A Higher Level Abstraction for Mobile Computing Middleware

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 application developers now choose between many communication paradigms e.g. Remote Method Invocation, publish-subscribe, data sharing, mobile agents and tuple spaces. Each offers benefits to different application styles; however, their heterogeneity means applications and services implemented using different paradigms cannot interoperate. In this paper, we propose a higher-level abstraction based upon the Web Services Description Language (WSDL) that allows mobile clients to be developed independent of service implementations. To support this concept, a dynamic architecture (ReMMoC) that can reconfigure between paradigm implementations is required.

1. Introduction

New middleware technologies are emerging to explicitly support the field of mobile computing. Mobile hosts need to interact with other hosts and services that they discover at their current location. However, the mobile domain is characterised by the problems of wireless network performance, limited end-system resources and changing environmental context. Therefore, research into mobile computing middleware has produced a variety of solutions to overcome these issues and better support the development of mobile computing applications. These encompass different paradigms e.g. remote method invocation, publish-subscribe, tuple spaces and data sharing. However, developing a mobile client utilising a single paradigm is not feasible, because it will encounter services implemented upon alternative middleware as it moves from location to location. A similar argument applies to complementary middleware support services, such as: location, naming and service discovery. Therefore, there is a real need to provide an abstraction to enable mobile client application developers to deal with the multitude of service implementations in an integrated manner.

We propose that Web Services [1] can have a significant role to play here because its properties are well suited to middleware independence; it is already being utilised as the key technology in integrating existing heterogeneous middleware platforms [2]. At the core of the architecture is the Web Services Description Language (WSDL) [3]. This language separates abstract service definitions from concrete middleware implementation. Hence, this offers an interesting higher-level abstraction; also this alone is not enough. Support for context aware customisation of the underlying middleware is required to dynamically *adapt* between heterogeneous middleware. We argue that the web service abstraction must be complemented by the techniques of reflection, components and component frameworks to provide this mechanism. This paper documents the ReMMoC (Reflective Middleware for Mobile Computing) platform [4] that completes this implementation.

Recently related work on middleware integration has emerged. Bridges that map one middleware to another e.g. SOAP to CORBA [5] are available. Uniframe [6] dynamically generates bridges between discovered heterogeneous components. The Model Driven Architecture (MDA) [7] provides a 4GL approach to develop distributed solutions, hiding the generation of mappings between middleware components. Furthermore, the Web Service Architecture [1] has emerged as a key technology; its open standards provide a middle ground for middleware mapping, removing the need for complex direct integration [8, 9]. However, these technologies are designed for heterogeneity in the fixed network, hence they are static in nature and do not consider the diversity of middleware paradigms.

The structure of this paper is as follows. Section 2 details background information about the technologies of web services, reflection and components. Section 3 illustrates the ReMMoC middleware platform, including the higher-level programming abstraction it offers. Section 4 describes the important future work we intend to carry out and finally section 5 draws concluding remarks and identifies important areas of future work.

2. Background

2.1 Web Service Architecture

The intended goal of Web Services [1] is to allow different service providers to implement centrally defined service interfaces using 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 can then be developed to interoperate with either service upon dynamic discovery. Hence, the service interface must be defined abstractly from heterogeneous middleware paradigms. 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. WSDL [3] is an XML-based language that describes the exchange of messages (containing typed data items) between the service requestor and service provider. The fundamental property of WSDL is that it separates the abstract description of service functionality from the concrete details of the service implementation i.e. the abstract definition of a message exchange can be concretely implemented by different network protocols, i.e. precisely the property we seek. Therefore, WSDL, through abstract service descriptions, offers a higher-level communication paradigm. Developers do not have to overcome the different and unknown communication paradigms offered by heterogeneous service implementations encountered by users at run-time. Hence, mobile client applications can be created that interact within an environment whose properties are unknown to the developer (for example, a tourist application that can operate in locations around the globe). However, what is missing from Web Services is the capability to dynamically reconfigure between concrete implementations and allow continued interoperation from one implementation of the abstract service to another. We discuss an example reflective middleware, which provides this capability in section 3.

2.2 Reflection, Components and Component Frameworks

There is now a growing community working on the area of reflective middleware [10]. The motivation for this research is to overcome the "black-box" philosophy of many existing middleware platforms by providing more openness; and to achieve this in a *principled* manner through a comprehensive reflective architecture [11]. The key to the approach is to offer a meta-interface, or *meta-object protocol (MOP),* supporting access to the engineering of the underlying platform. This MOP provides operations to inspect the internal details of a platform (*introspection*), and by exposing the underlying implementation; it is also possible to insert behaviour. In addition, the MOP typically provides operations to alter the underlying middleware (*adaptation*). More generally, middleware platforms typically offer two (complementary) styles of reflection: *Structural reflection* is concerned with the underlying structure of systems, and *behavioural reflection* is concerned with activity in the underlying system.

In parallel with the above developments, there has been increasing interest in the role of *components* in distributed systems. According to Szyperski [12], a component can be defined as "a unit of composition with contractually specified interfaces and explicit dependencies only". In addition, he states "a software component can be deployed independently and is subject to composition by third parties". A key part of this definition is the emphasis on *composition*; component technologies rely heavily on composition rather than inheritance for the construction of applications, thus avoiding the fragile base class problem (and the subsequent difficulties in terms of system evolution) [12]. To support third party composition, they also employ explicit contracts in terms of *provided* and *required* interfaces. The overall aim is to reduce time to market for new services through an emphasis on programming by assembly rather than software development (cf. manufacturing vs. engineering).

Component frameworks [12] are reusable architectures that embody domain-specific constraints and strategies for composing components. The main contribution of component frameworks is that they provide a means of enforcing desired architectural properties and invariants by constraining the interactions among their plug-ins in a domain-relevant manner. As additional benefits, component frameworks simplify component development through design reuse, enable lightweight components, and increase the system's understandability and maintainability.

Reflection, component technologies and components frameworks are complementary technologies that can be used to create reflective middleware [11]. Reflection provides the necessary level of openness to access the underlying platform architecture, whereas components provide an appropriate structuring mechanism. The compositional approach inherent in components also provides a clean basis on which to re-configure the underlying architecture. Finally, component frameworks have the potential to impose appropriate constraints on this adaptation process.

3. Reflective Middleware for Mobile Computing (ReMMoC)

3.1 Overview

This section briefly describes the ReMMoC platform [4], a configurable and reconfigurable reflective middleware. ReMMoC uses OpenCOM [12] 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"* [11]. ReMMoC offers a higher-level abstraction that hides the complexity of programming using a dynamically changing binding paradigm.

Figure 1 illustrates the architecture of ReMMoC, which consists of two key component frameworks: (1) a binding framework for interoperation with services implemented upon different middleware types, and (2) a service discovery framework for discovering services advertised by different service discovery protocols. Our higher-level communication abstraction is implemented by the ReMMoC component, whose IReMMoC interface is documented in figure 2. This component also acts as a management component to the underlying elements by reconfiguring the underlying frameworks. Mapping components are dynamically plugged into the architecture (e.g. IIOP Map); these map from our abstraction to the current middleware abstraction. Hence, new paradigms can be dynamically introduced into the architecture. Section 3.3 describes these mappings further.

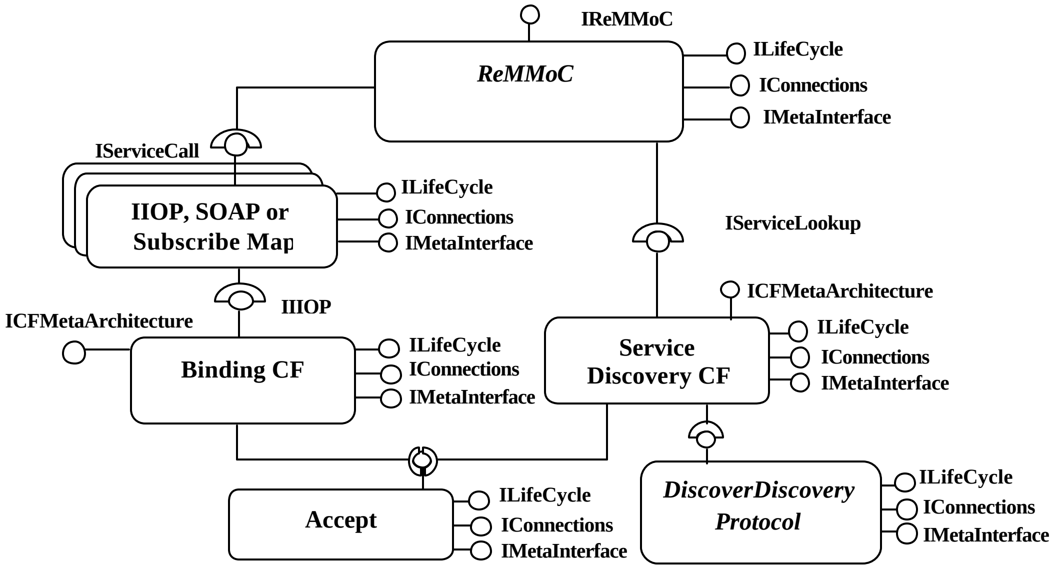

Figure 1. The ReMMoC Architecture

The primary function of the binding framework is to interoperate with heterogeneous 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 that provides operations to inspect and dynamically change its internal structure.

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 Universal Plug and Play (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 1 monitors the environment and controls this reconfiguration.

3.2 ReMMoC API

To program distributed client applications in ReMMoC, application developers must utilise the WSDL service definitions in a similar manner to IDL programming. That is, service developers implement the service to adhere to an abstract WSDL definition; therefore clients can invoke their abstract operations. Figure 2 illustrates the set of operations that make up the API; these methods allow abstract services to be discovered and their operations to be invoked. Furthermore, in order to override the different computational models of the underlying concrete heterogeneous paradigms and maintain a consistent information flow to the application, the programming model is event-based. Hence, when an abstract service operation is carried out, its result is returned as an event. Therefore, if the lower-level paradigm is RMI, publish-subscribe or tuple space its result is returned as an event to the application.

```
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 2. The ReMMoC API

3.3 Mapping Abstract Services to Concrete Binding Types

This section illustrates how the abstract operations of WSDL are mapped to two contrasting binding types (RMI and publish-subscribe). Four abstract operations can be defined 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 3 illustrates how input and output messages that make up WSDL operations map to the RMI and publish-subscribe communication paradigms. We assume that the concrete paradigms use the same set of types as the abstract definitions. For RMI, the input/output messages of Request-Response and Solicit-Response operations can be mapped directly to the corresponding synchronous RMI messages. 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 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 operation name maps to the event subject, the content of the input message is used to create a filter to receive the correct event, whose content maps to the output message. 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.

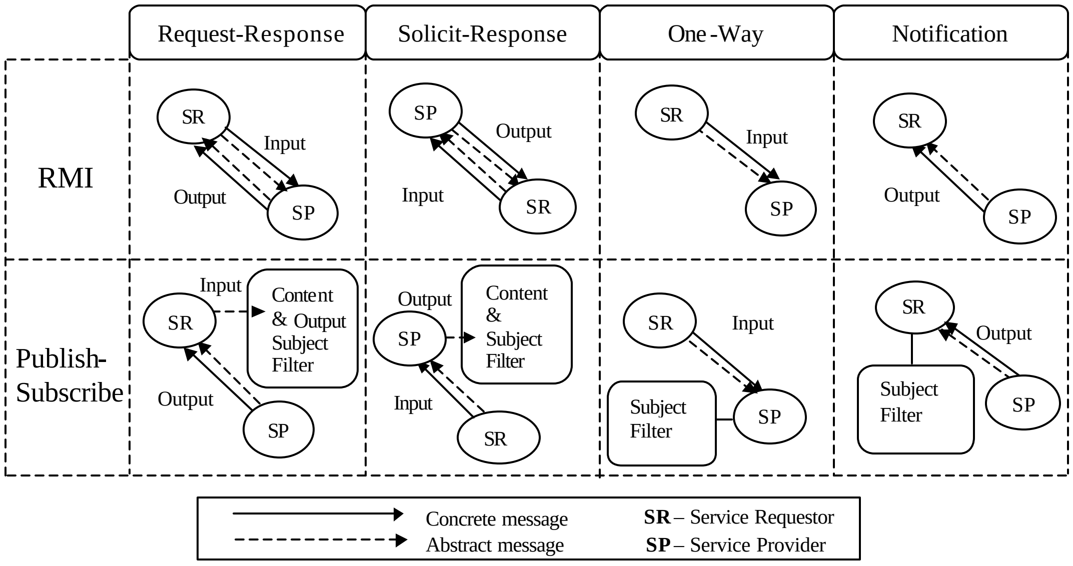

Figure 2. Mapping WSDL operations to different middleware paradigms

These mappings are implemented by the mapping components in figure 1. We hope to carry out further work in this area to map WSDL to both tuple-space and data-sharing paradigms to demonstrate the scope of this method.

4. Future Work

Our approach currently concentrates on developing distributed clients using a higher-level abstraction; the services themselves are fixed and implemented using standard paradigms. Hence, reconfiguration and mapping only takes place at the client side. We hope to identify scenarios where service side reconfiguration would be beneficial and extend both the reflective architecture and higher-level abstraction to include this. In this way, a server could manage requests from many heterogeneous client implementations.

Ongoing work includes an evaluation of this method on larger, complex applications in the domains of Grid computing, Ubiquitous computing, Smart Home Environments, and embedded real-time. To better support these a wider range of middleware implementations e.g. tuple spaces, group communication and data sharing will be implemented. By increasing the set of lower level implementations we can further evaluate our higher-level abstraction.

Our current approach is based upon the core of the Web Services architecture i.e. WSDL. However, new Web Service description formats are emerging (e.g. Web Service Endpoint Language) that extend abstract service descriptions to include non-functional aspects (e.g. quality of service and security).

Furthermore, 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.

5. Conclusions

Many middleware solutions are available to mobile application developers that provide heterogeneous communication paradigms. Therefore, developing applications upon a single solution is insufficient. Applications must be able to utilise a multitude of service implementations in an integrated manner. We have proposed that a higher-level abstraction is required to hide this complexity from the developer. The open standards (notably WSDL) offered by the Web Services architecture provide an interesting abstraction. The mobile environment requires the dynamic reconfiguration of middleware based upon the current environmental context; therefore the abstraction must be complemented by an adaptation mechanism to change between middleware. We propose that the combination of reflection, components and component frameworks provides such an open implementation.

The ReMMoC middleware platform combines reconfiguration and a higher-level abstraction to allow new mobile application types to be developed. At present, this middleware has been fully developed and tested using simple applications in the domain of mobile computing, e.g. chat, news and stock quote clients. In order to test and evaluate the ReMMoC platform, we have implemented IIOP client and server, SOAP client and Publish-Subscribe personalities and two service discovery protocol implementations: SLP and UPnP both with service lookup and registration capabilities. It is feasible for new protocols to be integrated into the architecture, provided that they are implemented as OpenCOM components. For a more detailed description of ReMMoC see [4].

6. References

[1] W3C. "Web Services Architecture", W3C Working Draft, http://www.w3.org/TR/ws-arch/. November, 2002.
[2] Vinoski, S. "Toward Integration Web Services Interaction Models", IEEE Internet Computing Online, 6(3), p89-91, May/June, 2002.
[3] W3C. "Web Services Description Language (WSDL) Version 1.2", W3C Working Draft, http://www.w3.org/TR/wsdl12/. March, 2003.
[4] Grace, P., Blair, G. and Samuel, S. "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability", International Symposium on Distributed Objects and Application, Catania, Italy, November 2003 (to appear).
[5] SOAP2CORBA and CORBA2SOAP. http://soap2corba.sourceforge.net/
[6] Raje R., Bryant B., Auguston M., Olson A., Burt C. "A Unified Approach for Integration of Distributed Heterogeneous Software Components", In Proceedings of the 2001 Monterey Workshop Engineering Automation for Software Intensive System Integration, pp. 109-119, 2001.
[7] Object Management Group. "Model Driven Architecture", document number ormsc/2001-07-01, July 2001.
[8] Vinoski, S. "It's just a Mapping Problem", IEEE Internet Computing Online, 7(3), p88-90, May/June, 2003.
[9] Fremantle, P. "Applying the Web Services Invocation Framework – Calling Services Independent of Protocols", IBM DeveloperWorks, June 2002.
http://www.ibm.com/developerworks/webservices/library/ws-appwsif.html
[10] Kon, F., Costa, F., Blair, G.S., Campbell, R., "The Case for Reflective Middleware: Building Middleware that is Flexible, Reconfigurable, and yet simple to Use", CACM, Vol. 45, No. 6, 2002.
[11] 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", IEEE DS Online, Special Issue on Reflective Middleware, Vol. 2, No. 6, 2001.
[12] Szyperski, C. "*Component Software: Beyond Object-Oriented Programming*". Addison Wesley, 1998.
[13] 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.