A Marriage of Web Services and Reflective Middleware to Solve the Problem of Mobile Client Interoperability

Paul Grace[1], Gordon Blair[1] and Sam Samuel[2]
[1]Computing Department, Lancaster University, Lancaster, LA1 4YR, UK.
{p.grace@lancaster.ac.uk, gordon@comp.lancs.ac.uk}
[2] Bell Laboratories, Lucent Technologies, Westlea, Swindon, SN5 7DJ, UK.
{lsamuel@lucent.com}

Abstract

Mobile client applications must discover and interoperate with application services available to them at their present location. However, these services will be developed upon a range of middleware types (e.g. RMI and publish-subscribe) and advertised using different service discovery protocols (e.g. UPnP and SLP) unknown to the application developer. Therefore, a middleware platform supporting mobile client applications should ideally adapt its behaviour to interoperate with any type of discovered service. Furthermore, these applications should be developed independently from particular middleware implementations, as the interaction type is unknown until run-time. This paper presents ReMMoC, a reflective middleware platform that dynamically adapts both its binding and discovery protocol to allow interoperation with heterogeneous services. Furthermore, we present the ReMMoC programming model, which is based upon the Web Services concept of abstract services.

1. Introduction

Mobile computing is characterised by mobile hosts physically changing location. As they move location, they are likely to encounter application services that have been developed upon a range of middleware paradigms (e.g. Remote Method Invocation (RMI), Publish-Subscribe, Tuple Spaces). Furthermore, the varieties of implementations of individual middleware paradigms e.g. SOAP and CORBA for RMI extenuate the problem of middleware heterogeneity within this domain. Hence, a client application implemented upon a particular middleware platform (e.g. SOAP) will not interoperate with the discovered services implemented upon different platforms (e.g. Internet Inter-ORB Protocol (IIOP) or Publish-Subscribe), as the user moves from location to location. Similarly, available services can be advertised using one of the contrasting service discovery protocols available, such as Jini, Service Location Protocol (SLP) and Universal Plug and Play (UPnP); if a mobile application discovers services using only one protocol then services advertised by others will be missed.

We argue that an adaptive middleware platform is required to support the interoperation of mobile clients. This platform should alter its behaviour dynamically to: i) find the required mobile services irrespective of the service discovery protocol and ii) interoperate with services implemented by different middleware types. We utilise reflection and component technology to develop a middleware with these capabilities, named ReMMoC (Reflective Middleware for Mobile Computing). Reflection is a principled method that supports introspection and adaptation to produce configurable and reconfigurable middleware. Given these properties, mobile applications can then be developed independently of the underlying middleware technology, allowing them to continue interoperation across different locations.

For an application to dynamically operate across different middleware implementations it must be programmed independently from them. An abstract definition of the application service's functionality is required. The mobile client application, which requests the service, can then be developed against this "interface" in the style of IDL programming. A request of the abstract service is mapped at run-time to the corresponding concrete request of the

middleware implementation. The emerging Web Services Architecture (described in section 2) includes a Web Services Description Language (WSDL) that provides this format of abstract and concrete service definition. ReMMoC bases its programming model on WSDL.

In this paper, we demonstrate the combination of a reflective middleware and the WSDL programming model to provide a solution to the problem of interoperation from mobile clients. The client is programmed independently from any specific middleware implementation using WSDL and the reflective middleware dynamically adapts between middleware implementations based upon the environment. Abstract requests are then mapped to this middleware. The paper is structured as follows. Section 2 describes the web services programming model and section 3 discusses the mapping of abstract definitions to concrete paradigms. Section 4 then introduces the reflective middleware platform (ReMMoC) and finally, section 5 draws concluding remarks and discusses future work.

2. Overview of Web Services

A web service is defined as: *"a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition using XML based messages conveyed by Internet protocols"* [1]. The web services architecture consists of three key roles: a service provider, a service requestor and the discovery agency, which the requestor uses to find the service description. Each service is described in WSDL [2], which is an XML format for documenting the exchange of messages (containing typed data items) between the service requestor and service provider. The exchange of messages is termed an operation, and a collection of operations is called a port type (interface). Finally, a service is a group of ports. The key property of WSDL is that it separates the abstract description of service functionality from the concrete details of the service implementation.

The aim of Web Services is to allow different service providers to implement this abstract service description upon their chosen concrete middleware binding. For example, a news service may be implemented using SOAP by one vendor while another may use publish-subscribe. Client applications, programmed against the abstract WSDL definition, can interoperate with both concrete implementations.

WSDL offers the ability to develop mobile clients, based upon agreed abstract service descriptions. Hiding the developer from the problem of middleware heterogeneity encountered across different locations. Furthermore, by utilising an existing standard the probability of interoperability is increased; rather than propose a new abstract service definition language ("new standard") that must be accepted. However, what is missing from Web Services is the capability to dynamically adapt the binding at runtime to allow continued interoperation with abstract services implemented across contrasting concrete bindings. We discuss a reflective middleware, providing this capability, in section 4. Firstly however, we illustrate in the next section the mapping of abstract WSDL descriptions to different bindings e.g. RMI and publish-subscribe. This shows that WSDL can be mapped to the diverse paradigms that are encountered within mobile environments.

3. Mapping Abstract Services to Concrete Binding Types

In this section we demonstrate how the abstract operations of WSDL can be mapped to two contrasting binding types exposed by a reflective middleware (RMI and publish-subscribe). The following four abstract operations can be described in WSDL. (1) *Request-Response (input, output)*, a service receives a request of its functionality and responds to it. (2) *Solicit-Response (output, input)*, a service provider acts as a service requestor. (3) *One-Way (input)*, a

service receives a notification message. (4) *Notification (output)*, a service outputs a notification message.

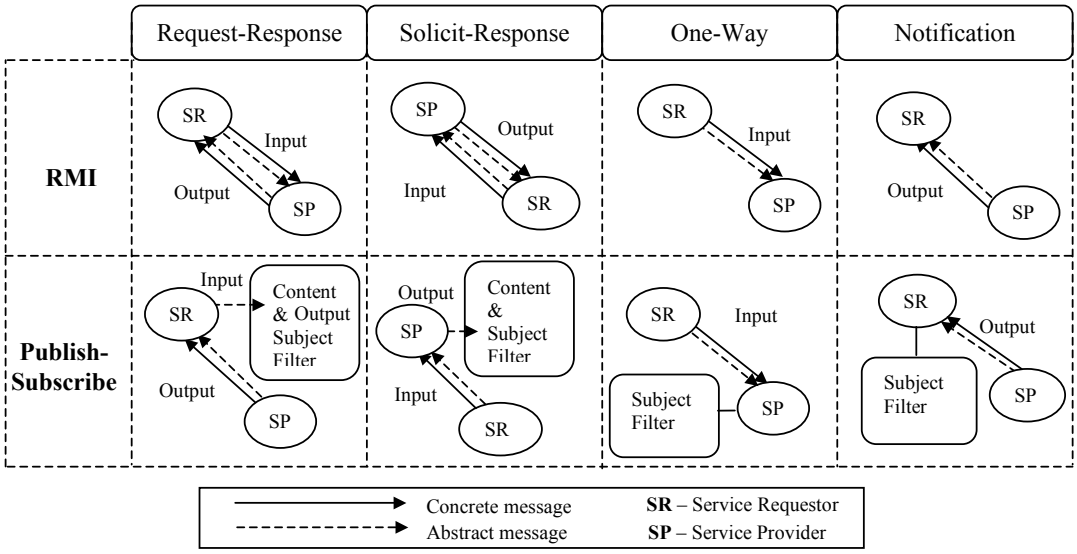

Figure 1. Mapping WSDL operations to different middleware paradigms

Figure 1 illustrates how abstract messages (input and output) that constitute each WSDL operation map to the RMI and publish-subscribe communication paradigms. The service requestor is the mobile client. We assume that each paradigm understands the set of types used by the abstract definition. In RMI, the input/output messages of Request-Response and Solicit-Response operations can be mapped directly to the corresponding synchronous RMI messages of SOAP and IIOP. The operation name maps to the method name, the input message to the input parameter list and the output message to the output parameter list. Similarly, Notification and One-Way operations can be mapped as one-way messages e.g. one-way IIOP invocations and asynchronous SOAP messages.

Publish-Subscribe however is an alternative communication paradigm whereby there is no direct message exchange between service requestor and provider. The service provider publishes events and a service requestor must filter to receive appropriate events. Therefore unlike RMI, the mapping of WSDL to publish-subscribe is not a direct correlation. The request-response operation is a request of a service based upon the input message. The input message can be used to filter published messages and receive the correct event, whose content maps to the output message. The operation name maps to the event subject, while the input message maps to the content filter attributes. Similarly, for Solicit-response the service filters to receive events from other services. For One-way operations and Notifications, services subscribe and publish events based upon subject filtering only, with the content of the concrete message mapping to the abstract message.

4. ReMMoC: A Reflective Middleware for Mobile Computing

4.1 Overview of ReMMoC

In order to support interoperation, the underlying middleware must dynamically adapt to communicate with different middleware implementations. This section briefly describes the ReMMoC platform, a configurable and reconfigurable reflective middleware that overcomes the heterogeneous properties of the mobile environment. ReMMoC uses OpenCOM [3] 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"* [4].

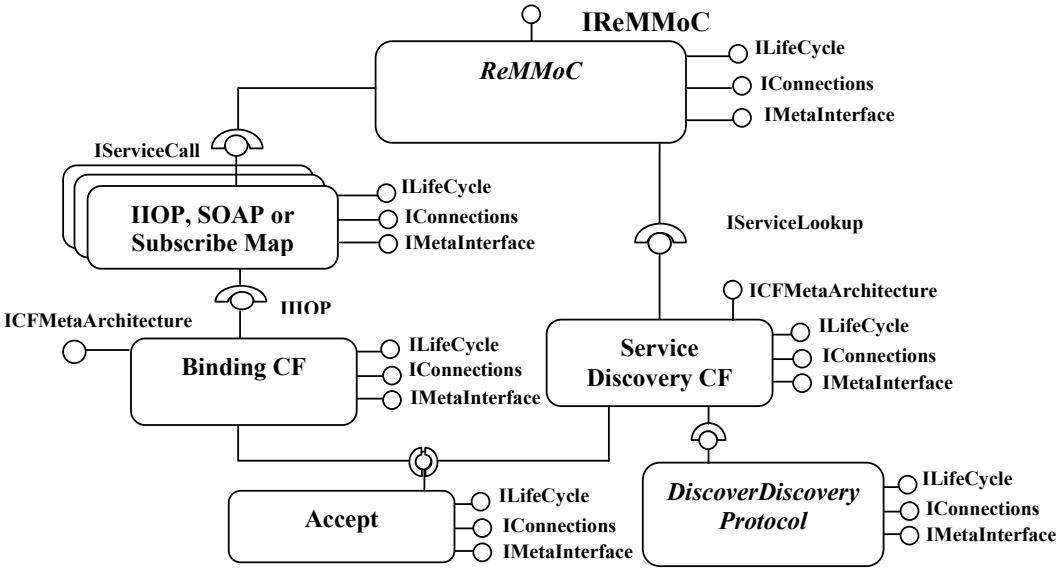

Figure 2. The ReMMoC Architecture

Figure 2 illustrates the architecture of ReMMoC, which 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. 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 ReMMoC component manages the reconfiguration of the underlying frameworks and provides an interface for discovering and interoperating with services (section 4.4).

4.2 The Binding Framework

The primary function of the binding framework is to interoperate with heterogeneous mobile services. 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 a single meta-architecture interface (ICFMetaArchitecture) that provides operations to inspect and dynamically change its internal structure.

Within the binding framework changes are made at two distinct levels. Firstly, each binding type implementation can be replaced. For example, an IIOP configuration can be replaced by a publish-subscribe configuration. Multiple personalities can also be created, e.g. 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. Secondly, fine-grained changes to each 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.

4.3 The Service Discovery Framework

The Service Discovery framework allows services that have been advertised by different service discovery protocols to be found. 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 DiscoverDiscoveryProtocol component in figure 2 monitors the environment and controls this reconfiguration.

The service discovery framework offers a set of generic service discovery methods. This includes a generic service lookup operation that returns the information from different service discovery protocol searches in a generic format. For example, a lookup of a weather service across two discovery configurations, e.g. UPnP and SLP, returns a list of matched services from both types. It is this information (the description of the service returned by the lookup protocol) that is used to configure the binding framework.

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.

4.4 The ReMMoC API

To allow clients to be developed independently of individual middleware types, ReMMoC offers an API based upon the abstract service definitions proposed by section 2. In particular, ReMMoC presents an event-based programming model that overrides the different computational models of each paradigm. Each abstract service operation is carried out and its result is returned as an event. For example, if that operation is executed by an RMI invocation or an event subscription the result is always an event. Similarly, service lookup operations return results as events. Figure 3 documents the API of ReMMoC, which consists of operations to: lookup services, lookup then invoke abstract WSDL operations, invoke operations on known services, or create and host service provider operations.

ReMMoC maps these API calls to the binding framework through the use of a reconfigurable mapping component, illustrated in figure 2. For example, an IIOP mapping component maps WSDL operations to IIOP invocations through the interface exposed by the binding framework; it can be replaced by a subscribe mapping component, which maps to subscribe requests. These components carry out the mapping of abstract to concrete operations described in section 2 (cf. Web Services). However, it is beyond the scope of this document to describe how each individual API call is mapped. By utilising different mapping components a greater range of middleware functionality offered by each implementation can be maintained.

```
interface ReMMoC_ICF : IUnknown {
        HRESULT WSDLGet (WSDLService* ServiceDescription, char* XML);
        HRESULT FindandInvokeOperation (WSDLService ServiceDescription, char* OperationName,
                        int Iterations, ReMMoCOPHandler Handler);
        HRESULT InvokeOperation (WSDLService ServiceDescription, ServiceReturnEvent
                ReturnedLookupEvent, char* OperationName, int Iterations, ReMMoCOPHandler  Handler);
        HRESULT CreateOperation (WSDLService ServiceDescription, ServiceReturnEvent
              ReturnedLookupEvent, char* OperationName, int Iterations, ReMMoCOPHandler Handler);
        HRESULT AddMessageValue(WSDLService *ServiceDescription, char* OperationName,
                char* ElementName, ReMMoC_TYPE type, char* direction,  VARIANT value);
        HRESULT GetMessageValue(WSDLService *ServiceDescription, char* OperationName,
                char* ElementName, ReMMoC_TYPE type, char* direction,  VARIANT value);          }
```

Figure 3. The ReMMoC API

5. Concluding Remarks and Future Work

In this paper, we have demonstrated the problem of middleware heterogeneity that exists in mobile computing environments, with the need to provide new techniques to allow mobile client applications to be developed independently of particular middleware implementations. 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 [5]. We argue that a marriage of web services with reflective middleware offers a solution to mobile client interoperability.

ReMMoC has been fully developed and tested using simple applications e.g. chat, news and stock quote clients across IIOP, SOAP and Publish-Subscribe bindings. Ongoing work includes an evaluation of this method on larger, complex applications and across a range of further middleware bindings e.g. tuple spaces, group communication and data-sharing to investigate the scope of its applicability.

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 a 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 based upon context information (e.g. components previously used at a location) is an interesting option.

Finally, Web Service description formats are emerging (e.g. WSEL) that extend abstract service descriptions to include non-functional aspects (e.g. quality of service and security). Languages to describe more complex interaction patterns are also available e.g. Web Services Flow Language. An investigation of the integration of these into ReMMoC is required.

6. References

[1] W3C. "Web Services Architecture", W3C Working Draft, http://www.w3.org/TR/ws-arch/. November, 2002.

[2] W3C. "Web Services Description Language (WSDL) Version 1.2", W3C Working Draft, http://www.w3.org/TR/wsdl12/. March, 2003.

[3] 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.

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

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