery and interoperation in unknown environments to continue.

4. ReMMoC

4.1 Overview

This section briefly describes the ReMMoC platform, a configurable and reconfigurable reflective middleware that supports mobile application development and overcomes the heterogeneous properties of the mobile environment. ReMMoC uses OpenCOM [18] as its underlying component technology and is built as a set of component frameworks (CFs). OpenCOM is a lightweight, efficient and reflective component model, built using a subset of Microsoft COM. ReMMoC is then implemented as a set of component frameworks, where a component framework is defined as *"a set of rules and contracts that govern the interaction of a set of components"* [19].

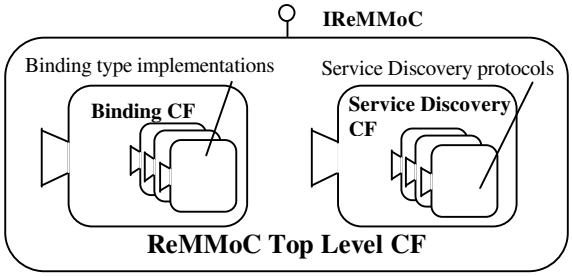

Figure 2. Overview of the ReMMoC platform

ReMMoC consists of two key component frameworks: (1) a binding framework for interoperation with mobile services implemented upon different middleware types, and (2) a service discovery framework for discovering services advertised by a range of service discovery protocols. These two component frameworks can be seen in figure 2. The binding framework is configured by plugging in different binding type implementations e.g. IIOP Client, Publisher, SOAP client etc. and the service discovery framework is similarly configured by plugging in different service discovery protocols. The top level CF of ReMMoC manages the reconfiguration of the underlying frameworks and provides an interface for discovering and interoperating with services. However, the application can also interact directly with either of the frameworks to avoid indirection and extra processing overhead, for example, if the application knows the binding type it wants to use.

4.2 The Binding Framework

The primary function of the binding framework (illustrated in figure 3) is to provide interoperation with heterogeneous mobile services that exist in the mobile environment. Therefore, over time it may be configured as an IIOP client configuration and make a number of IIOP requests, or change to a subscribe configuration and wait to receive events of interest. Different middleware paradigms, synchronous or asynchronous (e.g. tuple spaces, media streams, RPC, publish-subscribe or messaging), can be plugged into the binding framework if they have been implemented using OpenCOM components. The component framework structure manages the configuration and dynamic reconfiguration of these bindings and ensures that a correct binding type is in place before operation occurs. Each component framework in the platform implements three meta-interfaces (ICFMetaInterface, ICFMetaArchitecture and ICFMetaInterception) that can be utilised by a higher-level CF or by an application component; these are based upon the meta-spaces proposed in [6]. These interfaces allow the inspection of the current structure of the framework (represented in the CF as a graph) and the ability to dynamically alter its behaviour.

The binding framework allows changes to be made at two distinct levels. Firstly, the current binding type implementation can be replaced. For example, the IIOP configuration shown in figure 3 can be replaced by a publish-subscribe configuration or a SOAP configuration, to allow operation with services of that type. It is also feasible for multiple personalities to be created e.g. SOAP and IIOP clients together, or a publish-subscribe publisher and SOAP client together; their implementation is simply a configuration of components, but more than one interface is exposed by the framework. The implementation of each binding type is designed to minimise the number of components used, while maintaining a level of re-usable components and scope for fine-grained reconfiguration.

Changes to the framework can also be made at the component level; that is, more fine-grained changes to the configuration can be made in light of environmental context changes, such as those involving quality of service, or changes in the application's requirements. For example, an application may require IIOP server side functionality, in addition to the existing client side; therefore components implementing server side functionality are added.

In order to test and evaluate the ReMMoC platform, we have implemented IIOP client and server, SOAP client and Publish-Subscribe personalities.

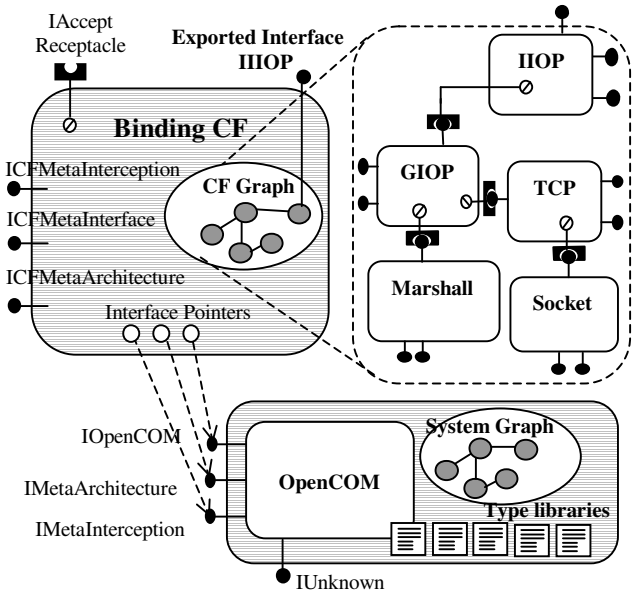

Figure 3. Binding component framework

4.3 The Service Discovery Framework

The Service Discovery framework allows services that have been advertised by different service discovery protocols to be discovered. The component configuration is configured to the discovery technology currently used in the environment. For example, if only SLP is currently in use, the framework's configuration will be an SLP Lookup personality. However, if SLP and UPnP are both being utilised at a location then the framework's configuration will include component implementations to discover both.

The basic constituents of this component framework are identical to the binding CF; the same three meta-interfaces are implemented and configurations are checked as valid service discovery personalities before operation continues. The service discovery framework differs in that it presents its own custom interface of the service it provides (*IServiceLookup*). This interface offers a set of generic discovery methods. The first makes a generic service lookup call and will return the information from any service discovery technology lookup call in a generic format. For example, if a generic lookup of a weather service is called and two discovery configurations are implemented by the framework (e.g. UPnP and SLP) then the framework will return a list of matched services from both types. It is this information that is used to configure the binding framework.

However, the discovery protocol(s) that are currently in use at a location must be determined. Therefore, the service discovery framework provides two styles of functionality to find whether particular types of service discovery protocols are in use. A single component for each technology (e.g. Jini, SLP, UPnP and Salutation) is plugged into the framework that tests if that service discovery protocol is in use, either upon a synchronous request or by continuously monitoring the environment and generating an event on detection. Continuous monitoring will quickly use up resources (e.g. battery power), therefore in some cases synchronous checking may be appropriate. Given this information the appropriate service discovery component configuration can be set up. The top-level ReMMoC component framework utilises this behaviour to automatically reconfigure the service discovery framework based upon monitoring of the environment.

Other methods for discovering discovery protocols, not currently included in the implementation, may utilise the device's context information, e.g. if the device is currently using a Bluetooth connection then an SDP personality is configured. Furthermore, the middleware may use prior knowledge to select an appropriate protocol, i.e. the platform stores context information per location that details which service discovery protocols were used at that point previously.

We have implemented the service discovery framework with two service discovery protocol implementations: SLP and UPnP both with service lookup and registration capabilities, allowing us to demonstrate how to overcome the problems of the availability of multiple service discovery protocols. However, as with the binding framework, it is feasible for new discovery protocols to be integrated into the framework.

5. Conclusions and Future Work

In this paper we have described the requirement for middleware for mobile computing to be aware of other middleware technology within the environment. Such a middleware platform must support discovery and interoperation with application services implemented on a range of middleware technologies. Given this functionality, mobile client applications can be developed without the knowledge of the middleware that will be encountered at run-time.

We have briefly presented a reflective middleware for mobile computing (ReMMoC) that supports the development of mobile client applications by overcoming the problems of middleware heterogeneity. ReMMoC can configure and reconfigure itself between a number of discovery and binding personalities. More detail on

ReMMoC can be found in [20]. This paper also contains an evaluation of the approach in terms of resource use and support for mobile applications. For example, it is shown that individual middleware configurations with a static memory footprint, typically less than 200 Kbytes can be created; we also demonstrate that the platform discovers and interoperates with services across a range of middleware types for a set of mobile applications.

The ReMMoC framework currently presents a complex dynamic programming interface to the application programmer and it is required for code to make dynamic calls across different bindings to be implemented (this is not a simple task). Therefore, we aim to construct an XML based language for describing service requests that abstracts the programmer from the different middleware implementations that may be encountered.

Furthermore, ReMMoC is made up of a number of components that may be used at one time or another. However, storing all of the components that may be used will quickly exhaust the mobile device's resources. Therefore, a component downloading architecture is needed that allows components to be downloaded and used only when required. To improve the performance times of this, a predictive cache for ensuring components are loaded on the device before they are used is an interesting option.

6. References

[1] 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.

[2] Seitz, J., Davies, N., Ebner, M. and Friday, A. "A CORBA-based Proxy Architecture for Mobile Multimedia Applications". *In Proceedings of the 2nd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS '98),* Versailles, France, November 1998.

[3] Reinstorf, T., Ruggaber, R., Seitz, J. and Zitterbart, M. "A WAP-Based Session Layer Supporting Distributed Application in Nomadic Environments". *In Proceedings of Middleware 2001*, Heidelberg, Germany, November 2001.

[4] Murphy, A., Picco, G. and Roman, G. "LIME: A Middleware for logical and Physical Mobility". *In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21),* May 2001.

[5] Meier, R. and Cahill, V. "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks". *In Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02),* Vienna, Austria, 2002.

[6] Blair, G., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., and Saikoski, K. "The design and implementation of Open ORB 2". *IEEE Distributed Systems Online*, 2(6), September 2001.

[7] 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.

[8] Roman, M., Kon, F. and Campbell, R. H. "Reflective Middleware: From Your Desk to Your Hand". *IEEE Distributed Systems Online*, 2(5), 2001.

[9] Preuss, S. *"JESA Service Discovery Protocol". In Proceedings of Networking 2002*, pp 1196-1201, Pisa, Italy, May 2002.

[10] Kagal, L., Korolev, V., Chen, H., Joshi, A., and Finin, T. "Centaurus: A framework for intelligent services in a mobile environment". *In Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC),* April 2001.

[11] Joseph, A., deLespinasse, A., Tauber, J., Gifford, D. and Kaashoek, M. "Rover: A Toolkit for Mobile Information Access". *In Proceedings of the 15th Symposium on Operating Systems Principles (SOSP '95),* Colorado, U.S., pp 156-171, December 1995.

[12] Segall, B and Arnold, D. "Elvin has left the building: a publish/subscribe notification service with quenching". *In Proceedings of AUUG97*, September 1997.

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

[14] Cugola, G., Di Nitto, E., and Fuggetta, A. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS". *IEEE Transactions on Software Engineering,* 9(27), pp827-850, September 2001.

[15] Davies, N., Friday, A., Wade, S. and Blair, G. S. "L^{2}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.

[16] 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 Middleware 2000, ACM/IFIP*, April 2000.

[17] Ledoux, T. "OpenCorba: a Reflective Open Broker". *In 2nd International Conference on Reflection and Meta-level Architectures*, St. Malo, France, July 1999.

[18] Clarke, M., Blair, G., Coulson, G. and Parlavantzas, N. *"An Efficient Component Model for the Construction of Adaptive Middleware". In Proceedings of Middleware 2001*, Heidelberg, Germany. November 2001.

[19] Szyperski, C. "*Component Software: Beyond Object-Oriented Programming*". Addison Wesley, 1998.

[20] Grace, P., Blair, G. and Samuel, S. "Interoperating with Services in a Mobile Environment*", Technical Report (MPG-03-01), Lancaster University*. 2003.