國立交通大學

資訊工程系

博 士 論 文

分散系統管理機制應用於企業 Web Services 環境之研究

Distributed Systems Management for Enterprise Web Services Environment

研 究 生：吳瑞祥

指導教授：袁賢銘　教授

中 華 民 國　九 十 六　年 六 月

分散系統管理機制應用於企業 Web Services 環境之研究

Distributed Systems Management for Enterprise Web Services Environment

研 究 生：吳瑞祥　　Student：Ruey-Shyang Wu

指導教授：袁賢銘　　Advisor：Shyan-Ming Yuan

國 立 交 通 大 學
資 訊 工 程 系
博 士 論 文

A Dissertation

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

分散系統管理機制應用於企業 Web Services 環境之研究

研究生：吳瑞祥　　　　　指導教授：袁賢銘

國立交通大學資訊學院

資訊工程系

摘要

Web services 透過標準界面的制定，提供了企業系統整合的良好解決方案。Enterprise Web Services Environment(EWSE)利用了這個優點，整合了企業的應用程式以及商業流程。雖然 EWSE 對整合帶來了相當多的好處，然後要為其加入管理機制並不容易。Web Services Distributed Management (WSDM) 是最常用來管理 Web services 的標準之一。然而，實作 WSDM 介面相當複雜，因為程式設計師需要了解許多相關的 Web services 標準以及開發 WSDM 相關的程式碼。此外，WSDM 只能管理單一服務，無法掌控服務間的互動以及訊息的流程。因此，EWSW 通常需要額外的軟體來處理這方面的問題。

在這個研究中，我們將簡化 WSDM 介面實作的難度並提供 EWSW 的訊息導向管理機制。藉由必須開發的管理功能程式，自動產生 WSDM 的 Web services 介面。開發者可以專注於管理程式，毋須深入了解 Web services 標準。另一方面，在不更動現有程式邏輯的情形下，便可以為現有的 Web services 自動提供訊息流程導向的管理，讓系統管理者掌握系統服務的狀態，做為企業檢視服務的使用狀況。最後，我們將展示一個參考架構，做為企業建立管理系統的參考。提供企業在不耗費太多成本的狀況下建立一個有效率且具延展示的 EWSE 管理系統。

Distributed Systems Management for Enterprise Web Services Environment

Student: Ruey-Shyang Wu　　　*Advisor: Dr. Shyan-Ming Yuan*

Department of Computer Science
College of Computer Science
National Chiao Tung University

Abstract

Web services define standard interfaces those provide good solutions for enterprise integration. Enterprise Web Services Environment (EWSE) uses the advantage to integrate enterprise applications and business processes. Although EWSE is excellence in integration, it is not easy to add management mechanism. Web Services Distributed Management (WSDM) is one of the most used standards to manage Web services. However, to implement the WSDM interfaces is complexity because programmers have to understand many Web service standards and develop program code for WSDM. Besides, WSDM can only manage single service. It does not cover service interaction and message flows of Web services. EWSE needs additional software to handle those processes.

In this research, we will simply the WSDM programs development effort and provide message flow oriented management in EWSE. The Web service interfaces are generated based on program code that implement management function and must be created. Programmers only focus on management functions and do not have to understand those Web services standards. On the other hand, the management system provides message flow oriented management atomically without modifying service code. Managers can know and control all flows as well as adjust business processes at any time. Finally, we present reference architecture for enterprise to build management system. Therefore, enterprise will not spend too much effort to build an efficient and extensible management system in EWSE.

誌謝

能完成本篇博士論文必須感謝許多人，首先感謝我的指導教授袁賢銘博士，謝謝老師多年來全力的指導並給予充分的發揮空間，才能學習到多方面的知識並獲得各種寶貴的經驗，也才能完成這份博士論文。同時也要感謝施仁忠教授、彭文志教授以及陳俊穎教授，對於論文中的架構、技術以及內容，適時給予重要的指引。在此要特別感謝所有的校外口試委員，曾黎明教授、楊竹星教授、鄭憲宗教授與羅文聰教授，在百忙之中給予我寶貴的建議，讓我可以讓本篇論文更加的完善。

此外我也要感謝分散式系統實驗室的各位學長，張玉山、梁凱智、何敏煌、焦信達、葉秉哲、許瑞愷、劉旨峰、蕭存喻、林獻堂以及其他學弟們，在博士生涯中，給予我的幫助，讓我度過許多的難關。這邊要特別感謝葉秉哲、邱繼弘、鄭明俊，我們互相打氣，一起作研究，一起玩樂，一起寫論文。要不是你們，我的博士班生涯不會多采多姿，也不會充滿著歡笑。另外也要感謝系辦的楊秀琴小姐、俞美珠小姐、陳小翠小姐給予我的所有協助。

最後將此論文獻給我的父母、大哥、二哥、妹妹以及惠屏，感謝你們一路的支持並提供我如此好的學習機會與環境，沒有你們，我不能在無後顧之憂的情況下，完成學位。求學的過程中，需要感謝的人實在太多，在此一併致謝。同時，希望在未來可以貢獻所學於社會上。

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Recently, Web services have become pervasive and critical to business operations in enterprise. They simplify interoperability and application integration. They provide a means for wrapping existing applications so developers can access those applications through standard languages and protocols. For the reason, many corporations make the effort to build the Enterprise Web Services Environment (EWSE), which integrates enterprise applications and supports business operations by using Web services technologies. In EWSE, enterprises can flexibly solve enterprise-wide integration challenges and thereby quickly address ever-changing business challenges and opportunities.

EWSE contains not only Web services but also service interactions. Those interactions between Web services are the key to support business process. Once services and business processes become operational, their progress must be managed and monitored to offer a clear view of those services perform within their operational environments. Besides, those services and business processes should provide a way to perform control actions to modify and adjust their behaviors. Therefore, managers will know enterprise status and make response quickly. To archive the goals, it requires distributed management solutions for EWSW.

Management in this case is defined as a set of capabilities for discovering the existence, availability, health, and usage, as well as the control and configuration of resources, where resources are defined as Web services, components of the Web services architecture, as well as roles undertaken in this architecture. It does not only handle single service status, but also cover service interaction and business process.

Although there are many Web services management standards, most of them cannot cover both single service management and services interactions. They usually define Web services management interfaces for management operations as well as message parameters. Because those management standards use Web services technology, different management applications and services can exchange information easily. However, those standards focus only on management protocols and functions. It means that both system designers and developers have to pay extra efforts for those management protocols and functions. Because management functions cannot solve business problems directly, some corporations may feel that developing those functions waste many resources.

In the meantime, controlling and monitoring single service are not enough. Without handling services interactions properly, it is not easy to guarantee that they can deal with all requests accurately. Some industry products and standards provide functions to manage business processes, like Business Process Execution Language for Web Services (BPEL4WS). It defines application interfaces to develop a service and use an engine for business processes execution as well as services communications. Conforming to the standards, all Web services should follow the application interfaces and can only interact with the engine. The drawbacks of this architecture are that a Web service cannot be managed out of engine. Besides, developing a service becomes very complex because programmers have to follow the application interfaces. For the reason, it is necessary to develop a solution that can manage business process without complex developing.

Because management in EWSE is quite different from traditional enterprise systems, it will have different requirements and solutions. It should solve the following problems:

1. A uniform management interface: a uniform management interface can manage all services from a single management application. It will define how to monitor service status and configure services to archive management goals. A uniform and standard interface can reduce system administrator’s effort. In EWSE, Web services technology is a good choice to create the interface.
2. Services interoperability monitor: services interoperability can composite complete services or business processes. Tracking and handling interoperability can show not only the status of each service request, but also the entire EWSE status.
3. Integration with existed environment easy: to add management functions, applications have to be modified many program code. For SOA, the program code is usually very complex. It is necessary to make it become easy.

In this dissertation, a distributed management mechanism for EWSE is proposed. It provides management interfaces for each Web services and handles message flows among services with the minimal effort. Programmers can only create the management functions without writing Web services definitions and knowing several standards. Because the Web services definitions and codes are generated, programmers can more focus on management functions. On the other hand, using Aspect-Oriented Programming (AOP), the message tracking features can be added into existed system without modifying service code. At system run-time, because the management service and business service are separated, they will not influence each other. It can guarantee the performance of business services. Finally, we provide reference architecture for adding management features in EWSE. Enterprise can use the management mechanism to build up better quality services and then enhance its productivity as well as competitiveness.

The dissertation is organized as followed: Chapter 2 introduces the backgrounds and industry status. Chapter 3 shows our management architecture and components. Chapter 4 will discuss the interactions between Web services and monitor mechanism. Chapter 5 is the reference architecture and program examples. Chapter 6 will evaluate the system. Finally, Chapter 7 is the conclusion and future works.

2. BACKGROUNDS

2.1. Enterprise Web Services Environment

Web services are architectural styles whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. The attractiveness of Web services is its open architecture. By defining standard invoking methods, different services with different implementation on heterogeneous platforms can be integrated together smoothly.

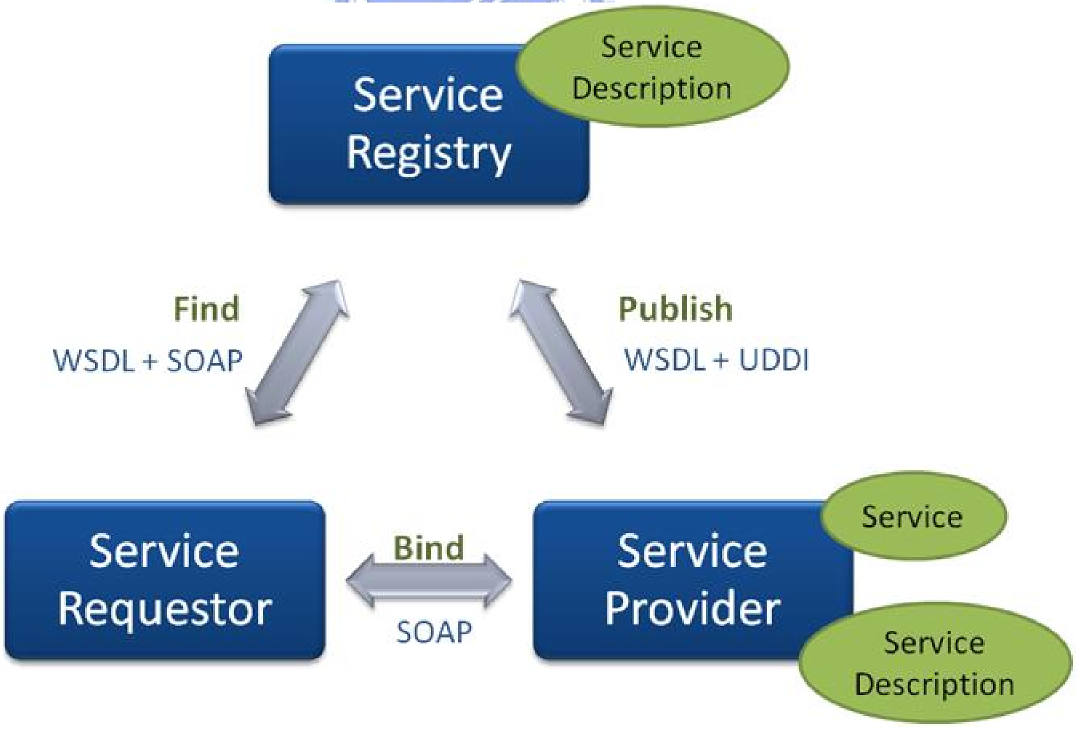


Figure 2-1 Relationship of Web Services and Three Fundamental Standards

Web Services is based on XML [7] and defines three fundamentals standards, showed as Figure 2-1. SOAP (Simple Object Access Protocol) [14] can invoke other services; WSDL (Web Service Definition Language) [15] can describe the interfaces of a service; UDDI (Universal Description, Discovery and Integration) [31] can be used to find the location of the service that is needed. Web Services can also support several transportation protocols, such as HTTP and MOM (Message-Oriented Middleware). Because Web Services has those characteristics, it can be rapidly deployed on many different platforms and let them to work together. EWSE uses Web services technology to create the flexible information technology (IT) environment. Figure 2-1 shows its architecture.

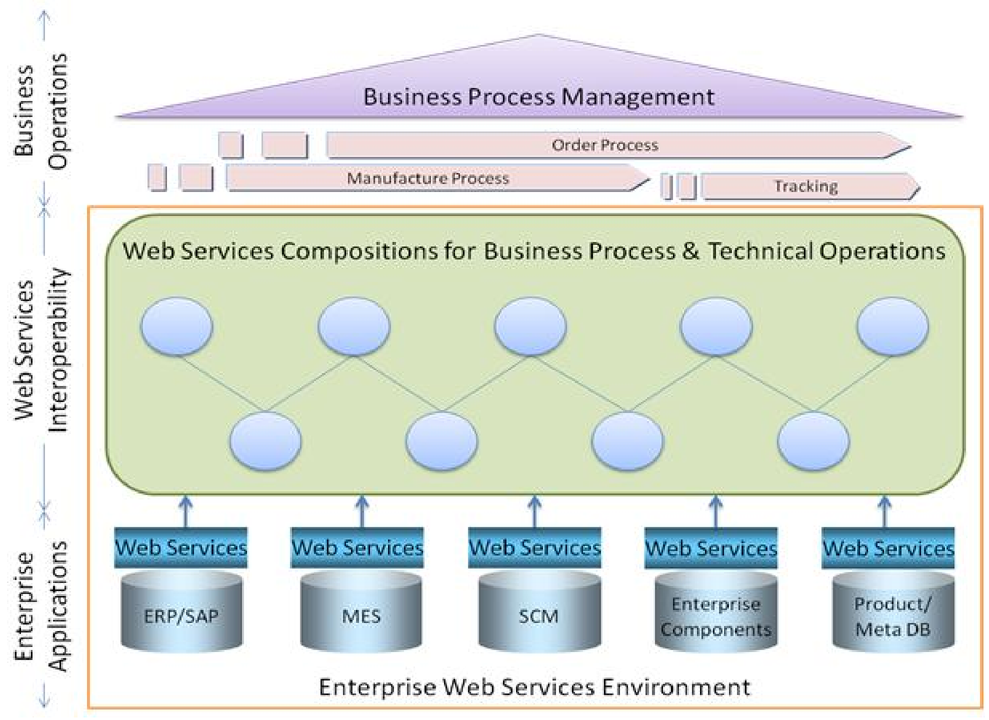


Figure 2-2 Enterprise Web Services Environment

EWSE integrates enterprise application to support business operations. Top of the figure is business operations. Every corporation owns their business processes, such as order process or manufacture process. Enterprise managers have to define those processes in some systems, execute them and monitor their performance. To perform a business process, many enterprise applications are involved. The middle layer of Figure 2-1 shows the interoperability. The services interaction will fulfills business process. The fundamental applications exposes Web services and are composed at this layer. Those applications are at the bottom of the figure. Because all applications use Web services technology and follow standards, EWSE provide good solution for message exchange to narrow the gap in integrations.

2.2. MANAGED RESOURCES

A managed resource is the management information which, from the management point of view, describes a resource to be managed, monitored, and controlled. In other words, a managed resource is an abstract representation of a real entity to be managed. The real entities in distributed systems (such as network devices, end-user applications, etc.) need to be captured and represented as managed resources.

- A managed resource is an abstraction that is available to the manager and services. Some other mechanism, outside the scope of management architecture, maintains the relationship between the managed object and the actual resource.
- A single managed resource may represent a single resource or many resources.
- The same resource may be represented by a single managed object or by a number of different objects, each representing a particular aspect of the resource.

- Not all resources need to be represented by a managed resource. This does not mean that such resources do not exist, only that they are not available to the management system.

The concepts for describing objects and information relevant to management are collectively called information models. The information model approach used to described managed objects could be entity-relationship, data-type, or object-oriented. For example, Internet management uses simple data types (scalars and two-dimensional arrays of scalars) to represent managed objects. OSI management on the other hand uses an object-oriented approach for its information model.

For each managed resource, the following items should be considered:

1. Identification: It describes basic information of a services and static management information, such as relationship of other services and standards that are supported.
2. Availability: A service should avoid system failures to response correct information to the client.
3. Metrics: Evaluate the performance and health of services.
4. Operations: Operations are all function of a service. They include business and technology functions.
5. Configuration: A service behavior will change if the service configuration is changed.
6. Events: By defined events, management system can notify system managers or other procedures to take this event.

2.3. Management Standards

There are many standards those define the communication protocol and what agent should have, such as SNMP (Simple Network Management Protocol) [1] DMI (Desktop Management Interface) [2] , CIM (Common Information Model) [3], WBEM (Web-Based Enterprise Management) [5] and WSDM (Web Services Distributed Management) [6]. All of them provide communication protocols and programming interface so that they can be used for most applications.

The SNMP is an application layer protocol that facilitates the exchange of management information between network devices. It is part of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth. SNMP is widely deployed in many systems because it is quite simple and easy to use. However, using SNMP, the interfaces should be defined during program compiling time. It cannot add and more a managed unit dynamically.

The Desktop Management Interface (DMI) provides management protocol and architecture. Every computers and hardware are installed a single agent that can collect all management information. Unlike SNMP, it is unnecessary to install agent for each component and can add management functions dynamically.

Although DMI can integrate management information on a single computer, it does not define methods to collect information from several computers in a distributed environment. CIM is developed to solve the problems. It has management interfaces and the models that describe the relationship between manageable components. Based on the model, each component can exchange information and the whole system can be managed. Because those components in a model should be tied together, DMI is not suitable for those components that are loosely coupling. In other words, if a unit is unknown at design time, it cannot be managed by DMI.

JMX provides a Java technology framework for building distributed, Web-based, modular and dynamic solutions for managing as well as monitoring devices, applications, and service-driven networks. Its management standard defines a complete management architecture for Java applications, including an API to instrument applications with manageability. JMX, an open technology suitable for management and monitoring wherever these are needed, is a standard expressly designed for adapting legacy systems, implementing new management and monitoring solutions and plugging into future devices. JMX standard enables Java applications to be managed without heavy investment, including an API to instrument applications with manageability. JMX also uses protocol adapters to enables Java to integrate with existing management solutions. On the contrary, WSDM is a set of management specifications from the WSDM Technical Committee in OASIS that provides more structures and semantics for a manageability interface than JMX does.

WSDM contains two major specifications, Management of Web services (MOWS) and Management using Web services (MUWS). The WSDM specification defines how the manageability of Web service endpoints and resources exposed as Web services can be accessed via Web services. In order to achieve this goal, MOWS is based on the MUWS specifications, and the architecture, definitions and dependencies thereof MUWS. WSDM is suitable for heterogeneous environment. In most Web services, it is selected as management standard.

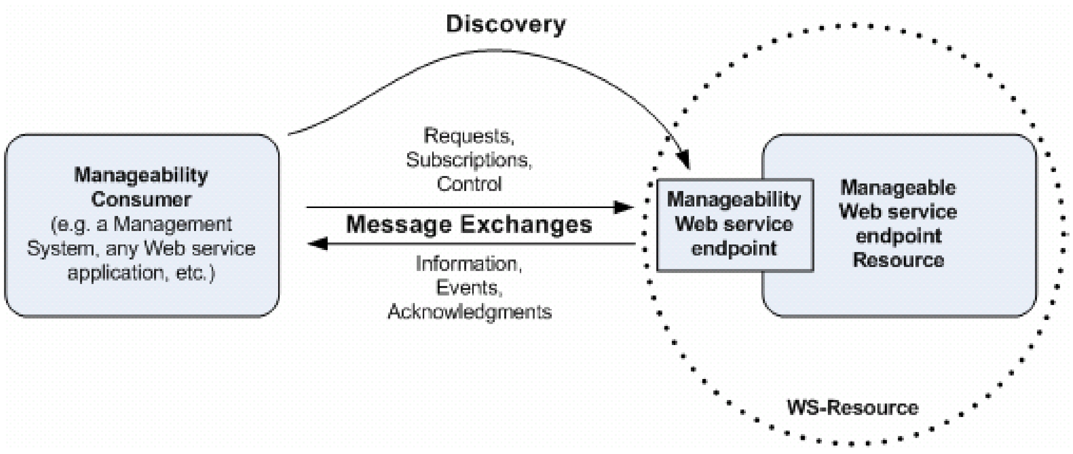

Discovery
Requests, Subscriptions, Control
Message Exchanges
Information, Events, Acknowledgments
Manageability Consumer (e.g. a Management System, any Web service application, etc.)
Manageability Web service endpoint
Manageable Web service endpoint Resource
WS-Resource

Figure 2-3 Management of Web services concepts

MUWS uses Web services to achieve easier and formal management of distributed information technology resources. This is accomplished by providing a flexible framework for manageability interfaces that advantage key features of Web services and protocols. Figure 2-3 describes the concepts: the manager employs a Web service endpoint adapter for a resource. Currently, the focus of the MUWS specification is on how to provide access to manageable resources, which are the central element and focal point of the WSDM architecture. The message patterns encapsulate access to resources into manageable resources instead of exposing them to indirect access to resources through such as agents, proxies, or observers. MUWS benefits greatly from Web services technology, for example from isolation from implementation and easily understood representation.

2.4. Industry Service Management Products

Many software products provide management capabilities for enterprise software well, including commercial products and open source software. Since enterprise Web services environment is more and more popular, Web services management has became one of the most features for the software. Some famous software includes:

Tivoli [33]

IBM provides a complete management product, Tivoli. It has business process management and control managed resources. It provides API for developing management applications. Tivoli also has a rich client that can monitor and control managed resources. Tivoli can be easy to integrated with IBM products and widely used in industry. Recently, IBM adds WSDM specification into the product so that it can communicate with other company products easier.

Unicenter products [30]

Computer Associates sells a management product suite Unicenter, which contains the product WSDM. WSDM enables different management areas, predominantly performance and security. The monitoring modules (called management observers) and security enforcement points can be implemented as SOAP processing handlers inside application servers or as independent proxies. Both business activity monitoring and business-level agreement management are driven by policies defined with WSDM.

AmberPoint [29]

AmberPoint has several products for WSM: Express, Management Foundation, Service Level Manager, and Exception Manager. These products address different management functional areas. The closest to my research is AmberPoint Service

Level Manager that enables definition (using Web forms), monitoring, and management of custom-made SLAs. Human administrators play the central role in these activities, particularly management. Monitoring is performed by agents, which can be plugged into provider Web services or executing between Web services as proxies or T-filters (T-filters observe the traffic, but do not act as intermediaries). The other two important components of the AmberPoint architecture are decentralized task-specific analytical servers that process management information and distributed management applications for interaction with human administrators.

All of those products are devoted to provide easy-to-use and rich functions for application management. Although they have powerful functions, programmers still have to pay great efforts to develop management functions. Programmers should follow their architecture and APIs so that the service can be managed. If an existed service needs to be managed, extra code is needed and the service has to be rewritten. Besides, most of those products focus on single Web services management. However, one business operation usually needs cooperation of several services. In other words, if one service has problems, it will make the business operations failure. For the reason, a complete management system should control the whole enterprise service environment, not only single service.

2.5. MANAGEMENT ARCHITECTURE

The management model for enterprise Web services can be separated as six layers, showed as Figure 2-4. The lower four layers access managed resources and upper two layers are used for service cooperation. The former collects information from hardware resources or software application; the latter coordinate the information and make those services co-work. Assuming that the interfaces between the layers do not change, the layered architecture allows changes to occur within the layer in relative isolation without affecting the other layers. This improves the scalability of the architecture as well as quality and testability.

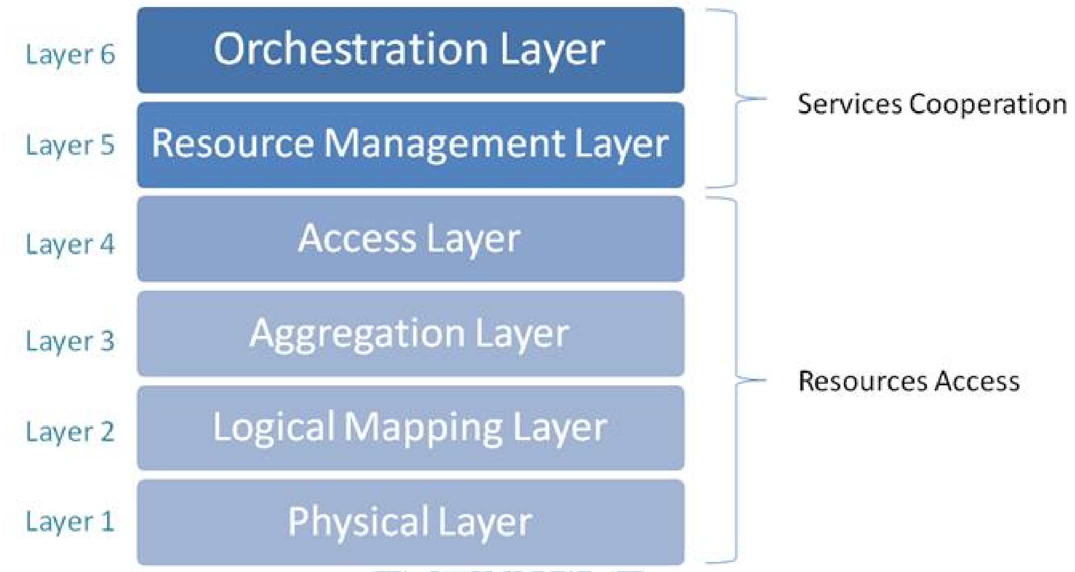


Figure 2-4 Enterprise Web Services Management Layer

1. Physical Layer: The bottom layer or platform layer describes physical elements that are being managed. This includes such things as processors, disk drives, memory, controllers and sensors.

2. Logical Mapping Layer: This layer consists of software drivers and providers used to map logical representations to the physical elements within the platform layer. Communications with the physical elements such as software and firmware create logical elements that are associated to physical elements.

3. Aggregation Layer: The Aggregation layer is responsible for the aggregation of logical representations to various data models used to correlate and describe the managed elements. This layer creates static and dynamic

elements that may be used to represent information to the logical mapping layer.

4. Access Layer: The access layer passes control and monitoring information between the management system and the managed node. This layer decodes the requests and passes them to various lower layers.

5. Resource Management Layer: The Resource Management layer, within the management system, enables the system to request and parse responses for resource management functions. This layer establishes the connection, encodes the request, decodes the response, detects any request failures and then passes the response back to the management system.

6. Orchestration Layer: The optional Orchestration layer provides a higher level of capability to management systems. This layer provides a level of management as it enables policy-based management via a policy engine with defined services levels. This layer, which sits on top of the resource management layer, reviews information and events and perform automated actions based on sets of predefined conditions

Most management application and standards focus on layer 4 and layer 5, especially in distributed environment because the management application and managed resources are not at the same host. Agent-based is the most common solution architecture for communication between layer 4 and layer 5, showed as Figure 2-5.

1. Management applications: the console provides management information that comes from managed units. It displays system status, usages, and related

information. It can also set system configuration for each components. By management applications, the system administrator can control the whole system status.

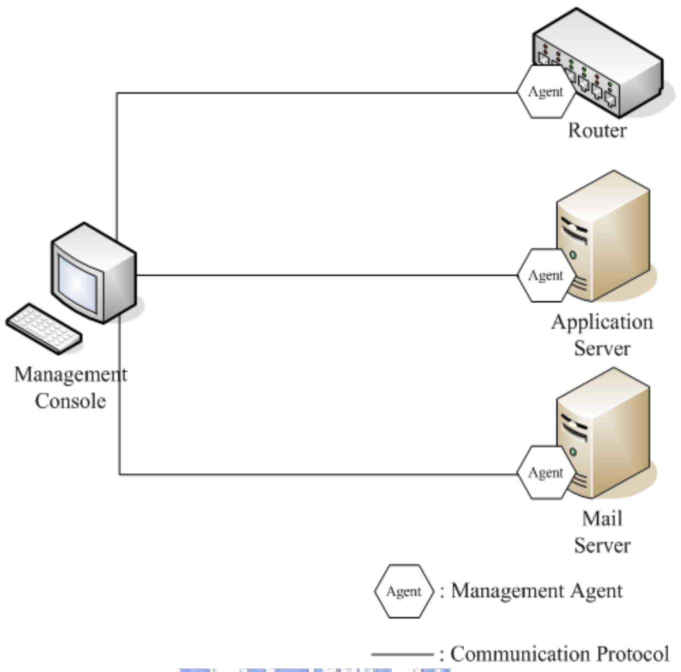


Figure 2-5 Agent-based Management architecture

2. Agent: the agents are installed at the managed resources. Agent is the bridge between management applications and managed resources. For each resource, the agent collects management information and sets configuration for this unit. It will also communicate with the management applications to send management information and get setting data.

3. Communication Protocol: the communication protocol defines the way to let agent and management applications to exchange information. The protocol will define how to initial the connection between the management applications and the agent. It will also define how to get information from agent and send configuration to agent.

In this research, the management system will focus on layer 3 to layer 6 and use agent architecture. It will try to plug management functions into Web services easy, provide necessary information to management applications and orchestrate business processes.

3. MANAGEMENT ARCHETUCTURE FOR ENTERPRISE WEB SERVICES

In this chapter, we will present the management architecture and system implementation. The discussion about message tracking and management mechanism are showed in Chapter 4.

3.1. OVERVIEW

To provide flexible and extendable management architecture in EWSE, the agent-based architecture is proposed. Agent can collect management information from managed resources and provide the information to the management applications using the predefined protocols. Then, system administrator can monitor and control the services. The proposed management system is developed in Java environment with JavaEE architecture. Although there are many Java-related implementation in this system, the same concept can be applied other architecture or programming language, for example Microsoft Web services architecture and C#.

3.2. SYSTEM ARCHITECTURE

The management system architecture is showed as Figure 3-1. In this architecture, several components are designed and mapped to the layers in Figure 2-4 so that each component has its major functions and can communicate with each other well.

Those components include *Management applications* (Layer 5, 6), *Agent* (Layer 4), *Management API/Hook API* (Layer 3) as well as *Policy & Configuration*.

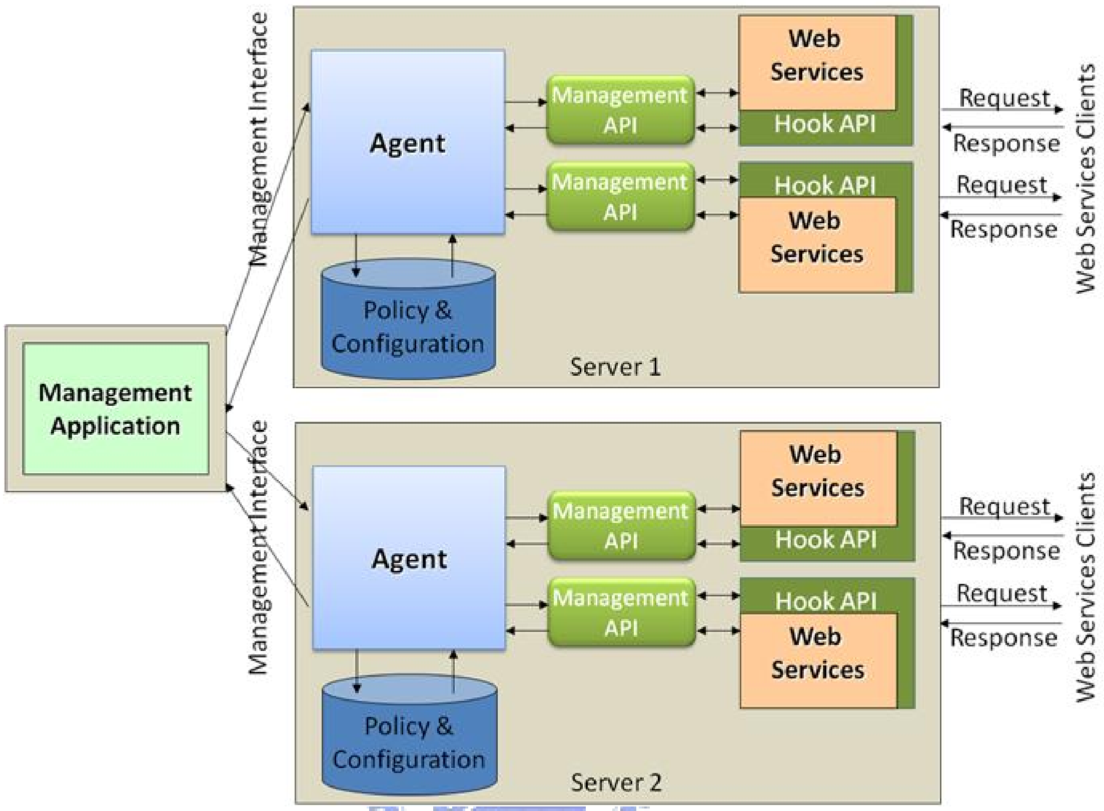


Figure 3-1 Management System Architecture

For each service, it will integrate *Management API* / *Hook API*. The two APIs are responsibility for monitoring Web services status and providing necessary information to agent. The APIs can monitor the whole process from invoking service to returning result. When a request is sent to the services, the APIs captures the request first. They will process the request and record necessary information. All messages in EWSE should contain an identification content added by other *Management API/Hook API* in the management system. The content is useful for message flow tracking. The two APIs will process not only incoming but also outgoing message. If a service issues requests to another service, the APIs will involve the request before the message is sent. They add message tracking information into the message. Therefore, all messages flows in the management system are monitored.

Agent is deployed between *Management applications* and services as well as exposes WSDM interfaces. It reads the system policy and configuration from the database, gets service status from *Management API/Hook API*, and communicates with management applications. Because *Management API/Hook API* exports configuration operations for services, *Agent* can get services status quickly and set parameters to services directly. On the other side, the management information will send to management applications when it requests. Management applications can also send service parameters to *Agent* to adjust service behavior. For each host, there is only one *Agent* deployed. In other words, each service would not have the WSDM interfaces. All managed resources only communicate with *Agent* but not management applications directly. *Agent* handles those interfaces. It will provide communication interfaces for both managed resources and management applications. *Management API/Hook API* collects information from managed resources and sends to *Agent*; on the other side, management applications can query or send data from or to *Agent*. Single management interfaces on *Agent* can reuse the similar management functions and coordinate resources needed for management purposes. If *Agent* is not here, management applications have to build several connections for each service and managed resources have to create management interfaces by themselves. On the contract, with *Agent*, the number of connection for a host is only one. *Agent* helps to reduce Web services implementation effort and improve system performance.

For each *Agent*, *Policy & Configuration* database is deployed. It is the storage that *Agent* can read and save information. Because the policy or configuration is complex and consisted by many parameters, it is not easy that management applications set those parameters one by one, especially when parameters may affect each other. One error would cause service crash. Each *Agent* can save those parameters by themselves. If necessary, those settings can be retrieved quickly. It guarantees system stability and save configuration effort.

Because there are many heterogeneous systems in an enterprise, it will become very difficult to manage them if each system has its own management methods. For example, if there are ten systems and each of them has specified protocol, the management applications should implement ten protocols. For the reason, a common communication protocol is necessary. The management applications can get information or set configuration from/to *Agent* by using the consistent protocol. Using standard protocols can let that Management applications developed by different vendors can communicate different Agents.

For each component in this architecture, the invocation flow is showed in Figure 3-2.

There are four operations in this flow:

(1) Client invokes Enterprise Web Services. Every invocation is intercepted by *Management/Hook API*. The APIs will process the request SOAP message and resend it to the actual services. When the request is completed by the service and it will return data to the client, the return data is also intercepted by the APIs. The APIs will add management information into the return message. Finally, the APIs will return result to the client. During the processing, enhanced message is used between the client and the APIs. Therefore, management system can control message flow more precisely.

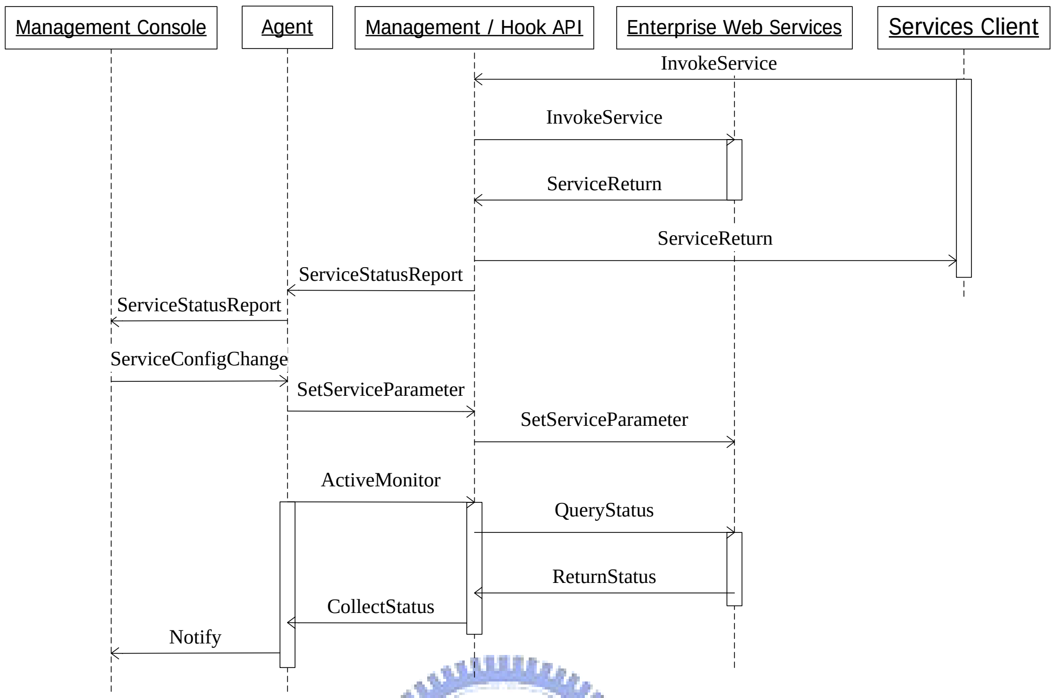


Figure 3-2 Management Invocation Flow

(2) Management applications get services status from *Agent*. After invoking services, *Management/Hook API* will collect invocation information. The APIs will report that information to *Agent*. *Agent* will coordinate the information and save into database. When Management applications request the information from *Agent*, *Agent* will retrieve them from database and send to management applications. *Agent* has standard operations based on MOWS and MUWS for management applications to lookup services and get management information. For the reason, different management applications can communicate with Agent well.

(3) Management applications send configuration to services. If system administrators want to adjust service behavior, they can specify configuration

from management applications. The configuration may influence several services. Management applications will send parameters to corresponding services based on the configuration. After *Agent* gets those parameters, it will change services configuration using *Management/Hook API*.

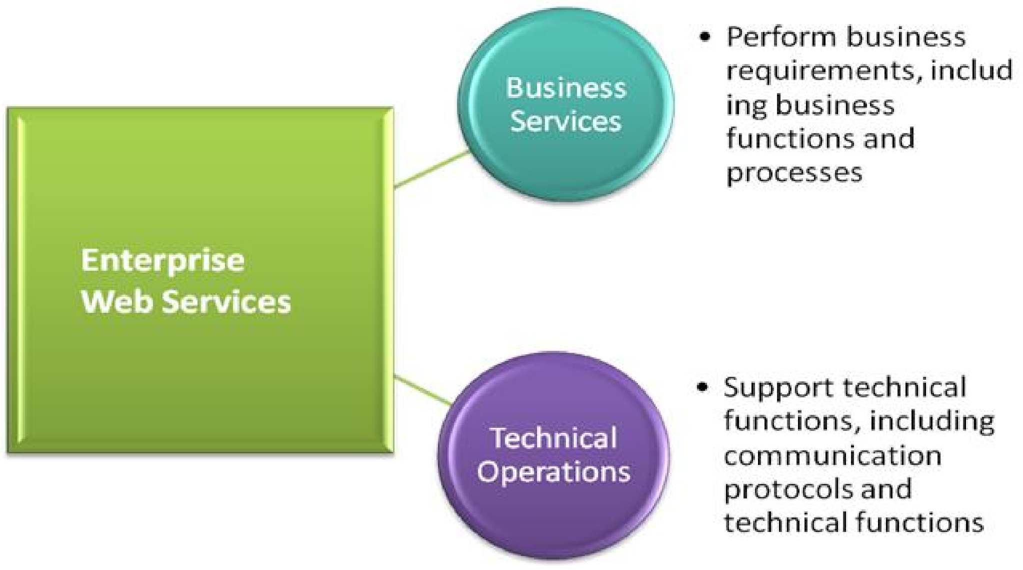
(4) Management applications are notified when events occurred. After *Agent* startups, it continues to monitor services status through *Management/Hook API*. The API can query services status. If there are any events, *Agent* will get the message. Then, it will send a notification to management applications. System administrator will recognize the events quickly.

For the traditional architecture, each service has to implement those management interfaces. It increases implementation efforts. Besides, once the management interfaces changed, all services has to be re-implemented. At the run-time, Management applications has to connect to each service by themselves. It will increase management overhead. While *Agent* controls all services on a host, not only management functions can be controlled but also the management overhead can be reduced.

Before describing detail design for every component, a characteristic of the enterprise web services is introduced. Each service contains several operations. They can be distinguished as Business Services and Technical Operation roughly.

3.3. BUSINESS SERVICES & TECHNICAL OPERATIONS

Every service in enterprise is built for business purpose. It solves specified business problems. For example, a *ProcessOrder* operation in a service is used to handle order. Those operations can be mapped to business purposes. However, it is not enough if there are only business services. In distributed environment, some technical operations are also provided to support specified technical requirements. Most Web services export WSDL for service consumers to retrieve service description so that the consumers can know how to invoke the service. For the reason, technical operations are also important. A server that can perform business services well relay on those technical operations.

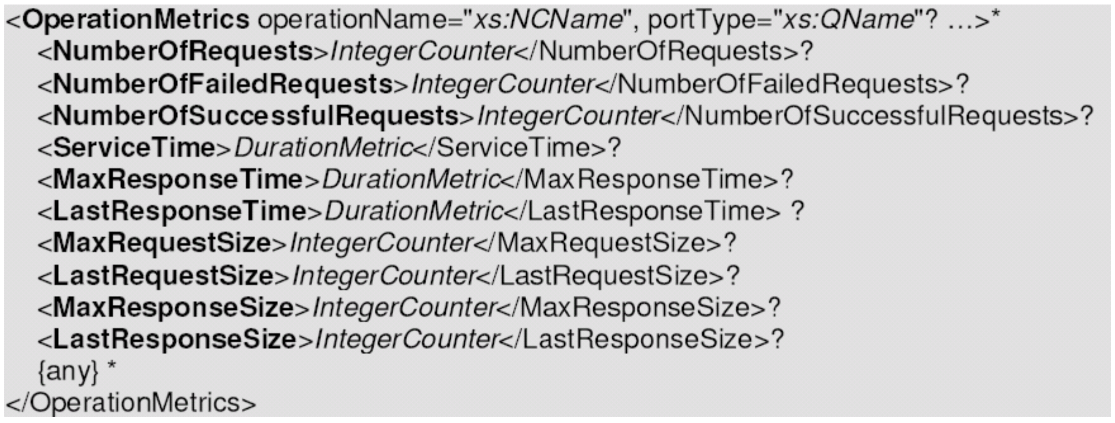


Enterprise Web Services
Business Services
Perform business requirements, includ ing business functions and processes
Technical Operations
Support technical functions, including communication protocols and technical functions

Figure 3-3 Enterprise Web Services Contains Business Services and Technical Operations

Generally, Business Services fulfill business requirements and those requirements are usually different. Therefore, only the different business requirements are implemented. If there are similar requirements at two systems, there will be only one implementation and the other will reuse it.

On the other hand, Technical Operations at different services are similar. For example, if one service needs transaction support, it has to implement those technical operations and protocols of Web service transaction. Because those requirements are similar, a reused program code can be created and all systems will use it. In the management system, there are several necessary *PortType* and context, showed as Figure 3-4. In the description, the OperationMetrics specify the complex property of operation metrics. For developers, each service has to implement OperationMetrics so that service metrics can be accessed. Otherwise, it cannot be accessed by the Management applications. However, the metrics operation is similar. It can be implemented once and other services re-use it to save the development effort.



```
<OperationMetrics operationName="xs:NCName", portType="xs:QName"? ...>*
   <NumberOfRequests>IntegerCounter</NumberOfRequests>?
   <NumberOfFailedRequests>IntegerCounter</NumberOfFailedRequests>?
   <NumberOfSuccessfulRequests>IntegerCounter</NumberOfSuccessfulRequests>?
   <ServiceTime>DurationMetric</ServiceTime>?
   <MaxResponseTime>DurationMetric</MaxResponseTime>?
   <LastResponseTime>DurationMetric</LastResponseTime> ?
   <MaxRequestSize>IntegerCounter</MaxRequestSize>?
   <LastRequestSize>IntegerCounter</LastRequestSize>?
   <MaxResponseSize>IntegerCounter</MaxResponseSize>?
   <LastResponseSize>IntegerCounter</LastResponseSize>?
   {any} *
</OperationMetrics>
```

Figure 3-4 Web Services Endpoint Operation in MOWS

On the other hand, it is not easy to plug the management technical operations into an existed service if they are not existed in a service. The general steps to add the management functions are showed as Figure 3-5. First, a WDSL file is necessary. It can generate the skeleton and stub for serve side and client side, respectively. Then, developers can develop the service using the generated code. Finally, the service has to be deployed at the server. The procedure takes a lot of time. For most managers, they do not accept to spend resource when the Business Service does not change. For the reason, an efficient way is necessary.

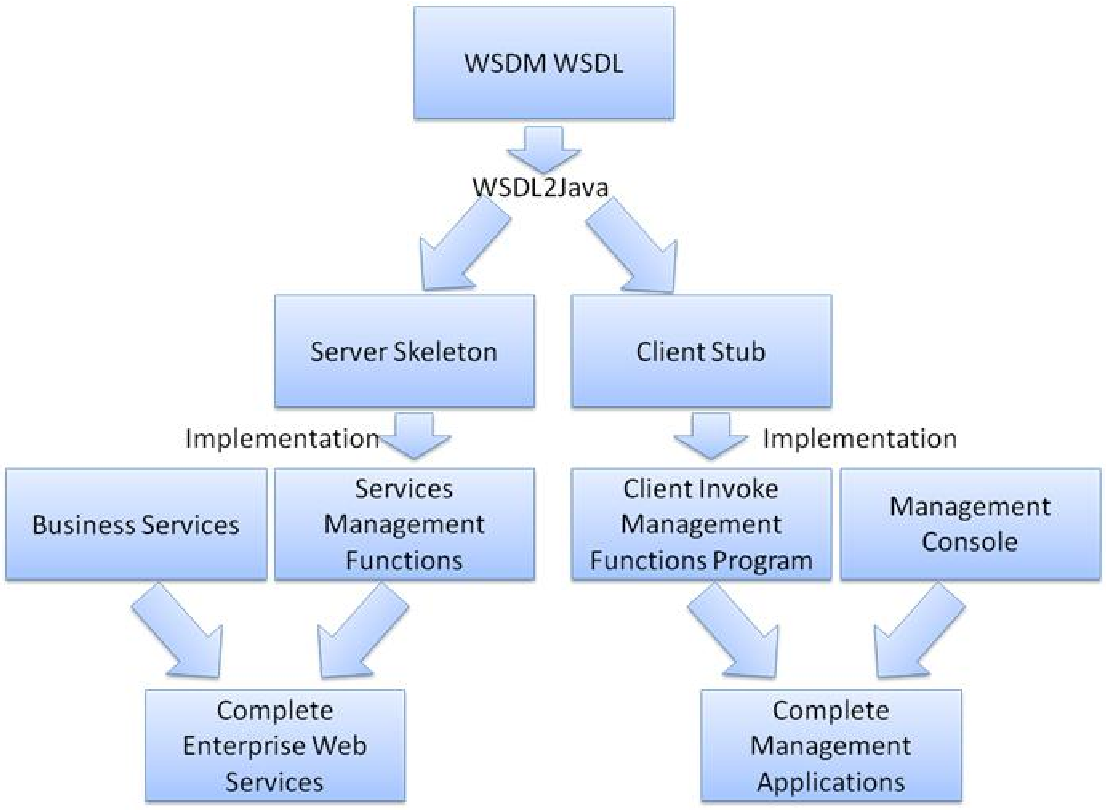

WSDM WSDL
WSDL2Java
Server Skeleton
Client Stub
Implementation
Implementation
Business Services
Services Management Functions
Client Invoke Management Functions Program
Management Console
Complete Enterprise Web Services
Complete Management Applications

Figure 3-5 Implementation Flow from WSDM WSDL to Program Code

3.4. Management API

To access and configure services are the most fundamental functions in a management system. It is also the most important part. However, it usually costs a lot of time to develop those access and configuration functions. Although there are many management products, most of them provide APIs that focus on management protocol and management applications. Programmers still have to follow the management architecture and APIs carefully when developing a service. Otherwise, the service cannot be managed.

In EWSE, it is necessary to define managed resources and provide concise software libraries for developers. Developers, then, can write the management code using those libraries. For the reason, an approach that can generate Web services interfaces and add management capabilities into services is proposed. They are *Management API*. *Management API* provides functionalities those will get the status of managed resource. Programmers can export service status to agent using this APIs.

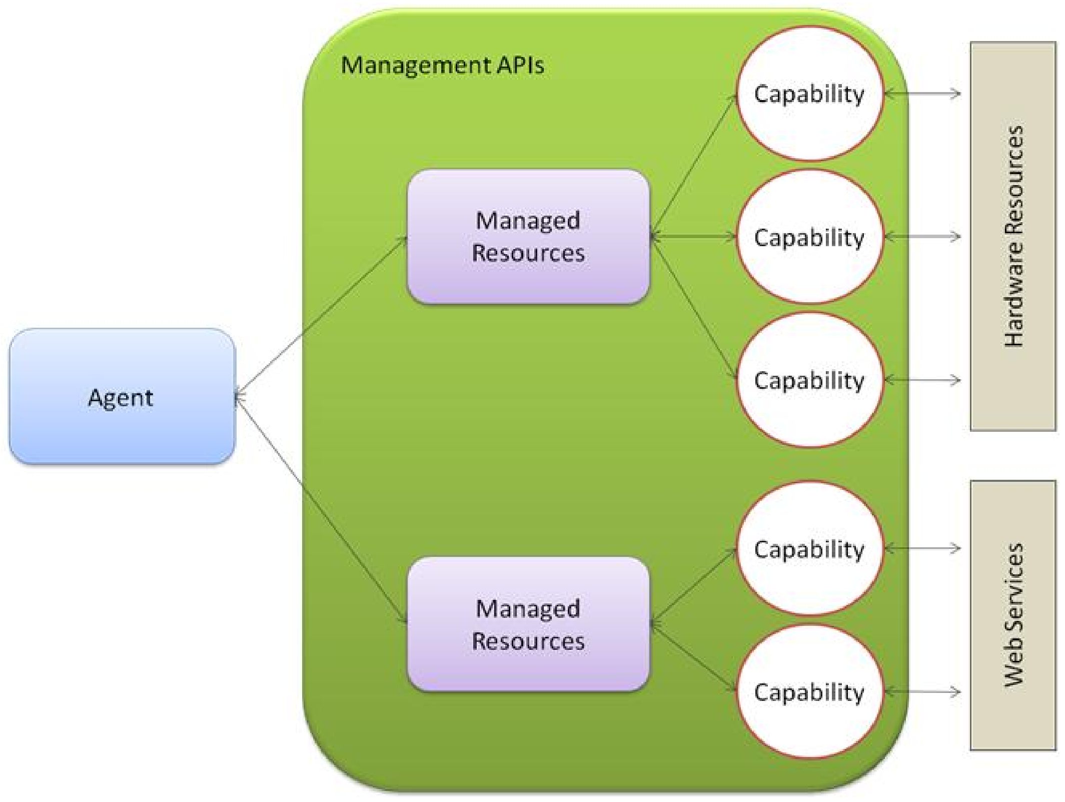


Figure 3-6 Managed Resources and Capability Object in Management API

Management API focuses on how to export managed resource status and management functions to the management applications. It contains management capabilities and management resource. Management capabilities indicate what kind of capabilities the resources are. One resource may contain several management capabilities. Their relationship can be showed as Figure 3-6. Those capabilities and resources are associated with a physical hardware or Web services.

Those capabilities contain management information and operations those describe the status and operations of the back-end resources. The managed resources expose the capabilities to be used in management for *Agent*.

3.4.1. ICAPABILTY INTERFACE

The capability represents management properties and operations for a service. It is atomic unit in the management system and contains the capabilities of a resource. Single capability shows one kind of management information for a resource. For the reason, a resource will have several capabilities.

The capability is represented as a Java class and defines an interface for this kind of class. There is an interface class, *ICapability*, which uses both WS-Resource as well as WSDM specification and defines several methods, showed as Figure 3-7.

```
import javax.xml.namespace.QName;

public interface ICapability
{
    // Capability lift-cycle control methods
    void initialize();
    void initializeCompleted();
    void shutdown();
    void shitdownCompleted();

    // Get properties for management based on WS-Resource Framework
    QName[] getPropertiesName();

    // Get operation names for management
    String[] getOperationNameforManagement();

    // Web Services specified methods
    String getNamespaceURI();
    String getNamespacePrefix();
}
```

Figure 3-7 ICapability Interface

ICapability interfaces only use simple Java language features so that programmers do not concern about WSDM and WS-Resource specification. Programmers only create a capability class that implements the interface. They only indicate the management operations, properties as well as information of Web services for management. Two methods of the interface, *getPropertiesName()* and *getOperationNameforManagement()* shows properties and operations of resources respectively. In the implementation class, the *getPropertiesName()* will return all properties of this capability. The method uses WS-Resource specification and defines each property as QName. At the same time, the implementation should contain all getter and setter methods for all properties. Therefore, M*anagement API* can know the properties of the capability implementation by *getPropertiesName()* and access them by those getter/setter methods. Another method *getOperationNameforManagment()* is the similar and shows the operation names for the managed resrources. The different is that the getter and setter methods are not implemented but the methods in the returned names should be implemented. In this case, the capability can expose properties for management and operations to use.

The properties and operations in *ICapability* implementation class should implement and provide management functions as well as information of a resource. The managed resource components will access them only through *ICapability* implementation class. For the reason, the properties and operations should use APIs of the resource to get the actual status.

There are another two methods related to Web services: *getNamespaceURI()* and *getNamespacePrefix()*. The two methods define the namespace used in generated WSDL file. The namespace is formatted as "http://schemas.xmlsoap.org/wsdl/soap/" and prefix is "http". They are used to generate the namespace of XML schema in the WSDL file. The schema can describe the content so that elements and attributes in the WSDL are correct.

To handle capability life-cycle, there are four methods, *initialize(), initializeComplete(), shutdown(),* and *shutdownComplete()*. When the capability implementation class startup, the *initialize()* method is called. When all capability implementation classes associated with the same resources startup completely, the *initalizeComplete()* method will be called. The same for *shutdown()* and *shutdownComplete()*. When the capability class start to shutdown or the associated capabilities are shutdown completely, *shutdown()* and *shutdownComplete()* will be called, respectively.

After creating *ICapability* implementation classes, *Managed Resource* components will process the interface definitions and expose those classes as Web services.

3.4.2. MANAGED RESOURCES

Every *ICapability* implementation class should register itself with Managed Resource components. The Managed Resource component will take two major actions:

1. During deployment, Managed Resource will generate WSDM related files and classes. Those files include WSDL and resource description. Those classes will

have the Web services features classes those are corresponding to ICapability implementation class and WDSL file. At this phase, error detections are added. For example, if the getter and setter methods for a property are not found, a WS-Resource property error will be raised.

2. During run-time, Managed Resource will aggregate those ICapability implementation classes as management functions and expose them to Agent. Agent, then, can access the system information and manage it through Managed Resource components. Besides, Managed Resource components also take care the life cycle of every capability. It will invoke the four life cycle methods at right time.

Generate the Web services related files can simply developing complexity program and save implementation effort. The generated flow is showed as Figure 3-8. ICapability helps to generate the WSDM WSDL. The WSDL will produce the skeleton and stub as the traditional flows. After the generated program, a specified classes, Service Management Functions, is created, which will use the ICapability implementation classes. Because ICapability classes have implemented the management function, the Service Management Functions can use them directly without writing the program again.

Programmers can only write the ICapability only when creating WSDM interfaces; on the contract, in the traditional program model, programmers have to create both management functions and WSDL by themselves. For the reason, the developing process is much simplify.

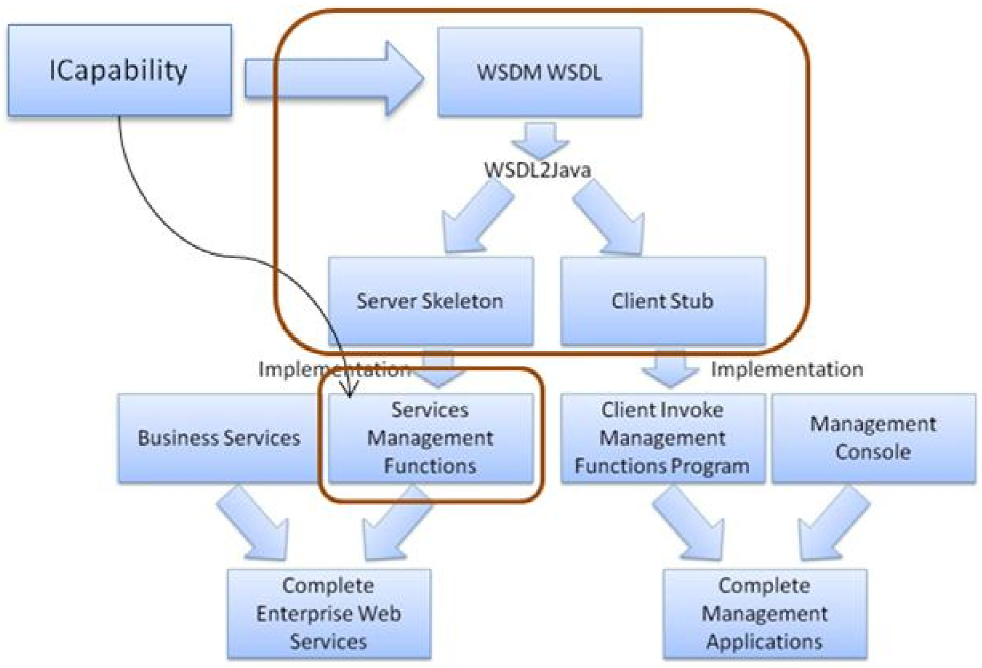

ICapability
WSDM WSDL
WSDL2Java
Server Skeleton
Client Stub
Implementation
Implementation
Business Services
Services Management Functions
Client Invoke Management Functions Program
Management Console
Complete Enterprise Web Services
Complete Management Applications

Figure 3-8 The Program Code Generated Flow with ICapability

To generate the Web services files and classes, programmers only write the implementation class. Figure 3-9 shows the sample code for the implementation of ICapability. Parts of code, such as life cycle control methods, is omitted to save space.

```
package org.dcslab.wsdm;

public class MailServerCapability implements ICapability
{
    String NAMESPACE_URI = "http://ws.cis.nctu.edu.tw/org/dcslab/wsdm";
    String PREFIX = "http";

    public QName[] getPropertiesName() {
        QName[] PROPERTIES = {
            new QName(NAMESPACE_URI, "Name", PREFIX),
            new QName(NAMESPACE_URI, "Port", PREFIX)
        };
        return PROPERTIES;
    }
```

```

   public String[] getOperationNameforManagement() {
      String[] OPERATIONS = { "start","stop" };
      Return OPERATIONS;
   }

   public String getNamespaceURI() {
    return this.NAMESPACE_URI;
   }
   public String getNamespacePrefix() {
      return thus.PREFIX;
   }

   // Methods for management
   public void setName(String name);
   public String getName();
   public void setPort(int port);
   public int getPort();

   public void start();
   public void stop();
        :
        :
}
```

Figure 3-9 Sample ICapability Interface Implementation Class

In the *MailServerCapability* class, *getPropertiesName()* and *getOperationNameforManagment()* are implemented. Method *getPropertiesName()* will return two properties: name and port. Therefore, there are getter and setter methods for the two properties. As for *getOperationNameforManagement()*, two strings are returned, start and stop. So, there are two methods, *start()* and *stop()* in the class. Managed Resource components invoke the two methods to get properties as well as operations name and generate the WSDL file. Because the generated WSDL is too long to show, only the major elements and their relation are showed as Figure 3-10. It will have a *PortType*: *MailServerPortType* and many operations. The Bindings part indicates the binding type and operation encoding. Finally, the Services part shows the server location. Parts of the WSDL file is showed as

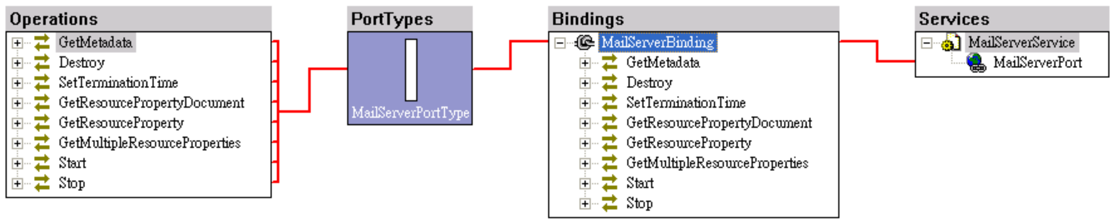


Figure 3-10 Generated WSDL Elements Relation

```
<wsdl:definitions name="MailServerResource"
    xmlns:tns=http://ws.cis.nctu.edu.tw/org/dcslab/wsdm ...>
  <wsdl:types>
        :
    <xsd:schema elementFormDefault="qualified"
        targetNamespace="http://ws.cis.nctu.edu.tw/org/dcslab/wsdm">
      <xsd:element name="Start"/>
      <xsd:element name="StartResponse"/>
      <xsd:element name="Stop"/>
      <xsd:element name="StopResponse"/>
          :
     <xsd:element name="Name" type="xsd:string"/>
     <xsd:element name="Port" type="xsd:integer"/>
     <!-- WSRP document for MailServer type -->
     <xsd:element name="MailServerProperties">
       <xsd:complexType>
         <xsd:sequence>
           <!-- WS-RL ScheduledTermination properties -->
           <xsd:element ref="wsrf-rl:CurrentTime"/>
         <xsd:element ref="wsrf-rl:TerminationTime"/>
         <xsd:element ref="tns:Name"/>
         <xsd:element ref="tns:Port"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
   </xsd:schema>
       :
  </wsdl:types>
       :
    <wsdl:message name="StartRequest">
         <wsdl:part name="StartRequest" element="tns:Start"/>
    </wsdl:message>
    <wsdl:message name="StartResponse">
         <wsdl:part name="" type="xsd:anyType"/>
    </wsdl:message>
       :
  <wsdl:portType name="MailServerPortType"
     wsrf-rp:ResourceProperties="tns:MailServerProperties">
       :
   <wsdl:operation name="Start">
     <wsdl:input name="StartRequest" ....>
     <wsdl:output name="StartResponse" ....>
       :
   </wsdl:operation>
       :
  </wsdl:portType>
       :
</wsdl:definitions>
```

Figure 3-11 Generated WSDL file

In the generated WSDL file, the properties and operations, defined in *MailServerCapability* class, are used to generate the XML schema, WSDL message and WSDL port type. Managed Resource components do not only invoke the two methods of the capability class but also analyze the properties and operations to get the data type of input parameters and return value. They are defined in the XML schema so that future invocation will use correct data type.

3.4.3. DATA TYPE MAPPING

Data type mapping focuses on data transformation between *Management / Hook API* and Web services, and vice versa. In most enterprise environment, the data format for management is defined as static because it is seldom changed. Because system has to test data format and cast it to correct type, it will cost more resource at run-time if every data is always treated as dynamic. For the reason, the static Java object is used in the production environment. The mapping is showed as Table 3-1.

Table 3-1 Data Mapping between Web Services and Java Class

Web services data type	*Java Class*
xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dataTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:Qname	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

When Managed Resources components generate WSDL from capability implementation class, those properties and operations can only use those classes in Table 3-1. In other words, programmer cannot define Java class as a property, operations parameters and return values. The reasons are:

1. The data class should be checked carefully so that it can be transferred. For Java, the data class must implement java.io.Serializable and follows serialization rules.
2. If programmers use data classes by themselves, the corresponding WSDL should contains XML schema for them. There will be a very complex <complexType> … </complexType> elements when the data class has many fields and methods. It will make a huge WSDL file. A huge WSDL file means huge SOAP message and causes bad performance.
3. For management applications, they have to discover the description of the data class from the WSDL or SOAP request/response. When WSDL changed, management applications have to recognize the change and product new client program so that it can access the management information. However, if there is only primitive data type, the process can be skipped. It can get the information from SOAP message directly.

Although there are only primitive data types, the management system still support complex data. Several capability classes can be associated with a single resource. A capability class is responsible for one management information. Using several capability classes can describe complete information for the resource.

3.5. Hook API

In enterprise, a business requirement usually involves several services. Those services have to interoperate and exchange information so that the whole the requirement can be satisfied. For the reason, it is common that one Web service invoke another service when it processes request. In enterprise environment, linking invocation chain helps to manage services. To monitor the activity of the service, all requests and invocation will be recorded for future query, showed as Figure 3-12.

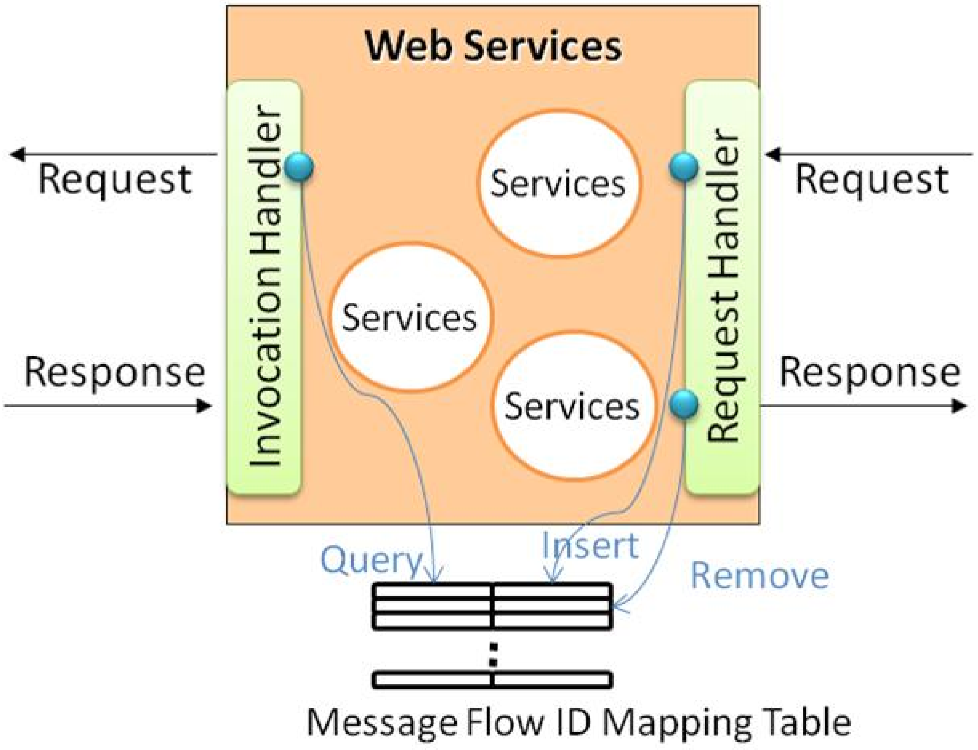


Figure 3-12 Request and Invocation Handler for Web Services

The Message flow ID contains four elements:

1. ID: a unique id that represents the Message flow ID.

2. Source: the Web service that sends the request.

3. Time_sent: the time that the source Web service sends the request.

4. Time_resp: the time that the source Web service gets the response.

Request Handler

The handler will process request/response message to/from the service. When a message arrives, it will intercept and process the message. If a message flow ID is in the message, the handler will insert the message flow id in the Message flow ID Mapping Table. Otherwise, a new message flow id is generated and inserted into the table. Then, the message will be forwarded to the corresponding Web services implementation program. After the service completes the request, it will return result to the caller. The handler will intercept the message again and look up the corresponding message flow id in the table. It will insert Message Flow ID into message and remove it from the Message Flow ID Mapping Table. Finally, the response message will be returned to the caller.

Invocation Handler

When the service invokes another Web services, the Invocation Handler will intercept the messages. The handler will do two things: (1) it will add Message Flow ID into the message. The Invocation Handler will look up Message Flow ID Mapping Table to get Message Flow ID. If there is existed ID, it will use it directly. Otherwise, it will create a new one. Invocation handler will put Message Flow ID into the SOAP message that is sent to the target service. At the same time, it will also append the Web services Port data of the destined Web services to that ID in the table. (2) If necessary, it will redirect the message destination to another one. For example, when the destination service is down for maintenance, the request should be handled by another service.

Message Flow ID Mapping Table

The table provides insert, query and remove functions. Besides, the table will keep all operations and request/response time for query services invocation status. The table can not only provide the response time of the service, but also present the correct time of message arrived and returned. Besides, several messages can be linked together with the message flow ID. It is useful to monitor the business activities in the Web service environment.

Under the architecture, if service A invokes service B and, then, service B invokes service C, they will all have the same message flow ID. From management perspective, the invocation flow is displayed and the relationship of those services can be known. The detail discussion is showed in section 4.2, 4.3 and 4.4.

3.6. AGENT DESIGN

Agent in the management system is the bridge between management applications and capability implementation class. It has the following features:

1. Web services interface: *Agent* exposes every Managed Resource as Web services. In the capability implementation class, the namespace URI is specified. The path of URI will be service location in Agent. Management

applications will only use those Web services interfaces to communicate with resources.

2. Basic technical management information: *Agent* can provide basic metric information. For example, the response time of a service.
3. Service configuration: *Agent* can read configuration from the registry and send those configuration to the corresponding services. Besides, if there is configuration sent from management applications, *Agent* could also set the related Managed Resource.
4. Security: security features can guarantee that only authorized clients can access the information.

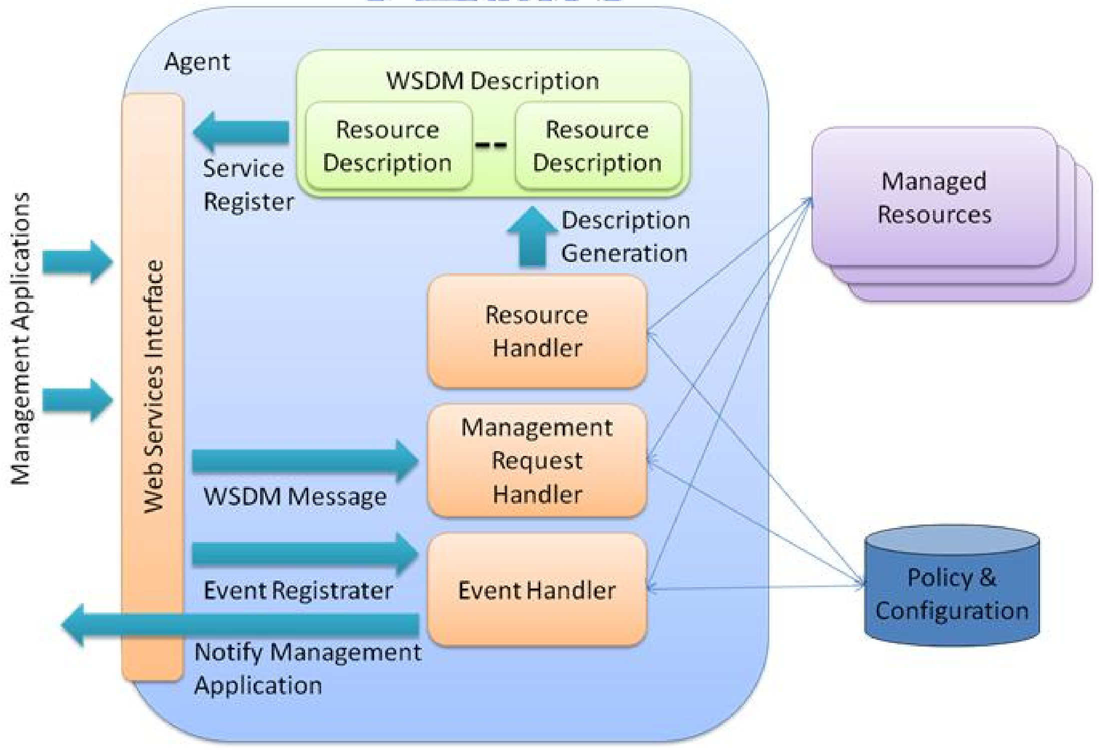


Figure 3-13 Agent Internal Components

Agent internal architecture is showed as Figure 3-13. Inside Agent, there are one data component to handle WSDM description and four components to handle different requests

WSDM Description

The data component saves the resource files, including registration file, WSDL and generated classes. Those files generated by Managed Resource components come from resource handler.

Web Services Interface

The component provides Web interfaces and handles all requests. It reads WSDM description and creates service interfaces for each Managed Resources. When a request comes, it will process it and invoke Management Request Handler to process it.

Resource Handler

Resource Handler controls life cycle of Managed Resources from installation, startup, shutdown, and un-installation. During installation, the component gets all WSDM files from Managed Resources. When system startup or shutdown, the components will invoke startup or shutdown procedure of Managed Resources, respectively. In the startup process, if there is configuration in the registry, the configuration will be retrieved and sent to Managed Resource. Finally, during un-installation, the component will remove the configuration from WSDM Description components.

Management Request Handler

When it gets requests from Web Services Interfaces, it will process the requests and call the corresponding Managed Resource to handle it.

Event Handler

Management Application can register notification so that it will get a notification when an event occurred. The event is associated with properties of resources. In other words, management applications may register "Property is equals 10" event. The Event Handler will handle those event requests, and continues to monitor resources. If the situation happens, it will notify Management Application by WS-Notification specification.

3.7. MANAGEMENT APPLICATIONS & PROTOCOLS

Management applications will access the information of resources and represent the information. They can also configure the resources so that the resources can perform better and are helpful in the enterprise environment. When there are specified events or situation happened, management applications can also get the events and response for it.

Many industry products begin to support WSDM to manage Web services or use Web services to manage applications. Those products implements pre-defined service monitor and control functions. They define service metrics and use SLA (Service Level Agreement) to manage services. Besides, those products many only support some specified specifications. For the reason, in general, the management applications can only communicate with those services used their management library. It violates the original idea of Web services.

In our system design, Agent exposes Web services interface for management and uses MUWS as communication standards. Management applications can use the WS-Propriety to get propriety of a resource and use the management functions to control the service. Under such architecture, the management applications do not need to know pre-defined information. Using the standards, management applications can get management information quickly.

For the management applications, the following protocols should support:

1. WS-ResourceProperties - GetResourceProperty operation
2. WS-MetadataExchange - GetMetadata operation
3. WSN NotificationProducer - Subscribe and GetCurrentMessage operations, as well as WS-Topics properties
4. WSDM MUWS: Identity, Caption, Description, and Version properties, OperationalStatus property.

4. WEB SERVICES PROCESS MANAGEMENT

The software as a service model can overcome many limitations that constrain traditional software use, deployment and evolution [20]. Web services architecture constructs such service environment and provides a suitable technical foundation for making business processes accessible within an enterprise because of the standard interfaces as well as communication protocols. The architecture focuses on separating the possession and ownership of software from its use. It delivers software's functionalities as a set of distributed services. Therefore, the business requirements can be fulfilled only when composing those services. Those composing Web services make up business processes that accomplish business requirements. For the reason, managing enterprise systems does not only monitor service status but also guarantee Web services process execution well.

4.1. MANAGEMENT OF SINGLE SERVICE

To perform Web services processes, it is necessary that every service can accomplish requests. Each Web service should accept the request and response correctly. Otherwise, the whole EWSE will be in abnormal status. For the reason, system administrator should monitor service status and take actions when there is something wrong.

Using the management framework with *ICapability* interface can build up management mechanism quickly. By defining suitable management functions, programmers can develop those functions easy without knowing advanced WSDM specifications. However, enterprise should define fundamental management functions those must be implemented by every service. It does great help for system management. Those functions include are listed in section 2.2. For example, all services should contain an ID to distinguish each service and define metric to evaluate performance as well as service health.

Those functions can be defined as interfaces or abstract classes. When derivative classes implement them, Agent of the management framework will generate the related WSDM interfaces. If enterprise can defines general purposes functions, for example database and mail, management functions for those features can be added into enterprise framework and will be reused in many systems.

ICapability interfaces and Agent provide an easy way to create consistent management functions to monitor Web services. They also provide a solution to reuse those management functions. Using the framework can build up the environment to control as well as handle services quickly and easily.

4.2. SERVICE COMPOSITION

The importance of service composition has been widely recognized in the distributed research community due to its high flexibility in allowing development of customized applications. Enterprise web services cannot exist along. They most co-operate with other services to accomplish business and technical needs. However, the two kinds of needs have different concerns.

4.2.1. SERVICES COMPOSITION FOR BUSINESS NEEDS

Web services are built to support business requirements. There are many concerns about services composition. For example, the selected service for composition should consider multiple criteria (e.g., duration, reliability) [18]. Some researchers define semantic Web services to help service selection [20][21][22]. After studying those researches, Web services composition should satisfy business requirements, including business functionalities and service quality. Although Web services architecture has flexibility to select suitable services, it does not define measure methods or standards. In other words, service clients have to handle and record the quality information by themselves.

Enterprise IT systems should improve and adjust by current target and status [23][24]. Nowadays, more and more corporations build not only internal systems (e.g., ERP) but also external systems (e.g., CRM, SCM, and SRM). Those systems can earn more business opportunities or reduce cost. If IT systems can adjust for the change quickly, corporations can make the best response. Web services architecture has proven the adjust solution. The standard interfaces make the communication between heterogeneous systems possible and dynamically service composition. However, it is hard to find out the service bottle without proper tracing mechanism. When a request is issued into the Web services environment, the system should know all services involved in their order. Therefore, the system can know the detail information from management framework of the services and adjust services as they need.

4.2.2. SERVICES COMPOSITION FOR TECHNICAL NEEDS

Many Web services standards defines inter-operations between services to fulfill technical needs. UDDI, one of three basic Web services standards, is for service discovery. Web services client can use UDDI to find out suitable services and its interfaces. This is the simple services composition because there are only two services involved.

Another standard, Web services transaction, defines complex services composition. Web services transaction has three specifications: WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity. To handle a transaction, a Coordinator service is necessary for each Web service. Figure 4-1 shows the message flow.

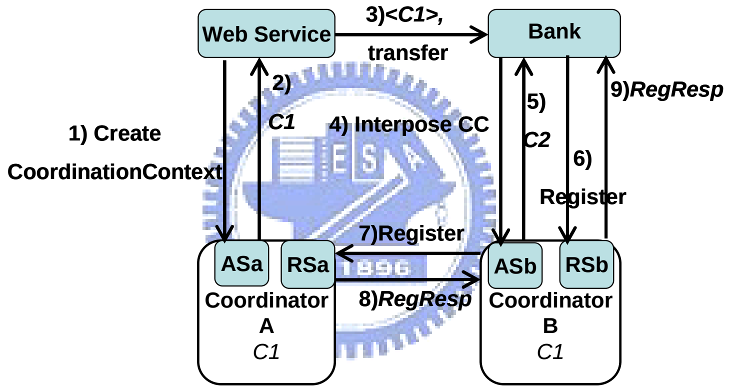


Figure 4-1 Web service create CoordinationContext from Coordinator service

In the flow, the Web service will invoke the transfer method of Bank service and declare the invocation in a transaction. There are nine steps in the flow:

(1) Coordinator A creates CoordinationContext: Web service requests a CoordinationContext from its coordinator A.

(2) Coordinator A return CoordinationContext C1 to Web service.

(3) Web service invoke transfer method of Bank service. The invocation SOAP message should contain CoordinationContext C1.

(4) Bank service requests another CoodinationContext from its coordinator B based on C1.

(5) Coordinator B returns Coordinationcontext C2 to Ban service.

(6) Bank service registers its transfer service to Coordinator B to join the transaction.

(7) Coordinator B registers itself to Coordinator A to join the transaction.

(8) Coordinator A returns registration result to Coordinator B.

(9) Coordinator B returns registration result to Bank service.

The interaction is quite complex. If there is any problem during message exchange, it will be very hard to trace the error. For the reason, a message tracing method for system run-time is necessary.

4.3. ENTERPRISE WEB SERVICES PROCESS

To know the Web services process models can be separated as two major types: sequence (showed as Figure 4-2 (A)) and plane (showed as Figure 4-2 (B)). The sequence process means that one service will invoke another service that will also invoke other services. The plane process is that one service will only invoke one service at a time; after the request completing, the service will invoke the next one. The two models can be used together, showed as Figure 4-2 (C).

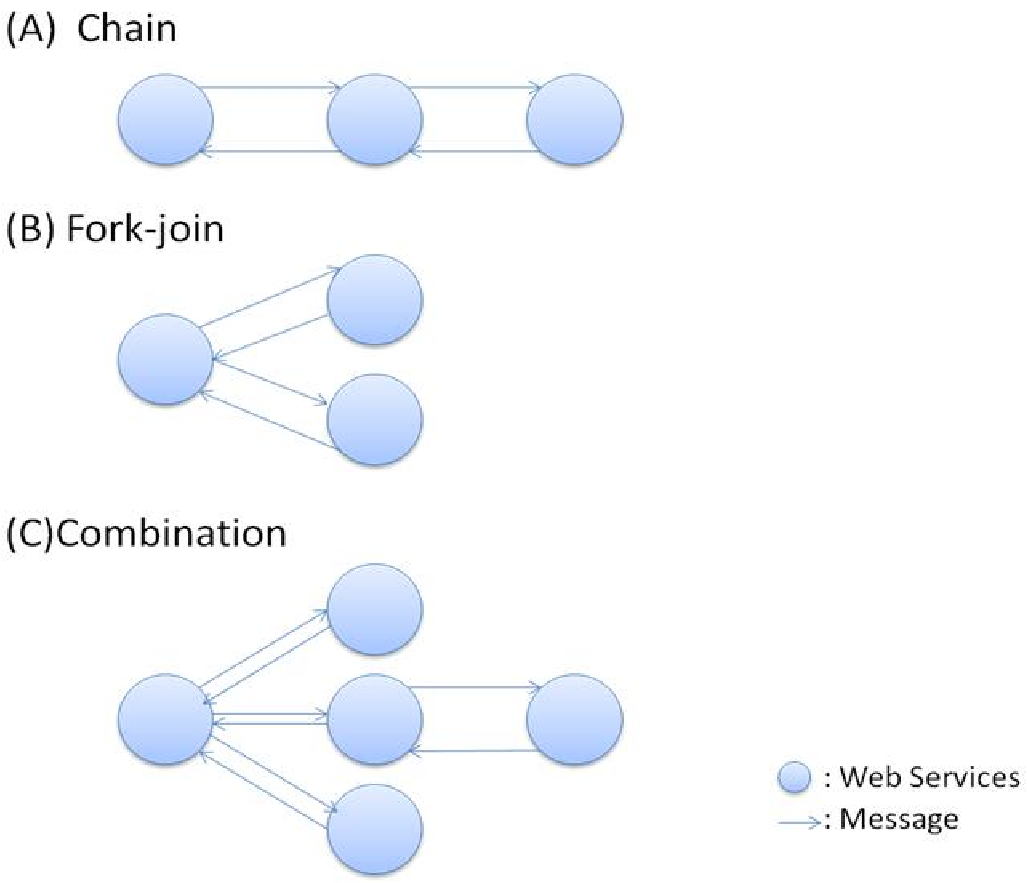


Figure 4-2 Web Services Process Model

In the management system, all messages in the same process will be assigned to the same ID that is used to trace entire process flow in management application.

Management applications can collect the message information and present the message flow among several Web services. Using the flow shows how the message is processed. At the same time, management system can identify error or exception if there is any error in the flow.

4.4. MESSAGE FLOW ORIENTED MANAGEMENT

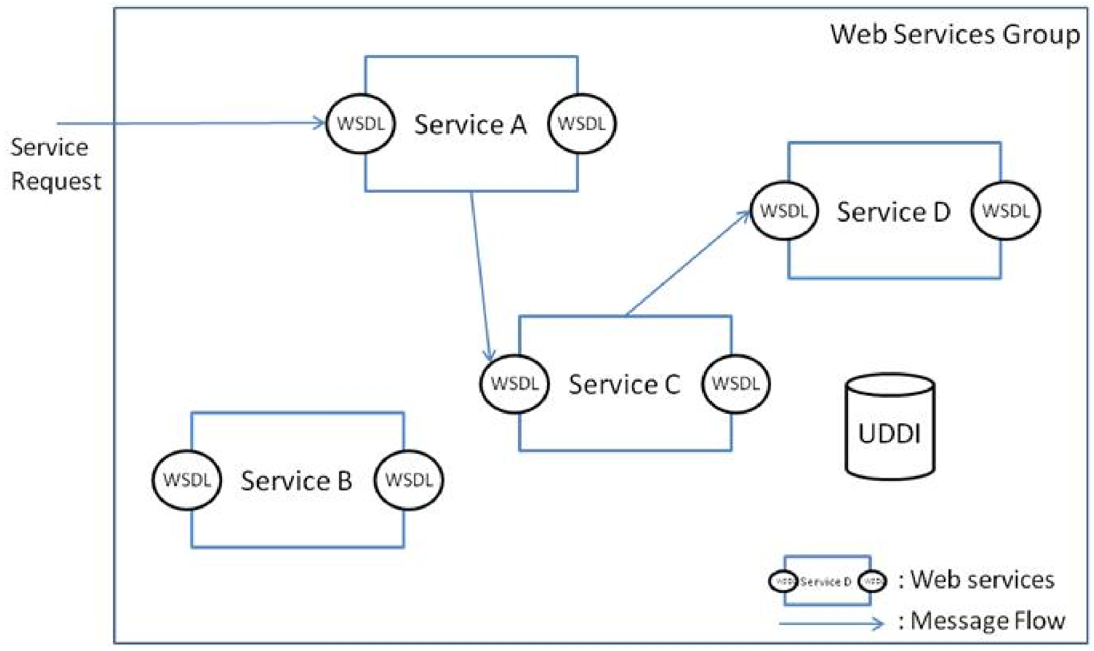


Figure 4-3 Message Flow among Web Services

Message flow represents how the message is processed. Figure 4-3 shows a sample invocation among Web services. There are many invocation happened at the same time in enterprise. It is not to find out a bottle bottleneck or problems during service invocation. In current Web services model, service clients only concern about the next service provider. Those clients only recognize that the next service has problem but they cannot provide the message source to adjust services. For the reason, to join Web services together with message flow can provide useful information to manage them.

4.4.1. INTEROPERABILITY STANDARDS

Web services society defines several standards for Web services composition. They provide an open and standards-based approach for connecting Web services together to create higher-level business processes. They include:

BPEL4WS (Business Process Execution Language for Web Services)

BPEL4WS uses XML-based language to describe the control logic for Web services coordination in a business process. A BPEL engine interprets and executes the BPEL4WS language. BPEL4WS is exposed as Web service and uses WSDL to describe the public entry and exit points of the process. Besides, it will interact with external Web services through WSDL interfaces. When invoking the public entry of the process defined by BPEL4WS, the process with message flow will be triggered. The message flow will go by the BPEL4WS flow or external Web services. Once the process is completed, it will return by the exit points.

WSCI (Web Services Choreography Interface)

WSCI is also XML-based language. It defines "Choreography" as the message between Web services that participate in a collaborative exchange. It only defines the Web services interaction without process or flow management.

BPML (Business Process Management Language)

BPML not only develop private logic implementations but also use WSCI to describe public interactions. Therefore, the process is constructed and message flow is made.

Although those specifications define the standard interacting standards for Web services, they do not provide proper management mechanism. Once the Web services group gets a service request, the group should monitor and trace the request status. Although BPEL4WS can control and manage the executing process, it takes extra effort to define business processes by BPEL4WS language. Besides, it is not suitable for all Web services process model, such as (A) of Figure 4-2 if the service invokes the next services but not returns result back BPEL4WS engine. On the other hand, both WSCI and BPML only define the message exchange interface and process flow, respectively. It is hard to know the current status of a request. Sometimes, the request will be left aside if there is no proper management mechanism. For the reason, the Web services management system should not only manage single service but also know the status of every message request.

4.4.2. MESSAGE FLOW TRACKING

Because of service composition dynamically, it is necessary to trace message flow to prevent lost or queued. On the other hand, the tracing mechanism should not affect overall system performance and cause many design as well as implementation efforts. For the reason, an automatic message flow tracing mechanism with few performance influenced is required. The message flow tracing in the management system is illustrated as three steps:

(1) Message logging: the *Request Handler* of *Hook API* will get the message flow ID from the request SOAP message and save them into *Message Flow ID Mapping Table*. When the services invoke other services, the *Invocation Handler* will get the id from the table and insert it into the request SOAP messages. Based on the propagated mechanism, the id of message flow ID in the same message flow will be the same. However, if *Request Handler* cannot find message flow ID from request message, it will create a new message flow

ID and insert it into the *Message Flow ID Mapping Table*. The remaindering process is the same as previous.

(2) Information collection: *Agent* provides Web services interfaces for management applications to retrieve those saved message flow IDs. *Agent* can collect all message flow IDs on the same host. It is unnecessary to create information provider for each one separately.

(3) Message flow analysis: after management applications get all message flow IDs from all Agents in the EWSE, they can begin to analyze the information. Message with the same id of message flow ID can be treated as the same message flow and link those processes together. Some business intelligence applications can analyze those data and message flow to improve business operations.

As the message flow in Figure 4-3, Service A, C, D will get service request in turn. Without message flow tracing, it is hard to find the relation between those services. However, in the management system, *Requestor Handler* of every service will get the ID from message and *Invocation Handler* will put the ID into the message sent to the next services. The two handlers only record the ID and time so it will not take much process time. Then, *Agent* can get the information from *Message Flow ID Mapping Table* and provide a Web services interfaces for management application. Management application gets the message flow tracing information from all *Agents* and makes up the flow. Although the final process costs a lot time, it is not done at the services side but a standalone system. Therefore, the architecture does not affect the overall system performance.

4.4.3. DYNAMICALLY SERVICE SELECTION

With the completeness of Web services environment in a corporation, the service requestor will have the chance to choose suitable service. A service is unavailable for many reasons, like maintenance, server loading high, hardware error, and software fault. Under such cases, service requestor has to issue request to another service. In current enterprise Web services architecture, it is not usually to look up UDDI before invocation for performance issues. For the reason, a adjustable solution for service selection can improve service quality.

Invocation Handler of Hook API can help to change service from one to another and redirect message to that service. Because the handler intercept the process that service requestor sent message to the service, it can re-arrange the message destination. Invocation Handler accepts administrator's configuration to create a Web services PortType mapping table with XSLT translation document. When a invocation message is sent to the original PortType field, it will redirect the message to the new Port type. At the same time, the message will be transformed based on XSLT if necessary.

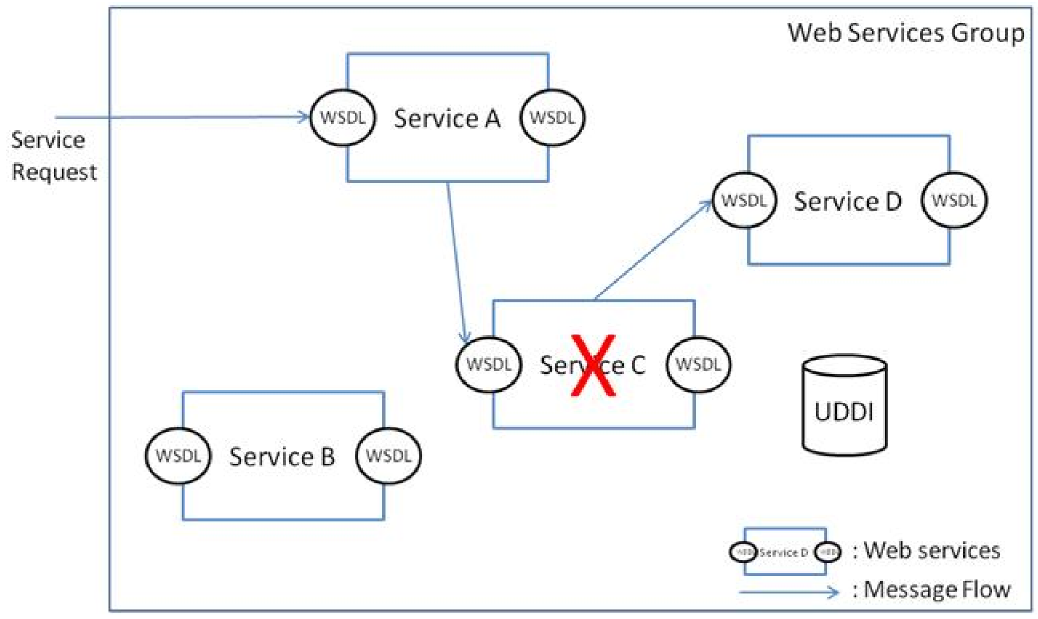


Figure 4-4 Message Flow with Service C Error

In the message flow from service A->C->D, if service C has error, service A will recognize the event and use dynamically service selection, showed as Figure 4-4. The invocation handler of service A will detect the error and choose another service B that has similar functions. It will read its input parameters and transform the SOAP message based on the predefined XSLT. Then, the invocation handler can redirect the message to the new service.

The drawback of dynamically service selection is performance impact. It invokes the desired services but gets an error message. Then, it selects other service at run-time and transforms the message to the new service. Using XSLT takes a lot of process time. Finally, it will invoke the new service again. The whole dynamically services selection process needs at least two service invocations and SOAP message transformation. The whole processes takes a lot of time. For the reason, it should be used only for abnormal status or critical services.

4.4.4. EXCEPTION AND ERROR HANDLING

If there is an exception or error during service invocation, the exception or error handling procedure will handle the exception or error. Usually, the procedure will record the problems and notify related people to eliminate it. It is the same in the Web service environment. The procedure will log the error and send a notification to the management architecture. Besides, if there are exception or error handling message flows, the flows will startup to deal with problems. The difference is the logging information for the events. If the system can provide more detail information, the error can be eliminated quickly.

In the management system, the exception and error handling mechanism is added with message tracking functions. It is one function of *Invocation Handler*. *Invocation Handler* will intercept all issued requests and know their status. It can know error occurred if it gets SOAP error code during invocation. It will know the execution Web services process that contains the error processes. In this case, the error service in the Web services process is recognized and the corresponding error handling processes also start to deal with the problems.

Exception and error handling flows in the management architecture are treated as special service selection cases. The different is that the flows for exception and error start when error occurred, but flows for service selection is used when the destination service PortType is match with one entry in PotyType mapping table. Invocation Handler will invoke the service at the first of exception or error handling flow and the mark the message as error in message flow ID.

In the management systems, exception and error handling does not only show the error but also provides message flow as well as failure service. Traditional management system can only provide failure service without message flow.

System administrators have to link the service invocation by themselves. In the management systems, the related information is provided so that administrators can solve the problems quickly.

4.5. ADVANTAGES OF MESSAGE FLOW TRACKING

In the management system, the message tracking mechanism is added into the system automatically. Developers do not have to implement any message tracking code. Request and Response Handlers of Hook API add the message flow tracking functions. They process incoming and outgoing messages. Management Applications then can get the information and link the message flows. Although there is no implementation effort, the message flow tracking can provides more powerful features than traditional web services process systems:

1. Message tracking covers both business functions and technical operations. Most Web services process systems can only handle business process. Those processes should be defined in the process system. Unlike the system, message tracking can handle all interactions between Web services.

2. No process engine is needed in this architecture. In the management system, services interactions are as before. They can invoke services directly without involving process engines. The architecture without process engines has several advantages. It will not influence system performance and the program flow is unchanged.

3. The failure service in the process can be recognized. In the management system, all message flow is recorded and traceable. System administrators can find out the status of every message. For the reason, if there is a failure in the message flow, it can be recognized quickly.

Old process management system needs a lot of effort to modify the program code and message flow as well as to deploy a process server. It costs huge effort and needs a lot of time. In the message tracking system, no program code modification is needed and message tracking mechanism is added automatically. It will give grate helps to system administrators.

5. REFERENCE ARCHITECTURE

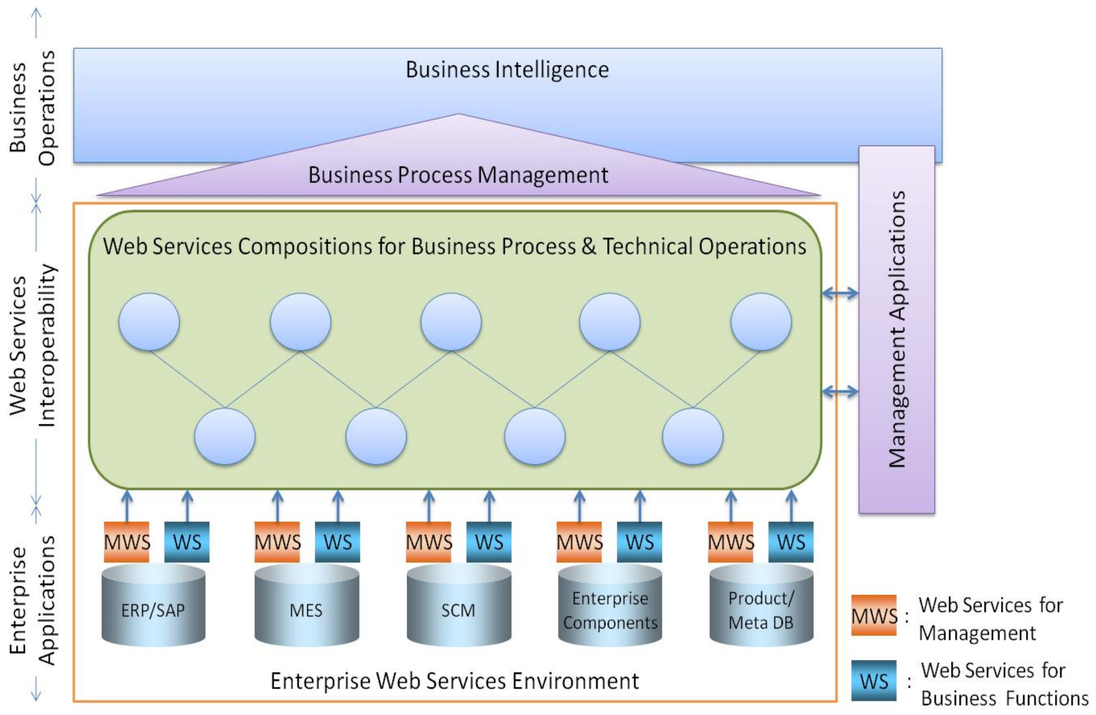

Business Operations
Web Services Interoperability
Enterprise Applications
Business Intelligence
Business Process Management
Web Services Compositions for Business Process & Technical Operations
Management Applications
MWS
WS
ERP/SAP
MES
SCM
Enterprise Components
Product/ Meta DB
Enterprise Web Services Environment
MWS : Web Services for Management
WS : Web Services for Business Functions

Figure 5-1 Reference Architecture for EWSE

The reference architecture for EWSE is showed as Figure 5-1. Each enterprise applications will have two Web services: one is for business services and the other is for management services. All Web services operations can be separated as two categories: business services and technical operations. Separating the two different kinds of services can let the two services adjustable independently. If the technical operations spends too many computing time and memory, its priority can be decreased.

At the right side, there are management applications. Those applications will access handle two things:

1. Monitor and control enterprise applications status.
2. Monitor business process.

The management applications will provide information related to technical operations. On the other side, business process management focuses on business process. With the two management systems, manager can get more detail information in enterprise and know both technical and business operations. With the two sides' information, enterprise can build up intelligence system with complete information.

5.1. WEB SERVICES FOR MANAGEMENT

In the reference architecture, each enterprise will have two Web services; one is for business operations, and the other is for management. The business Web services are unchanged. Developers only implement the management Web services.

Developing management Web services only implement the *ICapability* implementation classes. Figure 3-9 is the skeleton for developing management applications. Developers only implement the class to complete the management functions. In the sample, developers have to implement the mail server status monitor features and control start/stop functions.

The management system, then, will do the following tasks after deploying the *ICapability* classes.

1. It generates the WSDL for the *ICapability* classes. Based on the WSDL, many Web services description and code are generated.

2. Agent exposes Web services interfaces based on the *ICapability* classes. Agent also defines the endpoint and service locations. Management applications can get information through those interfaces using SOAP protocols.

5.2. Sample Messages

To get management information from Agent, management applications should follow WSDM standards. They can use *WS-MetadataExchange* to get WSDL and invoke service operations to control service status. Figure 5-2 and Figure 5-3 show the SOAP messages to request WSDL and invoke Start() operations, respectively.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
   <wsa:To ... > http://59.105.100.59/managment/services/MailServer
   </wsa:To>
   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata
   </wsa:Action>
   <wsa:MessageID ...>...</wsa:MessageID>
   <wsa:From ...> http://www.w3.org/2005/08/addressing/role/anonymous
   </wsa:From>
         :
 </soap:Header>
 <soap:Body>
   <wsx:GetMetadata
       xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
      <wsx:Dialect>http://schemas.xmlsoap.org/wsdl/</wsx:Dialect>
   </wsx:GetMetadata>
 </soap:Body>
</soap:Envelope>
```

Figure 5-2 Management Applications Request WSDL

```
 1  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 2   <soap:Header>
 3     <wsa:To ... > http://59.105.100.59/managment/services/MailServer
 4     </wsa:To>
 5     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
 6        http://ws.cis.nctu.edu.tw/org/dcslab/wsdm/Start
 7     </wsa:Action>
 8     <wsa:MessageID ...>...</wsa:MessageID>
 9     <wsa:From ...> http://www.w3.org/2005/08/addressing/role/anonymous
10     </wsa:From>
 :         :
25   </soap:Header>
26   <soap:Body>
27     <tns:Start xmlns:tns="http://ws.cis.nctu.edu.tw/org/dcslab/wsdm"/>
28   </soap:Body>
29  </soap:Envelope>
```

Figure 5-3 Management applications invoke Start()

In WSDM, all SOAP message should follow WS-Addressing specification. In the header, it describes the message source, destination and action. Line 6 of the two messages describes the action. The message body describes the actions. Line 27-30 of Figure 5-2 are the request for WSDL. Line 27 of Figure 5-3 will invoke Start() operations.

6. EVALUATION

Management functions are important in enterprise environment. Although they do not perform business operations, they do guarantee that all operations executing correctly. For a good management system, it should not cost too many resources. For development, the management functions can be implemented easy. For system execution, the management functions should not affect overall system performance too much. In this chapter, the two aspects are evaluated.

6.1. PROGRAMMING OVERHEAD

In the management architecture, the ICapability interface is used and the management logic is defined in Java class. Agent will read the class to generate WSDL interfaces and expose Web services interfaces directly. The approaches create the Web services from Java code, called Bottom-up approach. The other technique is called Top-down approach, which generate Java code from WSDL. Top-down approach is suitable for general-purpose services. However, for management systems, Bottom-up approach helps to create systems quickly and save a lot of effort. Table 6-1 shows the comparison between the two approaches for building Web services management systems.

Table 6-1 Comparison between Bottom-up and Top-down Approach for Technical Operations of Web services

Item	*Bottom-up Approach* *(from Java Code to WSDL)*	*Top-down Approach* *(from WSDL to Java Code)*
Required implementation	Only management class implementation	Management WSDL + class implementation
Code reuse	The code can be integrated with other management systems.	It is not easy. The code generated from WSDL is usually bundled with chosen Web services framework.
Inheritance from existed management functions	Yes. The derived class can inheritance functions from its parent class. The Web services interfaces will be also created at the same time.	No. Although the management code can inheritance existed class, it is only at code level. The WSDL still have to redefine carefully. They cannot leverage existed management services.
Interoperability with WSDM clients	Yes	Yes
WSDL file	Yes	Yes
Integration with IDE	Yes. It is easy. Developing management code is the same as general Java program.	Yes. However, the process is complex. Programmers have to understand the procedures to create WSDL and generate the program code.
Effort after code generated	No. The management framework will expose Web services interfaces and link the interfaces with program logic automatically.	Yes. Programmers have to understand the generated code and insert the management logic.
Effort to change interfaces	Only modify code. Others related files are generated.	Both WSDL and management code are required to modify.
Independent of underlying implementation	No. The implementation will affect the management systems.	Yes. Programmers can choose and implement their own implementations.

In the comparison table, the Bottom-up approach is quite dependent of the underlying implementation. In other words, programmers cannot choose program language to implement the management services. In enterprise, it is necessary. Enterprise system should follow similar architecture and program language to implement all IT systems. Therefore, the implementation and maintenance effort can be saved.

Unlike business services, many Web services need several common management functions and some specified management features. For the reason, a re-usable management framework aids development. Besides, WSDL is really a complex file. It is not suitable for human reading but for computer. Without proper tool supported, the file will waste programmers a lot of time. The ICapability interface helps to define the interfaces in Java code. The redundant parts are omitted. Therefore, programmers can focus on management functions and shorten development time.

6.2. RUN-TIME OVERHEAD

Because management functions do not aid business operations directly, it is better that those management functions do not affect system performance too much. To evaluate run-time overhead of the management system, it is necessary to discuss Hook API overhead and architecture overhead. The Hook API overhead shows the costs to record management information of every invocation and add message flow ID in SOAP message. The architecture overhead evaluate the extra loading after adding those management components into the enterprise Web services environment.

6.2.1. HOOK API OVERHEAD

When invoking business services, those components in the management architecture are not involved, besides HOOK API. However, the logic of HOOK API is quite simple.

(1) Request Handler: it extracts and logs id, time_sent, time_resp from SOAP message header

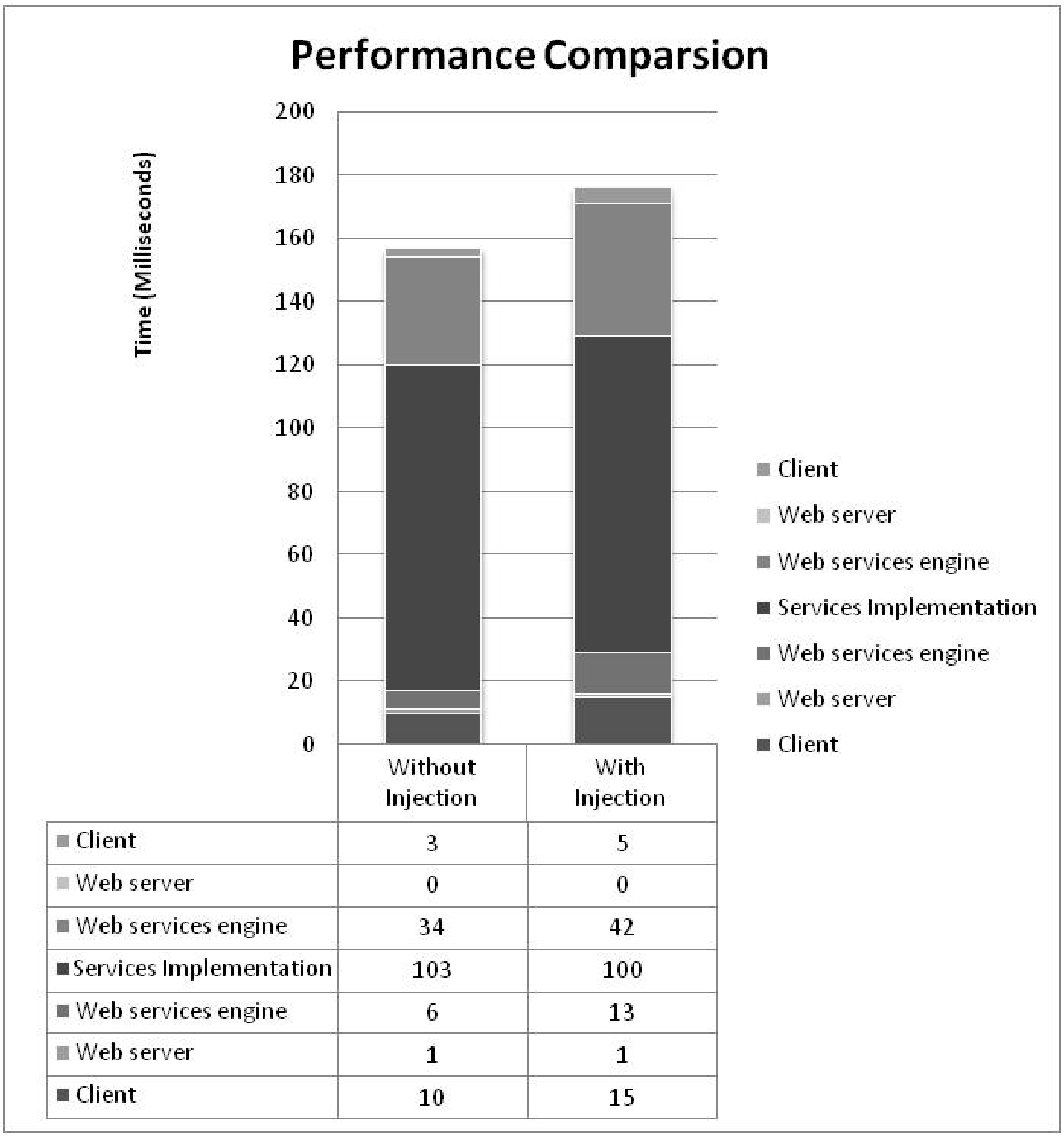
(2) Invocation Handler: it gets message flow ID from local memory and storage as well as inserts id source, time_sent, time_resp into SOAP message header.

Although the two actions are token every services invocation, their operations are quite simple and the program code size is small. To know the influence of Hook API for Web services, the testing environment is built as Table 6-2.

Table 6-2 Hook API Testing Environment

Hardware	CPU	Intel Pentium-M 1.5GHz
	RAM	1.50GB (Using 700MB during test)
Software	Operation System	Microsoft Windows XP sp2
	Java	Sun JDK 1.5.0_12
	Web Server	Apache Tomcat 5.5.23
	Web Services Engine	Apache AXIS2 1.1

The performance test will record the total time from sending request to receiving response. Several parts of the invocation time are recorded. They are the time from client sending request to Web server getting it, Web server process time, Web service engine process time and services implementation process time. The general case is the service without injection; on the other hand, the experiment case is the service with Hook API (injection). Figure 6-1 shows the result.



	Without Injection	With Injection
Client	3	5
Web server	0	0
Web services engine	34	42
Services Implementation	103	100
Web services engine	6	13
Web server	1	1
Client	10	15

Figure 6-1 injection Overhead comparison

In the experiment, service with injection spends more time than without injection. During service requesting, with injection service needs more 7 ms to call the injection program and process the message flow ID from SOAP API. Request Handler is called and extra message flow ID from SOAP header as well as inert it into local message flow ID Mapping Table. Then, it will remove message flow ID from SOAP header. SOAP message at this phase is processed and create object models in memory. For the reason, removing message flow ID from SOAP header will be very quickly.

During service responding, with injection service needs more 8 ms to call the injection program and insert message flow ID into SOAP header. Invocation Handler is used. It will lookup message flow ID from Message Flow ID Mapping Table and insert into SOAP header. The insertion needs more time than deletion because insertion has to create a new XML object and its hierarchy.

Although service with injection takes more time, the different is very limited. It only needs extra 20ms to call injection program and process message flow ID. While a simple Web services program takes around 100ms. If the business functions become more complex, it will take more time. 20 ms will be small and not influence overall system performance.

6.2.2. ARCHITECTURE OVERHEAD

Although many components are added in the architecture, most of they do not execute when a client requests business services. As discussed in Section 3.3, all Web services operations can be separated as two categories: business services and technical operations. Those management components are belonged to technical operations. They will execute only when a client requests management functions. Figure 6-2 shows the relationship between business services and management services. The container will load and execute Web services. There are two containers in the architecture. Business Services Container is response for business services, and Management Services Container is response for management functions.

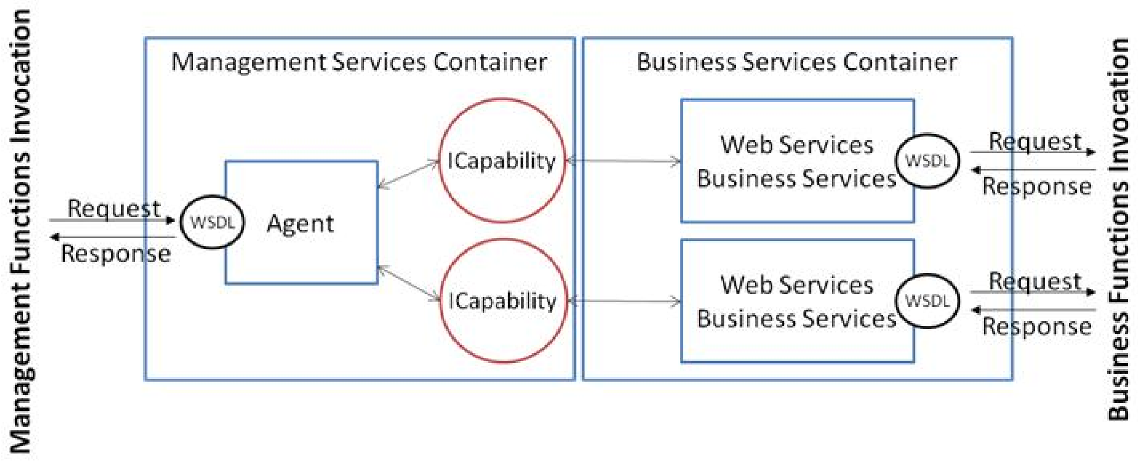

Management Functions Invocation
Management Services Container
Request
Response
WSDL
Agent
ICapability
ICapability
Business Services Container
Web Services Business Services
WSDL
Request
Response
Web Services Business Services
WSDL
Request
Response
Business Functions Invocation

Figure 6-2 Two Different Services Containers for Business Services and Technical Operations

Separated management functions from business services container has many advantages. The execution priority of management services container can be lower than business services container. In this case, those business services have priority. They will execute well and the management services will not race resources. The architecture can guarantee the quality of service.

6.3. Summary of Evaluation

In the management system, both programming effort and run-time overhead are considered. For programmers, they do not touch the complex WSDL file and the generated program code. They will focuses on management logic. Besides, the program logic can be reused. It helps management system with consistent management interfaces. On the other hands, the management logic can be integrated with other management technology, like Automatic Computing. The framework really save development efforts.

For system run-time, the overhead is small and adjustable. Although Hook API is involved in every service invocation, the overhead is quite small. Although many components are added, their executing priority can be adjustable. Therefore, the business operations have priority to complete request on time.

7. CONCLUSION & FUTURE WORKS

7.1. CONCLUSION

Web services management has became an important system in enterprise, especially Web services technology is more and more popular. The management system, proposed in this research, not only focuses on single service but also monitor overall enterprise web services environment. It creates development and run-time architecture to manage a service. The management architecture follows WSDM standards so that WSDM applications can manage them directly. Developing those management functions does not have to understand all WSDM specifications. Programmers only write the management code. The management system will expose them as Web services automatically.

Based on the architecture, the message flow in the services environment can be monitored. Managers can know status for every message so that they can adjust business process or improve system performance to enhance business operations.

With the management system, the enterprise Web services environment will have a complete management mechanism. Not only single Web service is managed, but also all message flow is under controlled. The environment will give corporate better support.

7.2. FUTURE WORKS

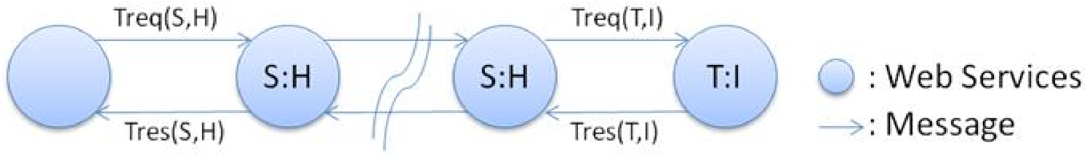


Figure 7-1 Disconnected Web Services Processes

Message tracking mechanism is an important features in the management system. However, some Web service clients cannot send out message with the ID, showed as Figure 7-1. The oblique lines denote messages without ID. In this case, there will be two message flow IDs in this process. It usually happens for the two reasons:

(1) The service client forgets to add the message flow ID into the message.

(2) The service client cannot find the message flow ID. When the service uses the functions those are outside the current process or thread, it will happen.

Under such situation, the request and response time of a services helps to identify their relationship. Let Treq(x,y) and Tres(x,y) denote the request and response time of service x on host y, respectively. If a service S on host H receives message A and sends message B to host I, message A and B are in the same process only when Treq(A,H) < Treq(B,H) and Tres(A,H) > Tres(B,h). In other words, the interval of message A should be larger than message B. However, even the inequality is satisfied, the two messages may not in the same process. They may be belonged to two different processes. Finding more methods to link those Web services processes will make the tracking functions more complete.

On the other hand, the management system focuses on service side management. It collects information from services and provides standard interfaces for management applications to access. Management applications will analyze and present the management information to administrators. In the next step, we will develop a management application. It will monitor services activities and message flows. It also can use properties to configure services. Besides, it can also show service bottleneck. Therefore, the whole IT system can be reviewed quickly and overall performance can be improved.

REFERENCES

[1] Internet Engineering Task Force (IETF), J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin, RFC1157, Simple Network Management Protocol., May 1990.

[2] Distributed Management Task Force (DMTF), Desktop Management Interface (DMI) Standards, http://www.dmtf.org/standards/standard_dmi.php

[3] Distributed Management Task Force (DMTF), Common Information Model (CIM) Standards, http://www.dmtf.org/standards/standard_cim.php

[4] Distributed Management Task Force (DMTF), CIM Operations over HTTP, http://www.dmtf.org/standards/standard_wbem.php

[5] Distributed Management Task Force(DMTF), Web-Based Enterprise Management (WBEM), http://www.dmtf.org/standards/wbem/

[6] Organization for the Advancement of Structured Information Systems (OSAIS), Web Services Distributed Management (WSDM), http://www.oasis-open.org/specs/index.php#wsdmv1.1

[7] World-Wide Web Consortium (W3C), eXtensible Markup Language (XML), http://www.w3.org/XML/Core#Publications

[8] Object Management Group (OMG), Common Object Request Broker Architecture (CORBA), http://www.omg.org/technology/documents/corba_spec_catalog.htm

[9] M. Papazoglou and D. Georgakapoulos. “Service oriented computing”. Communications of the ACM, 46(10):24–28, October 2003.

[10] World-Wide Web Consortium (W3C), Web Services Architecture, http://www.w3.org/TR/ws-arch

[11] Aphrodite Tsalgatidou , Thomi Pilioura, "An Overview of Standards and Related Technology in Web Services", Distributed and Parallel Databases, v.12 n.2-3, p.135-162, September-November 2002

[12] World-Wide Web Consortium (W3C), Web Services, http://www.w3.org/2002/ws/

[13] World-Wide Web Consortium (W3C), XML Schema, http://www.w3.org/XML/Schema

[14] World-Wide Web Consortium (W3C), Simple Object Access Protocol (SOAP), http://www.w3.org/2000/xp/Group/

[15] World-Wide Web Consortium (W3C), Web Services Description Language (WSDL), http;//www.w3.org/TR/wsdl

[16] Organization for the Advancement of Structured Information Systems (OSAIS), http://www.oasis-open.org/home/index.php

[17] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, and S. Weerawarana, “Business Process Execution Language for Web Services (BPEL4WS) 1.0,” August 2002, http://www.ibm.com/developerworks/library/ws-bpel

[18] Liangzhao Zeng , Boualem Benatallah , Marlon Dumas , Jayant Kalagnanam , Quan Z. Sheng, “Quality driven web services composition”, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary

[19] Ruey-Shyang Wu, and Shyan-Ming Yuan, “Enable Transaction in Web Services Environment”, International Conference on Next Generation Web Services Practices (NWeSP'05), Korea, 2005

[20] M. Turner, D. Budgen, and P. Brereton, "Turning Software into a Service," Computer, vol. 36, no. 10, Oct. 2003, pp. 38-44.

[21] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. "Importing the Semantic Web in UDDI." Proceedings of E-Services and the Semantic Web Workshop, 2002.

[22] Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D. , McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., "DAML-S: Web Service Description for the Semantic Web," Proc. Int'l Semantic Web Conf. (ISWC), LNCS 2342, Springer Verlag, 2002, pp. 348-363.

[23] L Aversano, M Tortorella, “An assessment strategy for identifying legacy system evolution requirements in eBusiness context”, Journal of Software Maintenance and Evolution, Vol. 16, NO. 4, 2004, pp.255-276

[24] Harry M. Sneed, "Integrating legacy Software into a Service-oriented Architecture," csmr, pp. 3-14, Conference on Software Maintenance and Reengineering (CSMR'06), 2006

[25] Kazutoshi Yokoyama, Eiji Yoshida and Shigeyuki Matsuda, “Requirements for Open Service Collaboration among Web Services”, Proceedings of the 2002 Symposium on Applications and the Internet (SAINT’02), 2002.

[26] George Pavlou, Paris Flegkas, Stelios Gooveris, and Antonio Liotta, “On Management Technologiaes and the Potential of Web Services”, IEEE Communications Magazine, July 2004

[27] Ruey-Shyang Wu, Fengyi Lin, Shyan-Ming Yuan, and Kai-Chih Liang, “A Reference Model and Integration Framework for Building Enterprise Computing Platform”, International Journal of Technology Management , Vol. 38, No. 4, pp.439-462, 2007

[28] Ruey-Shyang Wu, Shyan-Ming Yuan, Anderson Liang and Daphne Chyan, “iCell:

Integration Unit in Enterprise Cooperative Environment", The Second International Workshop on Grid and Cooperative Computing (GCC2003), Shanghai, China, 2003

[29] AmberPoint Comprehensive Web Services Management, http://www.amberpoint.com/

[30] CA Unicenter product, http://www.ca.com/us/products/default.aspx

[31] "UDDI.org", http://www.uddi.org

[32] N. Catania, P. Kumar, B. Murray, H. Pourhedari, W.Vambenepe, K. Wurster, "Web Services Management Framework, Version 2.0", Hewlett-Packard, http://devresource.hp.com/drc/specifications/wsmf/WSMF-WSM.jsp, July 16, 2003.

[33] IBM Tivoli Software, http://www-306.ibm.com/software/tivoli/

[34] Java Management Extensions (JMX), http://java.sun.com/products/JavaManagement/

[35] Ping-Jer Yeh, Ruey-Shyang Wu, and Shyan-Ming Yuan,"Distributed System Management through Automated Injection", WSEAS Transactions on Business and Economics, Issue 4, Vol. 3 , pp. 276-283, April 2006

[36] Andrea WESTERINEN and Winston BUMPUS," The Continuing Evolution of Distributed Systems Management", IEICE TRANS. INF. & SYST., Vol. E86-D, NO.11 November 2003

[37] Verheecke, B., Cibran, M. A. and Jonckers, V. ,"AOP for Dynamic Configuration and Management of Web services in Client-Applications". In Proceedings of 2003 International Conference on Web Services. Erfurt, Germany, September 2003.

[38] B. Verheecke, M. Cibrn, and V. Jonckers., "Aspect-oriented programming for dynamic web service monitoring and selection." In The European Conference on Web Services 2004 (ECOWS'04), Erfurt, Germany, Sept. 2004.

[39] M. A. Cibran, B. Verheecke, D. Suvee, W. Vanderperren, and V. Jonckers. “Automatic service discovery and integration using semantic descriptions in the web services management layer.” In Third Nordic Conference on Web Services, Nov. 2004.

[40] Y. Eterovic, J. M. Murillo, and K. Palma., “Managing components adaptation using aspect oriented techniques.” In First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT04, held in conjunction with ECOOP 2004), June 2004.

[41] Florian Rosenberg, Schahram Dustdar, "Towards a Distributed Service-Oriented Business Rules System," ecows, pp. 14-24, Third European Conference on Web Services (ECOWS'05), 2005

[42] Sahai A, Ouyang J, Machiraju V, Wurster K. ,”Message Tracking in SOAP-Based Web Services.” In the proceedings of NOMS 2002 33-51., Italy, 2002

[43] Esfandiari, B., Tosic, V.,"Requirements for Web Service Composition Management." In Proc. of the 11th HP-OVUA Workshop. (Paris, France, July 2004). Hewlett-Packard, 2004

[44] R. Berbner, T. Grollius, N. Repp, O. Heckmann, E. Ortner, and R. Steinmetz, "An approach for the Management of Service-oriented Architecture (SoA) based Application Systems", Enterprise Modelling and Information Systems Architectures (EMISA 2005), Klagenfurt, Austria, 2005.

[45] Fabio Casati , Eric Shan , Umeshwar Dayal , Ming-Chien Shan, "Business-oriented management of Web services", Communications of the ACM, v.46 n.10, October 2003

[46] J.A. Schey, Introduction to Manufacturing Processes, 2nd ed., McGraw-Hill, NY

[47] Dirk Krafzig , Karl Banke , Dirk Slama, Enterprise SOA: Service-Oriented

Architecture Best Practices (The Coad Series), Prentice Hall PTR, Upper Saddle River, NJ, 2004

[48] Wong-Bushby, I., Egan, R. & Isaacson, C., "A case study in SOA and re-architecture at company ABC", in Proceedings of the 39th Hawaii International Conference on Systems Science (HICSS-39 2006), Kauai, HI, USA, 4-7 January 2006,

[49] B. Esfandiari and V. Tosic, "Requirements for Web Service Composition Management", 11th HP OpenView University Association Workshop (HP-OVUA 2004), Paris, France, 2004.

[50] Therani Madhusudan, “A web services framework for distributed model management”, Information Systems Frontier, 9(1), 9–27., 2007

[51] Qi Fei and David L. Olson, “Web services composition strategy in enterprise systems”, Human Systems Management, Volume 26, Number 1, 53-61, 2007