Enabling Mobile Access to Adaptive Services and Portal

By Gia Hieu Dinh and Yan Liu

21/01/2008
Abstract

Currently, enhancing the interoperability of Web 2.0 Technologies over mobile devices remains a challenge to software engineering. Resource limitation of mobile devices and non-adaptability of web server application has restricted the interaction between the two aforementioned technologies causing clients difficulties in handling the response in an appropriate manner.

This project proposes an Adaptive Service Middleware Architecture liable to enhance interoperability of Web 2.0 Technologies over mobile devices through minimizing the barrier from web server application non-adaptability. The architecture makes a twofold contribution to enable the mobile access to web services and interactive portals. As for web services, a proof-of-concept implementation have been developed to demonstrate the adaptiveness of the architecture while invoking a third-party Amazon Web Services. A number of performance measurements have also been conducted based on this implementation to evaluate the performance overheads of the adaptive components. The results have proved that the overheads that adaptive components incurred are insignificant to the non-adaptive approach. In addition, the architecture can also be extended to an interactive portal which is accessible from mobile devices. This portal can adaptively render other third-party portlets that follow the Java Portlet Specification (JSR-168) standard and Web Services for Remote Portlets (WSRP) standard.

Once the architecture is implemented, it can act as a transparent middleware that enhances the adaptiveness of the servers to minimize the resources required from mobile devices.
Table of contents

Abstract ......................................................................................................................... 1
Table of contents ........................................................................................................... 3
List of Figures ............................................................................................................... 5
List of Tables ................................................................................................................ 6
CHAPTER 1: Introduction ............................................................................................. 7
1 Introduction ............................................................................................................... 8
  1.2 Overview and Motivation .................................................................................... 8
  1.3 Project Objectives and Scopes ......................................................................... 9
  1.4 Thesis Organization ......................................................................................... 11
CHAPTER 2: Background and Related Work ............................................................. 13
  2.1 Chapter overview: ......................................................................................... 14
  2.2 Adaptive Computing ...................................................................................... 14
    2.2.1 What is adaptive? ................................................................................... 14
    2.2.2 Characteristics of Adaptive Computing System .................................. 15
    2.2.3 Software adaptation approaches ......................................................... 15
    2.2.4 Compositional Adaptation Taxonomy .............................................. 16
    2.2.5 Why Autonomic Computing is Vital to Mobile Programming? .... 19
  2.3 Web Service .................................................................................................... 20
    2.3.1 What is Web Service? ........................................................................... 20
    2.3.2 Web Service Architecture: ................................................................. 20
  2.4 Portal, Portlets and Web Services for Remote Portlets Specification ....... 21
    2.4.1 What is Portal? ..................................................................................... 21
    2.4.2 What is Portlets? .................................................................................. 22
    2.4.3 What is Web Services for Remote Portlets Specification? .......... 22
    2.4.4 Advantages of WSRP approach over Web Services Approach .... 22
  2.5 Related Works ................................................................................................. 25
    2.5.1 Offloading Computation using Java Servlet ...................................... 25
    2.5.2 Adaptive proxy for Mobile Applications ........................................... 26
CHAPTER 3: Middleware Architecture ....................................................................... 27
  3 Middleware Architecture ....................................................................................... 28
    3.1 Chapter overview: ....................................................................................... 28
    3.2 Architecture Design Patterns ...................................................................... 28
      3.2.1 M-A-P-E Pattern ............................................................................... 28
      3.2.2 Policy Point Pattern ......................................................................... 30
    3.3 Architecture Candidate ............................................................................... 31
      3.3.1 General architecture ......................................................................... 31
      3.3.2 The Lightweight Portal Component ............................................... 32
      3.3.3 Adaptive Portal Component .............................................................. 35
      3.3.4 Apache Axis2 Web Services .............................................................. 41
    3.4 Use case Diagram ........................................................................................... 43
      3.4.1 Use case 1: ......................................................................................... 43
      3.4.2 Use case 2 ........................................................................................ 44
      3.4.3 Use case 3 ........................................................................................ 47
List of Figures

Figure 1 - Dynamism of Adaptive Component based on when it is composed ........... 17
Figure 2 - Web Service Architecture ........................................................................ 21
Figure 3 - Traditional approach of rendering portlet ................................................ 23
Figure 4 - WSRP Approach for Rendering Portlets ................................................... 24
Figure 5 - Computation Offloading Scheme Using Proxy for Mobile Web Services . 25
Figure 6 - Adaptive Proxy for Mobile Device ............................................................ 26
Figure 7 - Control loops in generic autonomic computing architecture .................... 29
Figure 8 - Policy Point Pattern ................................................................................ 30
Figure 9 - Component Architecture ........................................................................ 32
Figure 10 - Lightweight Portal Component Architecture ......................................... 33
Figure 11 - Apache Pluto Portal Architecture .......................................................... 36
Figure 12 - Adaptive Portal Component Architecture .............................................. 38
Figure 13 - Apache Jetspeed2 Pipeline ..................................................................... 40
Figure 14 - Apache Jetspeed2 Valves ...................................................................... 41
Figure 15 - Apache Axis2 SOAP Processing Model .................................................... 42
Figure 16 - Adaptive Web Service Architecture ...................................................... 43
Figure 17 - Lightweight portal handling incoming request ........................................ 44
Figure 18 - Adaptive Portal Component handles incoming request ............................ 46
Figure 19 - Lightweight Portal Component displays HTML markup response .......... 47
Figure 20 - Adaptive Portal Component handles request for remote portlet .............. 49
Figure 21 - MEMS Steps .......................................................................................... 52
Figure 22 - Rating Calculation with Sub-Categories .................................................. 64
Figure 23 - Deployment Architecture of Adaptive Middleware ............................... 66
Figure 24 - Deployment of Lightweight Portal Component ....................................... 67
Figure 25 - Deployment of Adaptive Portal Component ......................................... 68
Figure 26 - Deployment of Apache AXIS2 Web Services with Adaptive Handler .... 69
Figure 27 - Summary of Component Overhead under Different Stressful Conditions 72
Figure 28 - Overall Evaluation Results of the Adaptive Middleware Architecture .... 77
List of Tables

Table 1 - Quality Attributes Scale Definition ..............................................................61
Table 2 - Summary of Component Overhead under Different Stressful Conditions...72
CHAPTER 1: Introduction

About this chapter

This chapter provides an overview on the thesis topic of Enabling Web 2.0 technology on mobile devices platform, the motivation of this project, its scopes, objectives, and structure of the thesis.
1 Introduction

1.1 Chapter overview

This chapter is divided into the following sections:

Overview and motivations: provide an overview and motivation of the topic “Enabling mobile access to adaptive web services and interactive portlets”.

Project objectives and scopes: clearly identify the objectives and scopes of the project.

Thesis Organization: provides the structure of the project.

1.2 Overview and Motivation

Recently, Web 2.0 is recognized as an era when leadership in the computer industry has passed from traditional software companies to a new kind of Internet Service Company. The Internet has replaced the personal computer (PC) as the platform that matters, as PC replaces mainframe and minicomputer in 1980s (O'Reilly, 2005). The web-based companies, such as Yahoo!, EBay, Amazon, and Google are gaining more and more market values in software industry. Products from these web giants are usually native web applications (such as Web Services, Web APIs), which are never sold or packaged but delivered as paid services (O'Reilly, 2005).

One of the key features of Web 2.0 that deserves a mention is that Web 2.0 Applications should no longer be limited to the PC platform (O'Reilly, 2005). Mobile devices have proved to be a promising development platform for web 2.0 technologies due to the improvements of mobile devices’ features during the last few years. Two obstacles to usability of web applications on mobile devices, including the inefficiency of user navigation and lack of computability from mobile devices, have been removed. However, there are still problems that might prevent accessibility to web 2.0 technologies from mobile devices such as the lack of adaptiveness from Web 2.0 Server side and the resource limitation from mobile devices’ client side.
Although the mobile computability has been improved significantly, it is still not feasible to deploy a fully-functioned interactive web application on mobile devices. Other resources from mobile devices (i.e. screen resolution, stability of the network connection, limited battery life, bandwidth availability for mobile devices, and so on) are still very limited. A fully-functioned web application therefore is too resource-hungry to be applied to mobile applications. In this situation, a lightweight, adaptive web application could be considered as a feasible solution to be deployed for on mobile devices.

The lack of adaptiveness from server side could be considered as a problem that will affect the interoperability of web 2.0 technologies on mobile. Web 2.0 applications are delivered as services and are invoked remotely by consumer’s application (for example: web services, remote portlets and so on). From service consumer’s viewpoint, Web 2.0 applications are loosely coupled components of their application and can be invoked via the Internet. However, as a remote component, these services may not be consumed effectively in some situations, especially while consumer’s resources are limited, such as in the case when consumers are deployed in mobile device platform. For example: in mobile device platform, mobile clients cannot handle large amount of data coming out from server; the displayable area of mobile devices is limited and so on. These may cause the client many difficulties in handling the response from Web 2.0 applications in an appropriate manner. This project proposes an Adaptive Service Middleware Architecture liable to enhance interoperability of Web 2.0 Technologies over mobile devices through minimizing the barrier from web server application non-adaptability

1.3 Project Objectives and Scopes

This is a twelve-credit-point capstone project whose main objective is to enable mobile access to third-party web services and remote portlet adaptively. The project will suggest a middleware architecture that enhances the adaptiveness of server side to minimize the resources required from mobile devices. This project is one step to make Web 2.0 Technologies become widely adopted in mobile platform. In particular, this project will focus on the following issues:
Propose an adaptive services middleware architecture that minimizes the resources required from mobile devices by enhancing the adaptiveness from server side while invoking third-party services (including web services and web portlets)

Although there are many third-party web services and remote portlets available over the Internet, the adaptiveness that they provide is quite limited, and is entirely relied on the developer to be manually invoked inside web applications. The reusability, interoperability and maintainability of these adaptive components therefore are in a very humble scope. The first focal point of the project is to propose a middleware architecture with reusable, interoperable and maintainable adaptive components. This middleware architecture makes a twofold contribution to enabling the mobile access to both third-party web services and remote portlets.

Implement a proof-of-concept prototype to demonstrate the adaptiveness of the architecture while invoking Amazon Web Services.

The second part of the project is to develop a case study that demonstrates the adaptiveness of the architecture. This case study focuses on how adaptive the proposed architecture is while invoking third-party web services, such as Amazon Web Services. A proof-of-concept prototype is implemented. The prototype includes 3 components:

- A lightweight web portal that is accessible from mobile devices using Visual Studio 2005 and ASP.NET 2.0
- An Apache Axis2 Web Services that acts as a wrap-up component, handling requests and responses from the third-party web services. The adaptive components are embedded inside this Axis2 architecture.
- An Apache Pluto Portal that provides a user-interface to the Apache Axis2 Web Services. This can also be another entry point of the system, separating the business logic (Apache Axis2 Web Services) and client interface (ASP.NET lightweight portal)
The MEMS (Method for Evaluating Middleware architectures) could be used to evaluate the quality attributes of the proposed architecture based on this case study. However the case study which demonstrates the adaptiveness of the architecture when invoking third-party web portlets is out of the scope of this project and it can be left for future developments.

1.4 Thesis Organization

This thesis is divided into eight chapters:

**Chapter 1: Introduction**
This chapter provides an overview on the thesis topic of Enabling Web 2.0 technology on mobile devices platform, the motivation of this project, its scopes and objectives and structure of the thesis.

**Chapter 2: Background and Related Works**
This chapter provides background knowledge about adaptive computing, why adaptive computing is vital to mobile computing, an overview about web services and portal application, and finally some current approaches to enable mobile access to web services.

**Chapter 3: Middleware Architecture**
This chapter proposes an adaptive middleware architecture, which can enable mobile access to third-party web services and remote portlets adaptively. Each component in the proposed architecture will be discussed in detail in comparison with the design patterns.

**Chapter 4: Middleware Architecture Evaluation Method**
This chapter provides an overview of the method for evaluating middleware architecture quality attribute – the Method for Evaluating Middleware architectures (MEMS).
Chapter 5: Case Study: Enable Mobile Access to Amazon Web Services
A case study of the proposed architecture should be evaluated based on the MEMS process. The major quality attributes that the case study wants to evaluate include: performance, usability, maintainability, portability, programmability and dynamism of adaptive components.

Chapter 6: Conclusion
This chapter reviews the research and development being carried out during the project. It also discusses the areas of research that can be conducted in the future.

References
Reference section provides the list of references used for this project.

Appendices:
Appendix A: Glossary of the project.
Appendix B: Results of Overhead of Adaptive Component Measurement
CHAPTER 2:

Background and Related Work

About this chapter

This chapter provides background knowledge about adaptive computing, why adaptive computing is vital to mobile computing, an overview about web services and portal application, and finally, some current approaches to enable mobile access to web services.
2 Background and Related Work

2.1 Chapter overview:

This chapter is divided into the following sections:

Adaptive computing: clearly defines the term “adaptive”, identifies the main characteristics of an adaptive application; determines the criteria to measure the effectiveness of adaptive components. This also discusses the important of adaptive computing in mobile programming.

Web Services: defines the term “Web Service”, identifies the key technology that enable web services and other related fields.

Portal, Portlets and Web Services for Remote Portlets Specification: defines the term “Portal”, “Portlet” and the WSRP specification, analyses the advantages of WSRP Specification over the traditional web service approach.

Related Work: reviews the related researches, identifies the weakness from these researches.

2.2 Adaptive Computing

2.2.1 What is adaptive?

In mid-October 2001, IBM released a manifesto determining that the main obstacle to further progress in the IT industry is an exponential increase of software complexity. The IT environments are complex, heterogeneous tangles of hardware, middleware and software from multiple vendors, which make it impossible for IT professionals to install, configure, tune and maintain effectively. Kephart and Chess (2003) point out that “the only option remaining is autonomic computing – computing system that can manage themselves given high-level objectives from administrators”. Hewlett-Packard’s Adaptive Enterprise initiative and Microsoft’s Dynamic Systems are industry researches on the vital of self-managing components and system to the future of IT industry.
2.2.2 Characteristics of Adaptive Computing System

An autonomic computing or self-managed system is defined to be capable of self-configuration, self-adaptation and self-healing, self-monitoring and self-tuning and other self-* characteristics (Kramer and Magee, 2007).

For instance, considering that a set of software components is implemented to fulfil the set functional requirements, the aim of self-configuration system is that these components should be able to either configure themselves such that they satisfy the specification or be capable of reporting that they cannot.

On the other hand, self-adaptation and self-healing refers to the ability of the system to reconfigure itself to satisfy the changes in system requirements and/or environment. Normally, any change of the requirements or environment will imply an off-line process in which the system will be re-configured to adapt these changes. However, dynamic change, which occurs while the system is still operational is far more demanding, and requires that the system evolves dynamically. In other words, the self-adaptation should occur at run-time.

Finally, the requirement specifications of the system includes not only functional requirements but also non-functional requirements such as response time, performance, reliability, efficiency and security, which may requires optimization from the system. In order to fulfil these non-functional requirements, an adaptive system should be self-monitoring the changes of environment and self-tuning to adapt to those changes.

2.2.3 Software adaptation approaches

There are two general approaches to implement software adaptation, including parameter adaptation and compositional adaptation. “Parameter adaptation modifies program variables that determine behaviour” (McKinley et al., 2004). An often-cited example of parameter adaptation is the capability of Internet’s Transmission Control Protocol to adjust its behaviour by changing values that control window management and retransmissions in response to apparent network congestion. However, parameter adaptation is proved to have a significant disadvantage. It now allows new algorithms or components to be added to an application after the original design and construction. It can tune parameters or modify an application based on different existing strategies to adapt to the changes in environment, but it cannot adopt new strategies itself.
Because of this limitation, the second approach - compositional adaptation is more preferred. “Compositional adaptation exchanges algorithmic or structural system components with others that improve a program’s fit to its current environment” (McKinley et al., 2004). Compositional adaptation allows new algorithms, components be added to the existing system and solve the problems which are unforeseeable during development process. This flexibility makes adaptive components more than simple tuning program variables or strategy selection. It enables dynamic recomposition of software in real-time, for example switch program components in or out of a resource-limited devices, or add new adaptive components to a deployed systems.

2.2.4 Compositional Adaptation Taxonomy

Recently, researchers and developers have proposed a wide variety of methods for implementing adaptive components. In their article, McKinley et al. (2004) have suggested a taxonomy to evaluate the effectiveness of these different methods based on how, when and where the adaptive components take places during the development process.

The first dimension of the taxonomy addresses the specific software mechanisms to enable the compositional adaptation (i.e. how the adaptive components are composed). Some key techniques that enable adaptation in different stages of the development progress are function pointers/delegates, proxies, middleware interceptions, integrated middleware technique. The effectiveness of these software mechanisms could be evaluated based on transparency of components. The transparency refers to whether an application or system is aware of the adaptive components invoked to adapt to the changes of environment. For example, a middleware approach is transparent with respect to the application source code if the application does not need to be modified to take advantage of the adaptive feature. Different degrees of transparency (with respect to application source, virtual machine, middleware source, and so on) will determine the portability of the components across platforms, and the ease of implementing/adapting new components to the existing programs.

Another classification criterion to evaluate the effectiveness of the adaptive components is when the adaptive component is composed with the business logic. In fact, the later this composition takes place, the more effective the adaptation
Enabling mobile access to adaptive services and portal

component is. However, the late composition of adaptive components also leads to the more complicated in ensuring consistency of the adapted program. The run-time composition of adaptive components is very powerful, but it is difficult to use traditional testing and formal verification methods to ensure the safety or other correctness properties of the system. Figure 1 has summarized the dynamism of adaptive components with respect to when it is composed with the application. The static composition implies that the adaptive components are composed at development, compile or load time, while the dynamic composition refers to the adaptive components are composed and applied at runtime.

![Figure 1 - Dynamism of Adaptive Component based on when it is composed](Source: (McKinley et al., 2004))

- An adaptive component is *hardwired* to into a system if it is composed at development time, with the rest of the system. The adaptive component could not be modified, extended, removed without recoding of the system.
- Alternatively, a limited form of adaptation can be implemented at compile time or link time by configuring application for a particular environment. This *customizable* application requires only recompilation or re-linking of the adaptive components to fit to new environment.
- *Configurable* applications delay the final decision on algorithmic units to use in the current environment until a running application loads the corresponding components. When the application requests the loading of new environment, it will select from a list of adaptive components with different capabilities or
Enabling mobile access to adaptive services and portal

implementation to choose the most suitable one for the current needs. For example, if an application runs on mobile devices, the runtime system might load a minimal display component to guarantee proper presentation on that mobile device.

- **Tunable** application uses dynamics composition method that prohibits the modification of business logic. Instead, it fine-tunes the adaptive components to response to the changes of environment conditions, such as dynamic conditions encounter in mobile device platform.

- **Mutable** application allows the adaptive components to change even the business logic that enable dynamic recomposition of running program into a functionally different system. While very powerful, this flexibility will raise a system’s integrity across the adaptations.

The final dimension in which McKinley et al. (2004) compare the effectiveness of adaptive components is where in the system the components are embedded inside the middleware architecture. One approach is to construct a layer of adaptable communication services. An example is the Adaptive Communication Environment (ACE) that used service wrappers and C++ dynamic binding to support adaptable inter-process communication and event handling services. The second approach is to provide a virtual machine with facilities to intercept and redirect interactions in the functional code. For example: Java Virtual Machine (JVM) and common language runtime (CLR) are two virtual machines that could facilitate dynamic recomposition through refection facilities provided by Java language and .NET platform. However, the use of virtual machine to provide transparency to the application may reduce the portability of the adaptive component on other virtual machine. Another approach could be considered is introducing adaptive component in higher middleware layer such as distribution layer, common services, domain-specific services, and so on. This approach enhances the portability of these adaptive components across virtual machines. It typically involves middleware components which directly intercept message associated with remote method invocations and redirect or modify them adaptively based on current conditions.
To sum up, the effectiveness of the adaptive components could be evaluated based on how, where and when the adaptive components is embedded into the system.

### 2.2.5 Why Autonomic Computing is Vital to Mobile Programming?

Mobile devices such as Pocket PC, Smartphones or generally Personal Digital Assistants (PDAs) are becoming more and more popular. The combined use of wireless network on these mobile devices makes it possible for devices’ owners to access the Internet on demand. However, developing web applications targeted mobile device platform still a challenging problem. For example such devices may encounter temporary loss of network connectivity problems when in movement. They also have very limited resources available, compare to the desktop platform, such as poor battery life, slow CPU speed, little memory, which have to be consumed in an efficiently manner (Costa et al., 2007). Therefore, in order to successfully develop a mobile distributed web system on top of network layer, application designers and developers has to deal resource-limitation natural of the mobile devices platform. Moreover, the availability of these resources may also change significantly during the system operation such as rapid and significant fluctuation in communication link quality, frequent network disconnections, device resource restriction, power limitation, and so on. The mobile application then needs to be capable of adapting those changes to offer the best possible level of services to users. For example: under low available bandwidth, the system should only allow email-reader to read the body of small email message, and avoid the large one; or the resolution of the incoming image will be resized, rescaled to fit the resolution of the mobile devices. In order to have a fine-grained adaptation, the mobile application needs to handle the changes in mobile devices environment, including:

- Device information: display size and capabilities, battery power status, memory, CPU usage.
- Information about the wireless link: bandwidth, latency, error rates;
- Infrastructure information: reachable servers/services
- Location-related environment: physical location, logical location (IP address, subnet) (Kunz and Black, 1999)
Some may argue that it is possible to develop a mobile application given under the worst case of the environment conditions. However, it is not reasonable to design such a client that provides user with a consistent functionality, despite the environment conditions (Kunz and Black, 1999). Whenever possible, users should be able to experience the best services achievable. That solution deprives the user of full services and functionality. For example, with this worst-case scenario application, inline images will always be displayed in low-resolution greyscale, even in the presence of enough bandwidth to download full-resolution colour images.

2.3 **Web Service**

2.3.1 **What is Web Service?**
World Wide Web Consortium (W3C) defines Web service as “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL)” (W3C, 2004).

2.3.2 **Web Service Architecture:**
According to this architecture, the invocation of web services actually is a client-server communication using Extensible Markup Language (XML) messages that follow the SOAP standard. The UDDI (Universal Description, Discovery and Integration) has been used as a directory, where service requesters can browse their required services using WSDL (Web Service Description Language) standard.
UDDI is a platform-independent, XML-based registry in which businesses can list their web services on the Internet. Web services could be written in using different frameworks such as Apache Axis2, Java Web Services, or Windows Communication Foundation framework, but they could be remotely invoked in any platform using SOAP message. It maximizes the interoperability of the web services on different platforms.

2.4 Portal, Portlets and Web Services for Remote Portlets Specification

2.4.1 What is Portal?
Web Portal is a Web-based application that presents information from different sources in a unified way. The web portal provides the opportunity to develop intelligent, personalized workspaces for end-users. This also integrates the relevant resources of information and provides tool to manage the underlying connection that make the information valuable to end-users (Delphi, 2002). Some others fundamental features of portal are single point of entry, integration, federation, personalization and access control. (Wikipedia, 2007)
2.4.2 What is Portlets?
Portlet is a pluggable user interface components that is managed and displayed inside a web portal. Portlet produce fragment of markup code (i.e. HTML fragment) that are aggregated into the portal page. In order to maintain the interoperability of portlet across different portal vendors, the Java Portlet Specification (JSR168) has been introduced. This defined a set of APIs for interaction between portlets and portal containers in personalization, presentation and security.

2.4.3 What is Web Services for Remote Portlets Specification?
The Web Services for Remote Portlets (WSRP) Specification is a product of the Organization for the Advancement of Structured Information Standards (OASIS, 2005), which is a consortium that facilitates the adoption of technical standards. Version 1.0 of the WSRP Specification was finalized in August of 2003. Web Services for Remote Portlets (WSRP) is a network protocol that is designed for communication between remote portlets. While JSR168 may be used to defined portlet, WSRP may be used to define portlet’s operations to remote container. This enhances the interoperability of the portlet across different vendors. For example, the .NET portlet could be created for the use of WSRP and it could be remotely rendered by Java Portal (such as JBoss Portal).

2.4.4 Advantages of WSRP approach over Web Services Approach
As mentioned before, the web services offer a mechanism to create platform-independent, loosely couple services, and JSR-168 defines a standard to develop the portlets, a question may arise: what is the purposes of WSRP Specification?
Traditionally, if the component wants to be invoked remotely inside a third-party Web Portal, it has to be exposed as a web service. The third-party portal render should invoke that web service and embedded it into the local portlets. The JSR168 standard might be used to create that local portlets and rendered inside the portal.

In this traditional approach, only the back-end service could be reused using Web Service. This approach reveals two problems. The first problem is the overhead added from consumer side while invoking the remote services. For each Web Service that it wants to invoke, a new portlet has to be developed as the user-interface for that service. If the service is invoked by multiple portal vendors, this overhead is significantly high. Moreover, another process has to be accounted for is the deployment of the newly developed portlet and Web Service client into the rendered portal. While reusing the third party web services may cut down the development time, the process of developing portlet and deploying on the portal server is more time-consuming (Castle, 2005).
The WSRP Specification could be used to solve these two problems. The WSRP Specification allows the third-party Portal to reuse not only the back-end services but also the entire interface. Instead of going through a front-end development process, the third-party portal now can render the portlets from more than one source and integrate it inside the portal without programming overhead.

The WSRP Specification defines the following actors within WSRP architecture:

- **WSRP producer**: This is the Web service that offers one or more portlets and implements WSRP interfaces. Similar to Web Service Provider, WSRP Producer is described using a standardized WSDL document.

- **WSRP Portlet**: A WSRP Portlet is a pluggable interface component that is exposed by WSRP producer and could be remotely rendered by third-party portal. However, WSRP portlet is not web service that can be invoked directly. It needs to be accessed through the producers.

- **WSRP Consumer**: This is the third-party portal that can render the WSRP portlet from its producer.
2.5 Related Works

Currently there are a number of researches on implementing web services on mobile platform.

2.5.1 Offloading Computation using Java Servlet

Early model of the mobile Web Services was based on a computation off-loading scheme, called wireless portal network architecture (Figure 5) (Yuan and Long, 2002). A Servlet is a Java object that uses the functionalities provided by Java platform to receive HTTP requests and generate corresponding HTTP responses. By using Servlet as a gateway (proxy), mobile application developers can build a thin Web Service client on the mobile devices. The client sends an HTTP requests to the gateway, and the gateway in turn will create the corresponding SOAP request and deliver it to the Service Providers.

Given that the wireless network portal architecture combined with thin client is easy to implement, they are do not fit for dynamic Web services for the following reasons:

- Thin clients offer limited user interfaces and these user interfaces are not customizable to fit user purposes. (Yuan and Long, 2002)
- Thin clients do not remember complex application state information; thus, the user must navigate through multiple pages to complete complex tasks, such as transactions. Not only is that inconvenient to the user, but it is also slow and...
uses precious wireless bandwidth. Moreover, it renders thin clients vulnerable to network interruptions. (Yuan and Long, 2002)

2.5.2 Adaptive proxy for Mobile Applications

Kunz and Black (1999) have also suggested an adaptive architecture to enable mobile access to web applications. They have proposed an adaptive architecture to implement a proxy between client (mobile devices) and server (Proxy Server). However, implementing adaptive components as a proxy could reduce the dynamism and adaptiveness of these components (i.e. these adaptive components can only self-tuning the requests/responses). As discussed in section 2.2, adaptive system is more than a self-tuning system. Adaptive system could also be self-configured, self-managed, and other characteristics under self-* categories, which can not be done using the adaptive proxy.
CHAPTER 3:

Middleware Architecture

About this chapter

This chapter proposes an adaptive middleware architecture, which enable mobile access to third-party web services and remote portlets adaptively. Each component in the proposed architecture will be discussed in detail in comparison with the design patterns.
3 Middleware Architecture

3.1 Chapter overview:

This chapter is divided into the following sessions:

**Architecture Design Patterns**: discuss the two design patterns that could be applied the proposed architecture including M-A-P-E pattern and Policy Point Pattern.

**Architecture candidate**: propose an architecture candidate using the aforementioned design patterns.

**Use case diagram**: describes how the architecture handles the incoming requests from mobile client browser.

3.2 Architecture Design Patterns

This section will discuss two architectural design patterns, including M-A-P-E pattern and policy point pattern, which are particularly designed for adaptive systems.

3.2.1 M-A-P-E Pattern

M-A-P-E pattern is derived from the IBM Autonomic Computing architecture blueprint (IBM Online article, 2004), which is originally designed as reference architecture for autonomic systems rather than as a design pattern. M-A-P-E stands for four core functions that organize the structure of an autonomic system including *Monitor, Analyse, Plan* and *Execute*. 
These four core functions work together to provide a control loop functionality.

- **Monitor**: The *monitor* function collects details from operating conditions, such as network condition, topology information, metrics, configuration property settings, client-side capacities and so on. The information will be collected and correlated into symptom. These symptoms then will be passed to analyse function.

- **Analyse**: The *analyse* function provides mechanisms to receive the symptoms from Monitor function, observe and analyse these symptoms to determine where changes needs to be made. If the changes are required, *analyse* function will generate a change request and pass it to the Plan function. The change request describes the modifications that the system needs to make to adapt to the changes in the Monitor function.

- **Plan**: the *plan* function creates or selects a procedure that is suitable to fulfill the change request from the analyse function. Then the plan function will generate the appropriate change plan, which represents a desired set of changes to the system. This change plan will be passed to the execute function.
Enabling mobile access to adaptive services and portal

- **Execute**: the *execute* function provides mechanism to schedule and perform the necessary changes to the system based on the change plan.

The data used by these four core functions (monitor, analyse, plan and execute) are stored as a shared knowledge. The shared knowledge includes data such as topology information, historical logs, metrics, symptoms and policies.

### 3.2.2 Policy Point Pattern

The policy point pattern is another pattern that is important in adaptive system construction. In the adaptive system, policies specify the goals of adaptive strategies. A policy is a representation of desired behaviour or constraints on behaviours defined in a standard external form. A decision whether a modification should be required is depended on information collected and policy specifications. The policy point pattern models constructs and behaviours for managing and coordinating policies.

![Policy Point Pattern Diagram](image)

*Figure 8 - Policy Point Pattern*

The policy point pattern has three major entities, including policy definition point (PDP), policy information point (PIP) and policy enforcement point (PEP).

- PDP needs to refer to more than one policy when making decision.
• PIP is the entity that collects the information, but it does not return any decision.
• PEP actually performs the actions and policy enforcement.

As each policy is defined as an individual entity, which collects information, makes decisions and enforces policy in its own modules, it is necessary to separate these tasks into different components. The collection information should be reused in different policies and analysis algorithms. Moreover the PIP, PDP and PEP should be implemented as an interceptor, which is a middleware mechanism to intercept and modify requests and response, capture execution context and inject actions.

### 3.3 Architecture Candidate

#### 3.3.1 General architecture

The M-A-P-E and policy point patterns are closely related together in an adaptive architecture. The M-A-P-E control loop will depend on the policy point pattern to produce an accurate decision. The policy point pattern also relies on the M-A-P-E pattern, for example, the PIP component could be integrated with *Monitor* function in M-A-P-E pattern, which shares the same functionality of collecting internal and external conditions. The policy enforcement overlaps with execute function in M-A-P-E pattern, and the policy definition could be merged with *analyse* and *plan* functions in the M-A-P-E pattern. Based on the M-A-P-E pattern, the proposed architecture will be designed to enable mobile access to third-party web services and third-party portlets adaptively. The Policy Point Pattern is currently not embedded inside this architecture due to the lack of external policy components. However the architecture can be extended to handle the external policy components in the future.

This architecture contains three major components: a lightweight portal, an adaptive portal and the Apache AXIS2 Web services components. The communication between these components is via SOAP messages (WSRP producer also communicates with WSRP consumer via SOAP messages).
3.3.2 The Lightweight Portal Component

The Lightweight Portal of the architecture is the presentation layer of the architecture, which handle HTTP communication with the client’s mobile device browsers. Web portal has many advantages over other types of web application, especially while it is deployed on mobile platform.

- The technology is mature, widely deployed, and well supported. .NET Compact Framework 2.0 is one of the most well-known and widely supported frameworks for mobile platform. (Yuan and Long, 2002)
- Thin clients require little computing power and therefore can run on mobile platform. (Yuan and Long, 2002)
- One of the greatest advantages of Web Portals is that it enables “single sign-on” capability. It is even more important for mobile web application because mobile client does not need to repeat the authentication process while accessing different portlets inside portal.
- Web portal allows information from discrete sources to be displayed in a meaningful order to clients. This is essential for mobile web application because the mobile screen resolution is very limited. The portal can manage and render information from different sources to make it more relevant and valuable to user under the constraint of screen resolution.
• Web portal also enables personalization, so that mobile clients can customize the components of portal to meet their goals. This allows mobile clients to maximize the use of device resources to fit their purposes.

According to the M-A-P-E design pattern, the Lightweight Portal needs to fulfill the Monitor function of the pattern, where information about the mobile devices, network connection and other related environment conditions could be collected. The information about the environment condition, network condition and devices resources will be collected and aggregated inside the WSRP proxy component and send as request to the adaptive portal component via SOAP message. In this Lightweight Portal, the portlets are remote portlets offered by the adaptive portal component, which could be rendered using WSRP Specification.

![Figure 10 - Lightweight Portal Component Architecture](image)

Components:
• Mobile Portal: the mobile portal component is responsible for mobile web browser to connect to the system. The Mobile Portal is actually the user interface of the middleware architecture which could be displayed in an appropriate manner inside the mobile web browser. This mobile portal component is also responsible for collecting environment and device information such as the network conditions, or devices specification.
• The Network Monitor component is responsible for gathering information from the Mobile Portal and correlated the collected information into symptoms which could be analysed later.

• The Device Identification component is responsible for gathering information about Mobile Devices specification, such as screen resolution, battery life, memory capacity, and so on. The collected information will be gathered and analysed later. This component and the network monitor component together will fulfill the Monitor function in the M-A-P-E pattern.

• The WSRP Proxy Component is responsible for communication between the Lightweight Portal component and other WSRP Producer (in this architecture is the adaptive portal component). The WSRP Proxy component could be used to manage the lifecycle of the remote portlet, such as clone portlet, perform blocking interaction, get service description, get service markup, and so on.

Connectors

• Mobile Portal to Device Identification component: the Mobile Portal component connects to the Device Identification Component through a process call as these two components are located inside the same Lightweight Portal component. The information about the device specification will be sent from Mobile Portal Component to the Device Identification Component, where it is collected and classified into different categories.

• Mobile Portal to Network Monitor component: the connector between Mobile Portal and Network Monitor Component is similar to the connector between Mobile Portal and Device Identification. Instead of the information about the device specification, network conditions will be sent to the Network Monitor component, where it is gathered and classified into different symptoms.

• Mobile Portal to WSRP Proxy Component: the Mobile Portal communicates with the WSRP Proxy using the method call. The mobile portal will send the request about remote WSRP portlet to WSRP Proxy component and display the HTML markup fragment response from the WSRP proxy on the mobile client browsers.

• Device Identification to WSRP Proxy Component: The Device Identification component communicate with the WSRP Proxy component using method call.
The WSRP Proxy Component will add the device specifications from the Device Identification Component to the current request.

- Network Monitor to WSRP Proxy Component: The Network Monitor component communicate with the WSRP Proxy component using method call. The WSRP Proxy Component will attach the network conditions from the Network Monitor Component to the current request.

**External Interface:**

- HTTP interface: the HTTP interface is the entry point of the mobile portal, where the mobile portal handles HTTP request and sends HTTP response to mobile clients.
- SOAP interface: the SOAP interface is the external interface of the Lightweight Portal component, through which, the WSRP proxy will send requests for remote WSRP portlets and receive the HTML markup fragment responses from the remote WSRP producers.

### 3.3.3 Adaptive Portal Component

The Adaptive Portal component could be considered as the core component of this proposed middleware architecture. The Adaptive Portal component itself is a web portal application, which contains most of the business logic of the proposed architecture. The Adaptive Portal exposes its portlets as WSRP portlets, which in turn is rendered by the Lightweight Portal component discussed above. The Adaptive Portal Component is compliant with the standard Web Portal Architecture suggested by Apache Pluto Architecture. (Apache Pluto, 2007)
Apache Web Portal Architecture

A normal Web Portal Architecture usually includes two components: a portal web application and a portlet container.

![Figure 11 - Apache Pluto Portal Architecture](Source: Hepper and Hesmer, 2005)

The portlet container is a component in the portal architecture which is responsible for portlets invocation using Portlet API. It is possible to embed a standalone, third-party portlet container into any portal architecture as long as the architecture complies with portlet container’s requirements, such as implementing all Service Provider Interfaces (SPIs). As the standalone portlet container could be embedded into any portal architecture, this adaptive portal component will invoke the Apache Pluto Portlet Container to manage and control the lifecycle of the portlet application. The Apache Pluto Portlet Container is a simple, but fully functioned, JSR168-compliant portlet container, which is part of the Apache Pluto Portal 1.0.1. This portlet container is also embedded inside other commercial Apache Portals such as the Jetspeed portal and Apache Geronimo Project (Hepper and Hesmer, 2005).

The architecture of web portal application, however, is not defined by this portal architecture. The web portal application can access to the portlet container using Portlet Container API. The web portal application uses this Portlet Container API to manage and control the Portlet Container lifecycle (such as init, destroy) and via the
portlet container, it manages and controls the lifecycle of each individual portlet application. The functionalities of the web portal application are up to developer to decide. For example, the Apache Pluto Portal is just a reference implementation of portal server, which purposes are to demonstrate how the portlet API works and provides developers with a JSR-168 compliant portal in which they can test and deploy their portlets. Apache Jetspeed, on the other hand, focuses more on the portal itself, rather than the portlet container (Hepper and Hesmer, 2005). The Apache Jetspeed reuses the Pluto Portlet Container, but it is developed as a commercial web portal, rather than a reference implementation (RI) as the Apache Pluto Portal.

**Adaptive Portal Component architecture**

Based on the standard portal architecture, this adaptive portal component could also be divided into two subcomponents: a web portal application component and a portlet container component. Although the Portlet Container component is reused from the Apache Pluto Portlet Container, the web portal application component has to enable adaptive components inside its architecture. According to the M-A-P-E pattern and the Policy Point pattern, the web portal component has to fulfill *Analyse* function and *Plan* function of the pattern. An adaptive component is added to the normal architecture, which will gather the required information and make the decision on whether some actions are required to adapt to the current environment conditions.
Components:

- **WSRP4J Proxy**: this component is responsible for extracting information from the incoming SOAP message (i.e. from the Lightweight Portal component above). The information could be separated into two parts, a normal remote portlet request and the information about the environment condition.

- **Adaptive Analyser Component**: this component is responsible for extracting environment symptoms and device specifications from the incoming SOAP request. These symptoms and device specification will be analysed to decide whether changes are required to adapt to the environment and device conditions. This component can fulfill the *Analyse* and *Plan* function in the M-A-P-E pattern.

- **Web Portal Application**: this component could be a normal web portal application, which can manage, control and access the Apache Pluto Portlet
Enabling mobile access to adaptive services and portal

Container using the Portlet Container Invoker APIs. Moreover, if the Adaptive Analyser Component decides that changes are required to adapt to the environment conditions, the Web Portal Application may also invoke the adaptive components to fulfill the Execute function in the M-A-P-E pattern. This could be further discussed later.

- Portlet Container: this component is responsible for handling, managing and invoking the portlet applications. There are two types of portlet applications: local portlets and remote portlets.

- Local Portlet Applications: These portlets are developed locally, which may invoke the web services from Apache AXIS2 Web Service Component via the SOAP interface.

- Remote Portlet Application: these portlets are the remote portlets, which are offered by the third-party WSRP producer. The Portlet Container cannot render these portlets directly. The WSRP4J Consumer Component is required.

- WSRP4J Consumer Component acts as the proxy so that the Portlet Container can invoke the remote portlets as local portlets.

Connectors:

- WSRP4J Proxy to Web Portal Application: the WSRP4J Proxy will communicate with the Web Portal Application using method call. The requested portlet will be extracted from the incoming SOAP message and pass through the Web Portal Application.

- Web Portal Application to Portlet Container: the web portal application component communicates with the Portlet Container via the method call (i.e. using Portlet Invoker API).

- WSRP4J Proxy to Adaptive Analyser Component: this connector is similar to the connector between WSRP4J and Web Portal Application, but in this connector, the information about environment conditions will be passed to the Adaptive Analyser Component.

- Adaptive Analyser to Web Portal Application: The decision made from the Adaptive Analyser will be passed to the Web Portal Application.

- Portlet Container to Local Portlet: The Portlet Container will render the local portlet using method call (i.e. Portlet API).
Enabling mobile access to adaptive services and portal

- Portlet Container to WSRP4J consumer: The Portlet Container will communicate with the WSRP4J consumer using local method call.
- WSRP4J consumer to Remote Portlet Application: the WSRP4J consumer will render the remote portlet application using network call. This could be done using SOAP message.

**External Interface:**

- SOAP interface: the external interface is the entrance point of the adaptive portal component, where the incoming SOAP message will be handled by the WSRP4J Proxy before processed by the adaptive portal component. This is where the response HTML fragment will be sent to the WSRP consumer.

As can be seen from the Adaptive Portal component architecture, Portlet Container could be able to render not only local portlets but also third-party remote portlets. Since it is not possible to adaptively modify the remote portlets from WSRP producer; these remote portlets require a different way to be rendered adaptively. The invocation of adaptive components has to be embedded inside the Web Portal Component. The Apache Jetspeed 2 portal has suggested an architecture for Web Portal Application, in which the adaptive components could be invoked transparently.

**Figure 13 - Apache Jetspeed2 Pipeline**

(Source: Apache Jetspeed2, 2007)
According to this architecture, the Request from the user will be passed through a Jetspeed2 Engine, which includes a pipeline before it reaches the Portlet Container. This pipeline contains a number of valves which perform various operations on the incoming request. These valves could be divided into different categories such as Capability valve, Container Valves, and so on. The adaptive components could be implemented as user-defined valves and included into the Engine pipeline to intercept the request/response from the Portlet Container. These adaptive components could be implemented to fulfill the *Execute* function of the M-A-P-E architecture.

![Figure 14 - Apache Jetspeed2 Valves](Source: Apache Jetspeed2, 2007)

### 3.3.4 Apache Axis2 Web Services

The third component of the adaptive middleware architecture is the Apache Axis2 Web Services component. The Apache Axis2 Web Services component could be considered as the wrapper component of the back-end third-party web services, which could be considered as the business logic the local portlet applications. The third-party web services such as Amazon Web Services will be invoked by this component to fit the purposes of the system. Without the adaptive components, this web service component will just increase the complexity of the architecture by adding another layer to the proposed architecture since the local portlet applications can invoke the
Enabling mobile access to adaptive services and portal

third-party web services directly. However, in order to enhance the adaptiveness of the system, this Axis2 web services component is required.

According to the M-A-P-E pattern, the adaptive components in this web services component will be responsible for *Execute* function in the M-A-P-E pattern. The actions planned in Adaptive Analyser Component will be performed in these adaptive components.

![Figure 15 - Apache Axis2 SOAP Processing Model](Source: Apache Axis2, 2007)

In order to maximize the dynamism of the adaptive components, these components should not be implemented as part of the web services, but as a part of the Apache Axis2 Web Service Architecture. The Apache Axis2 Architecture has defined two basic actions that the SOAP processor model should be able to perform, including sending and receiving SOAP messages. Two pipes to handle the SOAP messages has been defined to perform these functions, including *In Pipe* and *Out Pipe* and the complex Message Exchange Patterns (MEPs) has been constructed to combines these two pipes. The incoming SOAP messages and outgoing SOAP messages have to pass through these two pipes and the MEPs in order to reach either requester side or service provider side.

The invocation of adaptive components could be done by intercepting the incoming and outgoing SOAP messages using the user-defined interceptors embedded inside the handler chain. When a SOAP message is being processed the handlers that are registered would be executed. The handlers can be registered in global, service, or
operation scopes and the final handler chain is calculated combining the handlers from all the scopes.

![Adaptive Web Service Architecture](image)

**Figure 16 - Adaptive Web Service Architecture**

### 3.4 Use case Diagram

As mentioned before, the adaptive middleware architecture makes a two-fold contribution that enables mobile access to either third-party web services, or third-party remote portlets. The following use cases will show the invocation of these web services and portlets from mobile devices.

#### 3.4.1 Use case 1:

The HTTP request comes from mobile devices will be handled by the Mobile Portal Component in the Lightweight Portal component. The Mobile Portal component will request the HTML markup fragment of the remote portlet from the adaptive portal component using WSRP Proxy component. Along with the request, the devices specifications, network conditions and other environment condition may be captured and attached to the WSRP request.
3.4.2 Use case 2

Adaptive portal component will handle the incoming SOAP messages using WSRP4J Proxy and invoke the corresponding local portlet to fulfill the incoming request. Along with the portlet request, the WSRP4J proxy also extracts information about the environment condition and passes it to the Adaptive Analyser component to determine the need for adaptive changes to adapt to current environment condition. The local portlet will engage the corresponding web services in the Apache AXIS2 Web Services component using SOAP request. The Apache AXIS2 Web Service will send the response SOAP message back to local portlet application in the adaptive portal component. The adaptive handler inside the Apache AXIS2 Web Services Framework will adaptively modify the incoming and outgoing SOAP messages based on the decision made by the Adaptive Analyser Component.
Enabling mobile access to adaptive services and portal

Diagram of the communication flow between the Adaptive Portal Component, SOAP Proxy, Web Portal Application, Portlet Container, Local Portlet Application, and the Inflow/Outflow SOAP Messages. The process involves handling messages from the Transport Layer, AXIIS2 Handle, and Message Receive, leading to a response from the Apache AXIS2 Web Services.
Figure 18 - Adaptive Portal Component handles incoming request
3.4.3 Use case 3

The HTML markup fragment from the Adaptive Portal then will be returned to Mobile Portal component of the Lightweight Portal using the WSRP Proxy and display in the mobile client browser.

Figure 19 - Lightweight Portal Component displays HTML markup response
3.4.4 Use case 4:
The WSRP4J Proxy in the adaptive portal will handle the incoming SOAP messages. The Web Portal Application identifies that the requested portlet is a WSRP Portlet, and requests the HTML markup fragment from third-party WSRP producer. The environment condition from the WSRP4J Proxy will be used by the Adaptive Analyser Component to decide whether adaptive changes are required. If the changes are required, this request may be intercepted, modified inside the Web Portal Application to adapt to the environment condition.
Figure 20 - Adaptive Portal Component handles request for remote portlet.
CHAPTER 4:

Middleware Architecture Evaluation Method

About this chapter

This chapter provides an overview of the method for evaluating middleware architecture quality attribute – the Method for Evaluating Middleware architectureS (MEMS).
4 Middleware Architecture Evaluation Method

4.1 Chapter overview:

This chapter is divided into the following sections:

Overview of MEMS: provides an overview of a Method for Evaluating Middleware architecture.

4.2 Overview of MEMS

Currently, most architecture evaluation methods (such as Software Architecture Analysis Method (SAAM) or Architecture Trade-off Analysis Method (ATAM)) target higher level architecture issues rather than the middleware. As the nature of Middleware architecture is different from other architectures, a method that could specifically be used to evaluate middleware architecture is required. For example: middleware technologies are horizontal in nature, providing mechanism for a wide range of application in many vertical application domains. Therefore, the requirements of the middleware-based application focus more on the domain-specific application behaviours, rather than the middleware behaviours. This indicates that the evaluation methods for middleware architecture driven by the concerns of individual quality attributes rather than the domain-specific application requirements should be required.

Method for Evaluating Middleware architecture (MEMS) (Liu et al., 2006) is a scenario-based evaluation approach with clearly defined set of steps that could be used to evaluate the suitability of middleware architecture candidates for an application’s quality goals. MEMS leverages generic quantitative and qualitative evaluation techniques by prototyping, testing, rating and analysis. The output of MEMS represents the rating of each architecture candidate against the quality attributes of interest. The MEMS defines the evaluation process into 7 steps.
Step 1: Determine quality attributes

The first step is identifying the quality attributes of interest that the middleware architecture candidates need to satisfy. These quality attributes are not driven by domain-specific application requirements, but they are general attributes, such as performance, security, maintainability and so on. One quality attribute may embody many specific concerns. For example, the performance quality attribute can be considered from few different points, such as response time, round-trip time, resource usage, and so on. The purpose of defining these quality attributes is to set a context, in which different architecture candidates will be evaluated.

Step 2: Generate Key Scenarios:

The quality attributes defined in the first step is too abstract for analysis. The scenarios are created as a descriptive mean to capture concrete quality attribute requirements.

Step 3: Define Quality Attribute Scales:

There are two different method in evaluating quality attributes, including quantitative method and qualitative method. Quantitative method evaluates attributes based on
Enabling mobile access to adaptive services and portal

measurement, analytical modelling and other simulation techniques. In qualitative method, the weighted scoring method (WSM) could be used to evaluate quality attributes. The WSM method requires a clear, unambiguous definition of rating scale, so that the evaluator can compare weight and scores to qualitative attributes of different architecture candidates. This step defines consistent rating criteria for evaluating quality attributes of different architecture candidates.

**Step 4: Determine Architecture Alternatives**
This step determines the alternative middleware architectures possible evaluation. Different middleware architecture candidates could perform and support exactly the same functionality that described in the scenarios. Hence, these candidates will be evaluated based on the quality attributes.

**Step 5: Prototype**
The prototype implementation could be implemented. The quantitative measurements would be conducted for quantitative attributes of interest. Prototyping is also useful in obtaining feedback on architecture design and understanding how it may impact other qualitative attributes.

**Step 6: Evaluate Quality Attribute**
Evaluation for each quality attributes could be conducted. Both qualitative and quantitative attributes of architecture candidates will be evaluated in this step based on the scales in Step 3.

**Step 7: Present Evaluation Results**
The results from the evaluation step will be collected, correlated and analysed in this step. The evaluation results could be visually presented in a way that clearly identifies the rating of each architecture candidate regarding to individual quality attributes.
CHAPTER 5:

Case Study: Enable Mobile Access to Amazon Web Services

About this chapter
A case study of the proposed architecture should be evaluated based on the MEMS process. The major quality attributes that the case study wants to evaluate include: performance, usability, maintainability, portability, programmability and dynamism of adaptive components.
5 Case Study

5.1 Chapter overview:
This chapter is divided into the following sessions:

**Overview of the Case Study:** provide an overview of the case study: Evaluating Middleware Architecture for Enabling Adaptive Mobile Access to Amazon Web Services

**Step 1: Determine quality attributes:** identifies the list of quality attributes of interest, in which the architecture could be evaluated.

**Step 2: Generate key scenarios for each quality attributes:** generates scenarios that capture the quality attributes listed in step 1.

**Step 3: Define rating scale:** provides a reasonable, concrete rating scale that could be used to evaluate the quality attributes of architecture candidates.

**Step 4: Determine architecture candidates:** discuss the possible architecture candidates.

**Step 5: Prototype:** implements the architecture candidate prototype, which could be used to perform the quantitative evaluation.

**Step 6: Evaluate quality attributes:** evaluate the qualitative and quantitative attributes of the architecture candidates.

**Step 7: Present the results:** display the result from evaluation step.

5.2 Overview of Case Study
The chapter 3 has provided an adaptive middleware architecture, which supports the interoperability of Web Services and Interactive Portal on mobile platform. The architecture makes a two-fold contribution that enable mobile devices access to either third-party web services or third-party WSRP remote portlets. However, this case study will only focus on the first contribution of the architecture that allows mobile devices access to third-party web services, and in this case study, Amazon Web Services. The invocation of third-party portlet on mobile devices is out of scope of this case study.
The Amazon Web Services are widely used as third-party web services because it provides a direct access to Amazon’s robust technology platform. This case study will focus on how Amazon Web Services can be invoked from mobile device platform. In order to develop a prototype of the proposed architecture, the scope of this case study could narrow down to how the middleware architecture can adaptively resize the image returned from Amazon Web Services. Even the resizing image is such a preliminary task; it is still suitable for demonstrating the effectiveness of the architecture in enabling mobile access to the web services.

Currently, in order to support the interoperability of web services in different platform, the Amazon Web Services provides images in three different sizes: large, medium and small image. It is the responsibility of web services clients to choose which image is the most suitable for their purposes. While Amazon Web Services is invoked from mobile devices, there are two different approaches that could be applied. The first approach is that the Amazon Web Services always return a small image, which might be displayed on mobile devices. However, this approach has some limitations. For example: it is not guaranteed that the small image could be display correctly on mobile devices (i.e. the small image is not “small” enough to fit the purposes of the mobile application). Moreover, when the mobile resources are available, a better resolution of the image should be used. The used of worse-case scenario (i.e. fixed scale small image) should not be sufficient enough. Because of these limitations, the second approach could be considered. The Amazon Web Services will always return large, high resolution images and the middleware will analyse the availability of mobile resources to resize the image accordingly.

5.3 Step 1: Determine quality attributes

In this case study, the quality attributes are driven by the general scenario described above. The quality attributes of interest includes: usability, performance, programmability, reusability, maintainability and dynamism of the adaptive components in the architecture candidates. The following section will describe the specific concerns of each quality attribute and relevant techniques that could be used to evaluate these quality attributes.
Enabling mobile access to adaptive services and portal

**Performance** the middleware architecture is an adaptive, self-managed architecture which enable mobile access to third-party web services. The different between this architecture and other architecture is the introducing of adaptive components. The performance quality attribute therefore could be evaluated by measuring the impacts of adaptive components to the architecture, such as the overhead of the adaptive components.

**Usability** concerns with the ease of use of architecture candidate. Because the mobile platform is a resource-limited platform, then the usability is one of the key attributes that the architecture has to fulfill. The usability features may include aggregating data, aggregating commands, using application concurrently, single sing-on activities, navigating within a single view, and so on (Bass et al., 2005). This case study will focus on three of these usability features, including aggregating, navigating data within a single view, supporting single sign-on feature and supporting customization and personalization.

**Programmability** in this evaluation means the effort required to implement and test the middleware architecture prototype. A more complex architecture, whose components are implemented using different APIs and Framework, will require more efforts in both developing and testing process. The Programmability could also be evaluated based on the degree to which testing mechanisms are supported.

**Portability** indicates the ease with which the components or services of this architecture could be used in different platforms. In the adaptive architecture, the portability of adaptive components is one of the key quality attributes. The adaptive components of this architecture should be portable to other platform without incurring programming efforts. For example: in this case study, the component that is responsible for resizing images, could be reused by another architecture. The portability evaluates how easy it is to plug-and-play this component in those architectures. Another reusability feature that is worth to mention is the interoperability of the adaptive services provided by this architecture. The portability indicates the ease of invoking these adaptive services in other framework as loosely couple remote services.

**Maintainability** is the ease with which the system can have defects corrected or new elements added. The maintainability attribute in this case study could be evaluated
by the ease in which new adaptive components and adaptive services could be added to current system.

**Dynamism of the adaptive components** concerns with how the invocation of adaptive components in the architecture. As discussed in chapter 2, the dynamism of adaptive components could be evaluated based on how, where and when the adaptive components are invoked (McKinley et al., 2004). Therefore, this case study will evaluate the dynamism of the adaptive components based on this taxonomy. However, there is an overlapping between how and where the adaptive components are invoked. Under the scope of middleware architecture, the degree to which adaptive component is transparent to other architecture components should also indicate where these adaptive components are invoked in the development process. Therefore, the criteria of evaluating the dynamism of adaptive components could be limited to how and when these components are invoked inside the architecture.

In this case study, performance, usability, programmability, portability, maintainability and dynamism of adaptive components have created a context in which the architecture will be evaluated using MEMS. The evaluation of these attributes can either use qualitative or quantitative method. For the qualitative methods, expert’s opinion will be used to evaluate these attributes. The experts rate each quality attributes based on the pre-defined rating scale that will be discussed on section 5.5 – Step 3: define rating scale

5.4 **Step 2: Generate key scenarios for each quality attributes**

The purpose of generating the key scenarios for each quality attributes is to refine the context of these attributes, in which the evaluation method could be performed. The description of the quality attributes discussed in section 5.4 is too abstract to be assessed. A more detail description should be required, and it could be captured by using the set of scenarios.

- **Performance** quality attributes of the architecture could be evaluated based on *overhead generated by the adaptive components.*
Enabling mobile access to adaptive services and portal

- **Overhead of adaptive component**: the overhead of adaptive component under different stressful conditions could be measured using a dummy adaptive component (i.e. the component that intercept the incoming request and outgoing request, but does not modify it in anyway.). Firstly, this dummy adaptive component will be removed from the middleware architecture and the round-trip-time could be calculated under different stressful conditions (i.e. different number of simultaneous clients connects to the middleware at the same time). After that, the “do-nothing” adaptive component will be added to the middleware architecture, and the roundtrip time could be measured again. The difference between these roundtrip times is the adaptive component overhead.

- **Usability**: The usability features which could be evaluated in this case study include: aggregating and navigating data in a single view, supporting single sign-on facility and supporting customization and personalization.
  - **Aggregating and navigating data in a single view**: the mobile client could connect to different information from different sources in a single view.
  - **Supporting single sign-on facility**: The mobile client could login once and access to multiple applications from different sources.
  - **Supporting customization and personalization**: The mobile client could modify and personalize the view of the applications.

- **Programmability**: the programmability could be divided into two sub-categories including how much effort required for implementing the middleware architectures and how many mechanisms supported for testing purposes. This could be done based on the developer experience. No scenario is required.

- **Portability**: this quality attribute could be evaluated by the degree to which the middleware architecture components could be portable to other platform. It also be measured by how ease it is to invoke the adaptive services provided by this architecture. No scenario is required for this quality attribute.
• **Maintainability:** This quality attributed is evaluated based on how new components could be added into this architecture. The new components could either be new services that the middleware provides or the new adaptive component that middleware could invokes.
  - Adding new services: A new service will be added to the architecture, and it will be invoked from the mobile client. The maintainability could be measured by whether the system needs to recompile, reconfigure, restart, or do nothing in order to invoke this service.
  - Adding new adaptive component: the scenario is similar to the previous scenario, but instead of new services, new adaptive components will be added to the current architecture.

• **Dynamism of the adaptive components:** The dynamism of adaptive components is measured by when and how these components are invoked inside the architecture. There is no particular scenario required to evaluate this quality attribute. This can be done using qualitative method.

Based on these key scenarios, the middleware architecture could be evaluated using either qualitative method (i.e. reasoning about the quality attributes, based on expert’s opinions) or quantitative method (i.e. measuring the quality attributes based on the implemented prototype). The results from this method will be evaluated based on a rating scale, which is defined in section 5.5.
5.5 **Step 3: Define rating scale**

In this case study, a coarse grained scale could be used to evaluate the architecture candidates, which is defined as High (H), Middle (M) or Low (L). An ordinal scale should not be used in this context (i.e. from 1 to 5) because the values do not truly represent the differences between scales in neither ratio nor distances. In fact, the differences in their values only give an indication of their relative rankings. A more fine grained could be used, however it will requires more efforts to define and perform; and under the scope of a case study, it is not worthy to define such a scale.

**Table 1 - Quality Attributes Scale Definition**

<table>
<thead>
<tr>
<th>Quality Attributes</th>
<th>Scale Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td></td>
</tr>
<tr>
<td>Adaptive Component Overhead</td>
<td>Adaptive Component Overhead &lt; 5% while handle 20 concurrent requests at the same time.</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td></td>
</tr>
<tr>
<td>Aggregating in one view (Bass et al., 2005)</td>
<td>The information from different sources aggregates in one page view in a meaningful way to client.</td>
</tr>
<tr>
<td>Supporting single sign-on facility (Bass et al., 2005)</td>
<td>Client can login once and can access to multiple pages.</td>
</tr>
<tr>
<td>Supporting</td>
<td>Client can</td>
</tr>
<tr>
<td></td>
<td>personalization and customization (Bass et al., 2005)</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td><strong>Programmability</strong></td>
<td>Effort required for implementing architecture.</td>
</tr>
<tr>
<td><strong>Support of testing mechanism</strong></td>
<td>Testing could be done using Debugging IDE (i.e. Visual Studio 2005 or Eclipse IDE).</td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td>Adaptive Component</td>
</tr>
</tbody>
</table>
### Enabling mobile access to adaptive services and portal

<table>
<thead>
<tr>
<th>Maintainability</th>
<th>Services</th>
<th>plugged and configured to be used in other platform.</th>
<th>to be recompiled in order to be invoked in other platform.</th>
<th>modification of the current code to be invoked in other platform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new services</td>
<td>New services could be added to the system without restarting the system.</td>
<td>New services could be added to the system but require configuration and restarting the system.</td>
<td>New service could be added to the system by recompiling the system.</td>
<td></td>
</tr>
<tr>
<td>Add new adaptive component</td>
<td>New component could be added to the system without restarting the system.</td>
<td>New component could be added to the system but require configuration and restarting the system.</td>
<td>New component could be added to the system by recompiling the system.</td>
<td></td>
</tr>
</tbody>
</table>

### Dynamism of adaptive component

<table>
<thead>
<tr>
<th>How adaptive component is invoked</th>
<th>The middleware can use configuration method to invoke adaptive component.</th>
<th>The middleware can use the API to invoke adaptive components (i.e. to manage, control the lifecycle).</th>
<th>The adaptive components are embedded into middleware source code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>When adaptive component is invoked</td>
<td>Adaptive component is invoked at runtime.</td>
<td>Adaptive component is invoked at load time.</td>
<td>Adaptive component is invoked at build time.</td>
</tr>
</tbody>
</table>

Because each of the quality attributes has sub-categories, the rating of these quality attributes is evaluated based on their sub-categories. In order to aggregate the overall rating of these attributes from the rating given to each sub-categories, each sub-categories
Enabling mobile access to adaptive services and portal

has to be weighted. The value of the weight represents how much sub-categories contributes to the overall quality attributes. For example: dynamism of adaptive components has 2 sub-categories, and then each sub-category can be weighted by 0.5, which means that each sub-category is contributed 50% to the rate of dynamism of adaptive components attribute. The scale definition as High, Middle and Low must be assigned some values and the overall rating can be calculated. Finally, the overall rating will be converted back to the descriptive form such as High, Middle or Low.

1. Assign values to High, Middle and Low to convert the rating of each sub-category into values correspondingly.
   \[ H \leftarrow 1, M \leftarrow 2/3, L \leftarrow 1/3 \]

2. Calculate the overall rating based on weight and the converted rating of each sub-category
\[
Rate = \sum_{i=1}^{n} \text{Weight}_n \times \text{Rate}_n \quad (\sum_{i=1}^{n} \text{Weight}_n = 1)
\]
where \( n \) is the number of sub-categories

3. Convert back the value of the overall rating into the descriptive form.
\[
Rate \leftarrow \left\{ \begin{array}{l}
H \quad (2/3 < Rate \leq 1) \\
M \quad (1/3 < Rate \leq 2/3) \\
L \quad (0 < Rate \leq 1/3)
\end{array} \right.
\]

Figure 22 - Rating Calculation with Sub-Categories  
(Source Liu et al., 2006)

5.6 Step 4: Determine architecture candidates

The purpose of this case study is to evaluate the quality attributes of the proposed architecture in section 3. Therefore this architecture is an architecture candidate of this evaluation method. Some other architecture candidates might be considered, such as the traditional approaches discussed in section 2.5, but these architecture candidates are not under scope of this case study. This case study will focus on evaluating adaptive middleware architecture in section 3 only. The comparison between this proposed architecture and other architecture candidates could be left for the future development.
The overall architecture discussed in section 3 contains three layer components including a lightweight portal, an adaptive portal component and an adaptive web services component. In order to simplify the complexity of the architecture, it is assumed that adaptive actions should always be required to adapt to the current environment conditions (i.e. the Adaptive Analyser Component in section 3.3.3 always decides that the adaptive actions should be required). In other word, the image should always be resized adaptively based on the screen resolution of the mobile clients. In the future prototype, the adaptive portal should be refined to make decision based on analysing incoming information from lightweight portal and planning the adaptive actions to cope with the current environment conditions.

5.7 Step 5: Prototype

The prototype of architecture candidate in section 5.6 should be implemented in this step. When implemented, the three major components from the proposed architecture candidates could be deployed in different computation nodes. The deployment architecture of this candidate can be found in Figure 23.
The lightweight portal component is developed using ASP.NET 2.0. The main reason of using this ASP.NET framework is that it provides a high level support in implementing a mobile web application. With the supports of Microsoft Visual Studio 2005, it significantly reduces the efforts required to develop a web application for mobile platform (i.e. the Mobile Portal component in figure 10, section 3.2). Moreover, the ASP.NET 2.0 framework also supports mechanisms to capture the mobile client specifications and the network conditions. The only concern of this ASP.NET framework is that it does not explicitly support WSRP Specification. However, Nolan (2006) has provided a WSRP wrap-up component, which could be reused to implement the WSRP proxy component in the lightweight portal component (figure 10, section 3.2). The whole lightweight portal component could be deployed in the Microsoft Internet Information Services (IIS) container and could be accessed from the mobile client as a normal web application.
The adaptive portal component is the Apache Pluto Portal 1.0.1 which runs in the Tomcat 5.5 container with WSRP4J plug-in, which enable local portlets be invoked remotely using WSRP specification. The reasons of using Apache Pluto Portal 1.0.1 is that as discussed in section 5.6, the Adaptive Analyser Component always decides that action should be taken by the adaptive component to adapt to the current environment (i.e. the mobile devices screen resolution). The image should be resized based on the screen resolution adaptively. Therefore a more complicated portal component is not required at this prototype. The devices specifications will be passed directly to the adaptive component in order to resize the images.
The final component is the Apache Axis2 Web Services, with adaptive components embedded inside the Axis2 handler chain. The Axis2 Web services use Apache Tomcat 5.5 Server as its container. Axis2 Web Service is implemented as a wrap-up component of the Amazon web services, which captures the images return from Amazon Web Services, attaches the image object as a SOAP attachment (using SOAP with Attachment Specification (Gunarathne, 2007)). The adaptive components which could intercept the incoming SOAP messages and outgoing SOAP messages can access to and modify the attachment image according to the mobile resolution (which is sent from lightweight portal to adaptive portal component).
In order to measure the overhead of adaptive component costs to the whole architecture, Apache JMeter has been used as a testing tool. Apache JMeter is a 100% pure Java desktop application designed to load test functional behaviours and measure performance. Apache JMeter could generate a number of concurrent requests to the middleware application and measure the response time from the application. To measure the overhead of adaptive component, the first step is to measure the round-trip time by sending multiple concurrent requests from JMeter to the Apache Axis2 Web Service which does not include the adaptive component and measure the response time. The next step is implementing a dummy adaptive component as a user-defined handler (i.e. which do nothing, just capture the SOAP message and return). This handler will be included into the handler chain. Finally, the similar process as in first step will be conducted to measure the roundtrip time. The difference between these two round-trip times is the overhead of the adaptive components. This test will be conducted in different stressful conditions (i.e. different numbers of concurrent clients access to the services at the same time).

The test bed is setup in a lab environment. The machine hosting the web services is a Dell workstation, with 3 GHz Xeon Processor and 2GB RAM. The client that runs Apache
JMeter is a Dell Inspiron 6400 with Pentium Core 2 Dual T7200 and 1GB RAM. These client and server are connected using the Local Area Network at 100 Mbps.

5.8 Step 6: Evaluate quality attributes

The following section will evaluate the prototype of the architecture candidate discussed in section 5.7, based on the key scenarios (section 5.4) and the scale definition (section 5.5). As discussed in section 5.2, the quality attributes of the proposed architecture candidate and its prototype will be evaluated, including: usability, performance, programmability, maintainability, portability and the dynamism of adaptive component. Both qualitative method and quantitative method could be used to evaluate these quality attributes.

5.8.1 Usability

The usability of the architecture candidate could be evaluated based on how the architecture support single sign-on facility, how information from different sources could be aggregated in to one view and how the user clients can customize the view based on their interests.

Aggregating information from different source in one single view: The architecture has implemented a lightweight portal, which is designed specially for mobile device platform. The basic idea of portal is a place that brings together content from disparate sources, displaying results through a single interface (Microsoft Technical Professionals, 2007). Therefore, the use of portal in this middleware architecture could enhance the usability of the architecture. Therefore based on the scale definition in section 5.5, this sub-category could be rated as High (H).

Supporting single sing-on facility: The lightweight portal renders WSRP portlets from the adaptive portal component. Therefore, it is based on the adaptive portal component and WSRP specification to decide whether single sign-on facility is supported. In this prototype, the reference implementation of adaptive portal component (Apache Pluto Portal 1.0.1) does not require authentication; hence the single sign-on is not supported. However, the fully function portals (such as Apache Jetspeed2) do require authentication and support this single sign-on facility. WSRP Specification also supports
Enabling mobile access to adaptive services and portal

authentication and security by providing mechanism to handle security and authentication (i.e. WSRPRegistrationService is one of 4 services provided by WSRP Specification). Therefore, the single sign-on facility is supported in this architecture.

Supporting personalization and customization: the use of portlet and portal application can guarantee the personalization and customization of the architecture. Portlet that is compliant to Java Portlet Specification JSR-168 usually support 3 modes, including maximize, minimize and edit mode. The use of these portlet modes and the state persistence (from WsrpService.cs inside Wsrp Proxy Component – Figure 24) enables users customize the portal based on their interests.

Based on the above discussion, rating for each sub-category of usability quality attribute are:

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregating information in one view</td>
<td>High</td>
</tr>
<tr>
<td>Supporting single sign-on facility</td>
<td>High</td>
</tr>
<tr>
<td>Supporting personalization/customization</td>
<td>High</td>
</tr>
<tr>
<td>Overall Usability attribute</td>
<td>(1/3 * 1) + (1/3 * 1) + (1/3 * 1) = 1 → High</td>
</tr>
</tbody>
</table>

5.8.2 Performance

The performance of the system could be evaluated based on overhead of the adaptive components incurred to the architecture.

Adaptive component overhead: the adaptive component overhead could be calculated using the method discussed in section 5.7. The measurement process was conducted for different numbers of simultaneous client access to the Axis2 web services component at the same time. The numbers of simultaneous client are 1, 10, 20, 30, 40 and 50 accordingly.
Enabling mobile access to adaptive services and portal

Summary of Performance Result

As can be seen from the graph, the overhead of the adaptive component is insignificant unless more than 40 threads request the web services simultaneously.

Table 2 - Summary of Component Overhead under Different Stressful Conditions

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Response time without handler</th>
<th>Response time with handler</th>
<th>Adaptive component overhead</th>
<th>Percentage of overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 threads</td>
<td>218ms</td>
<td>208ms</td>
<td>10ms</td>
<td>4.6%</td>
</tr>
<tr>
<td>10 threads</td>
<td>1090ms</td>
<td>1093ms</td>
<td>3ms</td>
<td>0.3%</td>
</tr>
<tr>
<td>20 threads</td>
<td>1874ms</td>
<td>1912ms</td>
<td>38ms</td>
<td>2%</td>
</tr>
<tr>
<td>30 threads</td>
<td>3027ms</td>
<td>3046ms</td>
<td>19ms</td>
<td>0.6%</td>
</tr>
<tr>
<td>40 threads</td>
<td>4933ms</td>
<td>5277ms</td>
<td>344ms</td>
<td>7%</td>
</tr>
<tr>
<td>50 threads</td>
<td>8563ms</td>
<td>9439ms</td>
<td>876ms</td>
<td>10,2 %</td>
</tr>
</tbody>
</table>
Enabling mobile access to adaptive services and portal

Even while handling 50 concurrent threads at the same time, the overhead of the adaptive component incurred to the Apache Axis2 Web Services is still less than 10%. This indicates that the overhead of adaptive component is very favourable.

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead of adaptive component</td>
<td>High</td>
</tr>
<tr>
<td>Overall Performance attribute</td>
<td>High</td>
</tr>
</tbody>
</table>

5.8.3 Dynamism of adaptive components

The dynamism of adaptive components could be evaluated by how, where and when the adaptive component is invoked in the proposed architecture.

How adaptive component is invoked: the adaptive components that capture the devices specification such as screen resolution is embedded inside the lightweight portal. The adaptive component that intercepts the requests/responses is embedded inside the Axis2 handler chain as an adaptive handler. This handler is part of the Axis2 Web Service framework; therefore it does not belong to any particular component of the architecture. In other word, the adaptive component is transparent to other component of the architecture. The middleware can invoke the adaptive component using configuration method.

When adaptive component is invoked: the adaptive component is embedded inside the Axis2 Web Services; therefore it would not be called, unless the web services are requested. The adaptive components will tune the requests and responses of the web services at run time. As can be seen from Figure 1, section 2.2.4, the adaptive component is a tunable component. This can enhance the dynamism of the adaptive components.
Enabling mobile access to adaptive services and portal

Based on the above discussion, rating for each sub-category of dynamism of adaptive components are:

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>How adaptive component is invoked</td>
<td>High</td>
</tr>
<tr>
<td>When adaptive component is invoked</td>
<td>High</td>
</tr>
<tr>
<td>Dynamism of adaptive component</td>
<td>((1/2 \times 1) + (1/2 \times 1) = 1 \rightarrow \text{High})</td>
</tr>
</tbody>
</table>

5.8.4 Portability

Portability quality attributes could be measured by how easy it is for adaptive components and middleware services to be used on different platform.

Portability of Adaptive Component: The adaptive component that fulfills the Execute function in M-A-P-E pattern is implemented as a handler inside the Axis2 Web Services. In this approach, the adaptive components are part of the execution framework (i.e. the Axis2 Web Service framework). These adaptive components have to use the Axis2 APIs in order to intercept the incoming and outgoing SOAP messages. This will somehow limit the portability of these adaptive components on other frameworks. The adaptive handler is specifically designed for the AXIS2 Web Service; therefore it will implement the AXIS2 APIs. Even the business logic behind the adaptive components could be reused; it still requires modification of code in order to be invoked by other frameworks (i.e. using different APIs).

Portability of Adaptive Services: on the other hand, this middleware architecture enhances the portability of its services on different platform. In this case study, the adaptive services could either be the Axis2 Web Services or the WSRP Portlet, which could be consumed, rendered as remote components without incurring programming overhead. The lightweight portal in this case study is an example. From the adaptive portal component point of view, the lightweight portal is not a component of the architecture, but it is a WSRP consumer, which rendered the WSRP portlet from the adaptive portal component and embedded it inside the lightweight portal. Similarly to the WSRP portlet, the lightweight portal can also reuse the adaptive Axis2 Web Services directly (i.e. which has invoked the adaptive handlers) as a normal web services.
However, the WSRP approach is far more preferable, due to the advantages of WSRP Specification (section 2.4.3).

Based on the above discussion, rating for each sub-category of usability quality attributes is:

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability adaptive component</td>
<td>Low</td>
</tr>
<tr>
<td>Portability Adaptive Services</td>
<td>High</td>
</tr>
<tr>
<td>Portability quality attribute of the architecture</td>
<td>$(1/2 * 1) + (1/2 * 1/3) = 2/3$ $\rightarrow$ Middle</td>
</tr>
</tbody>
</table>

### 5.8.5 Maintainability

The maintainability of the architecture is evaluated by how easy it is to add new services or components to the existing architecture.

Add New Services: In order to add a new service to the existing system, an Axis2 web service and a Pluto portlet application have to be implemented. The Axis2 Web Service will be a wrapper for the external third-party web services (i.e. Amazon Web Services) and the Pluto portlet application will invoke the Axis2 Web Services and display it in a meaningful way. Therefore, deploying a new service to mobile devices will include deploying an Axis2 Web Service and a Pluto portlet application. Deploying both Axis2 Web service on Axis2 Framework and Pluto portlet application on Pluto Portal 1.0.1 does not require configuration or modification of the current system. It is possible to deploy the new services on middleware system dynamically during run-time.

Add new adaptive component: the adaptive component in this architecture is implemented as an Axis2 adaptive handler. The invocation of new handler to the Axis2 web services framework do require configuration, especially the modification of Axis2.xml file. The Axis2.xml file identifies the handler chain of the Axis2 Framework, which includes the list of handlers invoked in both Inflow and Outflow. Therefore, the whole Axis2 Web Service should be reconfigured and restarted to invoke the new adaptive components.
Based on the above discussion, rating for each sub-category of usability quality attributes is:

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new adaptive component</td>
<td>Middle</td>
</tr>
<tr>
<td>Add new Adaptive Services</td>
<td>High</td>
</tr>
<tr>
<td>Maintainability quality attribute of the architecture</td>
<td>( (1/2 \times 1) + (1/2 \times 2/3) = 5/6 )  ( \rightarrow ) High</td>
</tr>
</tbody>
</table>

5.8.6 Programmability

Programmability of the architecture is based mostly on developer experience while implementing the prototype. The programmability could be divided into two sub-categories, including how much effort is required to implement the architecture candidates and to which degree the testing mechanisms are supported.

Efforts required implementing architecture: The prototype of the architecture includes 3 components, and each component is implemented using different framework, APIs. The lightweight portal could be implemented with Visual Studio 2005 and ASP.NET 2.0; the adaptive portal component is developed based on portlet API, JSP, WSRP and JSR-168 specifications; and the Axis2 web services require particular knowledge about handler chain, message context and how Axis2 Web services and clients are developed. Even if the supports for these technologies are available, it still requires programming efforts from developers to integrate these components together. Moreover, some technologies are still in incubated stage, such as WSRP4J component. The support for these technologies are, therefore, very limited.

Testing mechanisms supported: The components of this architecture are distributed in 2 or even 3 computation nodes, therefore debugging and testing of these components is a challenging problem. For example: the communication between these components is based on SOAP message. It is possible to monitor the incoming and outgoing SOAP messages and unit testing for each component, but it is difficult to perform structural testing for whole architecture together. Moreover, because the invocation of adaptive component only happens during the runtime, the debugging
Enabling mobile access to adaptive services and portal

mechanism for structural testing this adaptive component is not supported. The only strategy left is black-box testing, which may be supported by the logging system.

Based on the above discussion, rating for each sub-category of programmability quality attributes is:

<table>
<thead>
<tr>
<th>Quality attribute features</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efforts required implementing architecture</td>
<td>Low</td>
</tr>
<tr>
<td>Testing mechanisms supported</td>
<td>Middle</td>
</tr>
<tr>
<td>Overall programmability of the architecture</td>
<td>$(1/2 \times 1/3) + (1/2 \times 2/3) = 1/2$</td>
</tr>
<tr>
<td></td>
<td>Middle</td>
</tr>
</tbody>
</table>

### 5.9 Step 7: Present the results

The overall result from this evaluation is presented in the radar diagram in Figure 28 with each axis representing an individual quality attributes.

![Figure 28 - Overall Evaluation Results of the Adaptive Middleware Architecture](image-url)
Enabling mobile access to adaptive services and portal

It can be seen from this diagram that the architecture may sacrifice the programmability and portability in order to enhance the usability, performance and dynamism of the adaptive components. The main objective of the architecture is to enhance the adaptiveness of the middleware so that it can be invoked from mobile device platform. Therefore the key quality attributes of interest are usability and dynamism of adaptive components. The evaluation of this architecture candidate shows that these key quality attributes could be achievable.
CHAPTER 6:

Conclusion

About this chapter

This chapter reviews the research and development being carried out during the project. It also discusses the areas of future research that can be conducted later.
6 Conclusion

6.1 Summary
In summary, the thesis has proposed an adaptive middleware architecture, which could be used to enable mobile access to third-party web services or remote portlets. The main obstacle that limits the portability of these web services and portlets on mobile platform is the lack of adaptiveness from the server side application. The mobile platform is a resource-limited environment, in which responses from a non-adaptive web application cannot be handled in an appropriate manner. In order to enable web services and remote portlets in mobile platform, the current middleware architecture needs to be enhanced with adaptive components. This project proposed a middleware architecture which could capture, analyse and response adaptively to the current environment conditions of the mobile devices. This minimizes the risks associated with the resource limitations from the mobile device platform.

6.2 Contribution
The main contribution of the thesis is to propose an adaptive middleware architecture, which could enhance the portability of web services and remote portlets on mobile device platform by using the adaptive components.

Chapter 3 has discussed in detail the architecture of the adaptive middleware which enable mobile access to web services and portlets adaptively. The current approach of web services invocation from mobile platform does not support the adaptability of the system to the current environment conditions. Therefore it does not minimize the risk associated with the resource-limitation problems from mobile devices. The architecture discussed in chapter 3 reduces this risk by embedding adaptive components inside the architecture, which in turn could capture, analyse the environment conditions and response to these conditions in an appropriate manner, which can be handled by the mobile devices.

The quality attributes of the proposed architecture could be captured, evaluated in chapter 5 using a case study. In this case study, the proposed architecture would invoke the third-party Amazon Web Service from mobile devices client. The quality attributes of the
Enabling mobile access to adaptive services and portal

architecture includes usability, performance, maintainability, portability, programmability, and dynamism of adaptive components. Among these quality attributes, usability and dynamism of adaptive components are two most important non-functional requirements that can guarantee the mobile access to third-party web services. An evaluation of this case study shows that the proposed architecture would be able to fulfill these two key requirements, even if it may incur implementing and testing efforts.

6.3 Suggestions for future work.

Although the architecture has proved to be able to enable adaptive mobile access to third-party web services, future research is still required to enhance portability of the architecture. Specifically the following point could be considered as future work:

- The adaptive middleware architecture in Chapter 3 has only embedded the M-A-P-E pattern. It is also possible to apply the Policy Point Pattern to enhance the adaptiveness of the system. The Policy Point Pattern requires that policies have to be defined in a standard external form. The architecture could be extended to handle these external policies.

- The case study that demonstrates the middleware architecture is relatively simple and lacks of some important features. For example: the Adaptive Analyser Component in Adaptive Portal Component is not fully implemented; as well as the front-end lightweight mobile portal. For the evaluation purposes of this case study, it is acceptable to omit these features; however, in order to have a more detail analysis, these components have to be improved.

- Currently, the quantitative measurements of the performance quality attribute are not fully performed (i.e. the round-trip-time, bandwidth usage). These measurements could be performed in the next iterations.

- The case study has focused on how the middleware architecture can be used to invoke third-party web services from mobile platform. However, it omits the second contribution of the architecture, where the mobile devices can access to the remote portlets. The prototype of this second scenario is worthy to be considered as the future work.
References:

- Apache Pluto, 2007, Welcome to Pluto [Online], Available at: 
  [http://portals.apache.org/pluto/] (last accessed: 22/11/07)

- Apache Jetspeed2, 2007, Jetspeed-2 Pipeline [Online], Available at: 
  [http://portals.apache.org/jetspeed-2/guides/guide-pipeline.html] (last accessed: 
  22/11/07)

- Apache Axis2, 2007, Apache Axis2 Architecture Guide [Online], Available at: 
  [http://ws.apache.org/axis2/1_1_1/Axis2ArchitectureGuide.html] (last accessed: 
  22/11/07)


- Castle, B. (2005) Introduction to Web Services for Remote Portlets [Online], 
  Available at: [http://www.ibm.com/developerworks/webservices/library/ws-wsrp/]
  (last accessed: 22/11/07)

  Aspect Oriented Middleware Architecture for Adaptive Mobile Computing 
  Applications. IEEE Conference Proceeding 2, 81-86.

  Business Portal. [Online]. Available at: 
  22/11/07)
Enabling mobile access to adaptive services and portal


Enabling mobile access to adaptive services and portal


• WIKIPEDIA (2007) Web Services. [Online]. Available at:

  JavaWorld Magazine Article. [Online]. Available at:
  accessed: 15/11/07)
Appendices

Appendix A: Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>JSR-168</td>
<td>Java Specification Requests-168, so called Java Portlet Specification</td>
</tr>
<tr>
<td>WSRP</td>
<td>Web Services for Remote Portlets (Section 2.4.3)</td>
</tr>
<tr>
<td>MEMS</td>
<td>Method for Evaluating Middleware architectureS (chapter 4)</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration (section 2.3)</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol (section 2.3)</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language (section 2.3)</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards (section 2.4.3)</td>
</tr>
<tr>
<td>M-A-P-E pattern</td>
<td>Monitor – Analyse – Plan – Execute pattern (section 3.2.1)</td>
</tr>
<tr>
<td>PIP</td>
<td>Policy Identification Point (section 3.2.2).</td>
</tr>
<tr>
<td>PEP</td>
<td>Policy Enforcement Point (section 3.2.2).</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Definition Point (section 3.2.2).</td>
</tr>
<tr>
<td>SPI</td>
<td>Service Provider Interface (section 3.3.3)</td>
</tr>
<tr>
<td>MEP</td>
<td>Message Exchange Patterns (section 3.3.4)</td>
</tr>
</tbody>
</table>
Appendix B: Detail measurement of Adaptive Component Overhead

Adaptive Component Overhead while handling request from only 1 client:
Enabling mobile access to adaptive services and portal

Adaptive Component Overhead while handling request from 10 clients concurrently:

10 threads

![Graph showing Adaptive Component Overhead with and without a handler for 10 threads.](image)

With Handler
Without Handler
Enabling mobile access to adaptive services and portal

Adaptive Component Overhead while handling requests from 20 clients concurrently:

20 threads
Enabling mobile access to adaptive services and portal

Adaptive Component Overhead while handling requests from 30 clients concurrently:

30 threads

With Handler
Without Handler
Enabling mobile access to adaptive services and portal

Adaptive Component Overheads while handling requests from 40 clients concurrently:

40 threads

- 91 -
Adaptive Component Overhead while handling requests from 50 clients concurrently:

![Graph showing performance with and without the handler](image-url)