國立交通大學

資訊科學系

碩 士 論 文

Web Services 環境中之交易架構

Transaction Architecture in Web Services Environment

研 究 生：洪崇凱

指導教授：袁賢銘　教授

中 華 民 國 九 十 三 年 六 月

Web Services 環境中之交易架構

研究生：洪崇凱　　　　　　　　　　　　　　指導教授：袁賢銘 教授

國立交通大學資訊科學研究所

摘要

Web Services 技術為企業環境整合提供了一項新的技術，以標準的 WSDL 服務介面以及 XML SOAP 訊息傳輸技術使企業應用程式整合更為容易。Web Services 可以讓不同的企業針對內部的流程，提供特定種類的服務與內容。

但是，現今的 Web Services 技術尚未完全解決企業間流程整合的問題。它沒有處理企業間交易行為的機制，只能針對服務分別處理，不能將數個服務視為單一服務，確保這些服務要不就全都成功否則全部回復到交易開始前的狀態。雖然目前已經有許多分散式交易的機制存在，但 Web Services 上的交易方式因為與這些交易的模式不盡相同，所以無法完全地整合在一起；這限制了 Web Services 的應用範圍。Web Services 環境中需要介於應用程式間的協調機制來達成傳統交易所達成的整體系統環境狀態，而且需要更有彈性的方式來進行，不遵守嚴格的 ACID 屬性。

目前在 Web Services 企業流程的交易機制尚未有完整的標準及架構，僅存在眾多不同版本的規格，如 OASIS BTP 、WS-C/AT/BA OASIS WS-CAF 等。分散式交易系統需要長時間、非同步並且牽涉到許多屬於不同範圍及組織的應用程式，這些是與傳統交易系統截然不同之處；Web Services 的訊息交換即可提供前者所描述的機制。本篇論文將提出一個 Web Services 環境中的交易架構，並介紹實作之系統，同時討論相關之規格研究及比較。

Transaction Architecture in Web Services Environment

Student: Chung-Kai Hong Advisor: Dr. Shyan-Ming Yuan

Department of Computer and Information Science

National Chiao Tung University

Abstract

Web Services provides a neo-technology for the integration of enterprise application environment. Enterprise application integration has become easier by using Web Services Description Language (WSDL) and XML Simple Object Access Protocol (SOAP) messaging technology. Web Services enables various enterprises provide specific types of service and content according to their inner business process flow.

However, Web Services hasn’t resolved the problems deriving from business process integration. Web Services doesn’t handle the transaction behaviors among different organizations and make each service function separately; it can’t make multiple services seem to be a single service and ensure the all-or-nothing logic among them. Currently, there have been a lot of distributed transaction systems/mechanisms, but none of them can cooperate with Web Services transaction well because of the different transaction models. These constrain the applicability of

Web Services. A coordination mechanism for Web Services among federated applications is needed to achieve maintained environment state in the traditional transactions. Moreover, such transactions need to proceed in a more flexible way and accommodate the relaxation of ACID properties.

Up to the present, however, the specifications and architectures for the Web Services transaction mechanisms are incomplete under Web Services business processes. There are multiple specifications proposed by various vendors, such as OASIS BTP, WS-C/AT/BA, and OASIS WS-CAF. Distributed transactions systems need long-period of time, asynchronous messaging and involve multiple applications belong to different organizations and enterprises. These are entirely different from the traditional transaction systems; Web Services messaging can provide the mechanism here. In this thesis, we propose transaction architecture in Web Services environment and design an implementation system. Furthermore, we'll discuss about the related specifications and comparisons.

ACKNOWLEDGEMENTS

I would like to give my sincere gratitudes to my advisor, Dr Shyan-Ming Yuan, for his suggestion and correction about my research in the past two years. I also thank my seniors, Mr. Ruey-Shyang Wu, who helped me devotedly to find out the solutions on the thesis, and Tzu-Han Kao, who gave me many suggestions in my research. Finally, I appreciate all supports of the members of DCS laboratory.

Besides, I would thank my family and my girl friend, Chiu-Yi Wu. They give me the greatest strength and supports, especially when I felt depressed.

I dedicate this thesis to my parents, all DCS laboratory members and everyone who concerns about me.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Motivation

With the maturity of World Wide Web [1] environment, the transactions proceed through the WWW among enterprises and organizations more frequently. However, different enterprises in different industry have different business culture, different technology, and construct different business process and technology environment. These all complicate the integration of enterprise applications.

The birth of Web Services is a great step in the distributed technologies. Web Services [3] provides a standard interface WSDL (Web Services Description Language) [4] and XML SOAP (Simple Object Access Protocol) [5] messaging for the integration of enterprise application environment. Web Services enables various enterprises provide specific types of service and content according to their inner business process flow.

However, fundamental Web Services hasn't resolved the so-called transaction problems deriving from business process integration. Many Web Services transaction systems are built based on the classic web technologies, such as using HTTP sessions/cookies, writing supported functions inside the application …etc., which are very inflexible.

Currently, there have been a lot of distributed transaction systems/mechanisms, such as OMG Activity Service [24], but none of them can be integrated well with Web Services transaction and thus constrain the applicability of Web Services. Most transaction managers (TMs) [2] are implemented using Flat Transaction model [2], such as TMs based on JTA/JTS [25], running within a enterprise or single trusted domain, and with resources all controlled by a single TM. Nevertheless, it has become an essential condition to enable multiple different participants to run in different security domains and within different business applications. A coordination framework for Web Services among federated applications is needed to achieve maintained environment state of the traditional transactions. Moreover, such transactions need to proceed in a more flexible way and accommodate the relaxation of ACID (Atomicity, Consistency, Durability, Isolation) [2] properties. The transaction of Web Services also has the responsibility to integrate multiple Web Services into a single reliable application. The execution of Web Services application is treated as a series of activities running in different Web Services.

1.2 Objectives

In this thesis, we propose Transaction Architecture in Web Services environment.

We have known that Web Services currently is still lack of the standard of transaction. But why the Web Services transaction is needed? Web Services transaction is different from the traditional distributed transaction processing with the following three conditions. First, participants who are in the execution of Web Services transactions may come from different trusted domain and use heterogeneous application logics, flows, and platforms. Second, the sub-transaction of each participant may execute for various periods of time. Some of them are likely to be very long-lived. And third, the next state of the transaction may be decided while the previous step is finished, but sometimes we can’t just undo or roll back the whole transaction. For these reasons, the transaction architecture we proposed should support the following functionalities:

1) Make transactions loosely-coupled

   Because of the first reason described above, this can make applications need not to be designed for certain participant or transaction system.

2) Use SOAP for asynchronous, firewall-free and reliable messaging

   Because of the first reason described above, participants may be inside a firewall

3) Allow relaxation of ACID properties and design with Nested Transactions model

   Because of the second reason described above, ACID are too critical for business activities, and resources cannot be locked for long duration of time and across enterprises. Nested Transactions model allows relaxation of ACID properties, especially those of atomicity and isolation.

4) Build Compensating Transactions with dynamic invocation

   Because of the third reason described above, compensations are built specific to certain application and involves application logic.

5) Coordinate transactions under applications

   Because of the first reason described above, we divide transaction coordination from the application design and reduce the complexity.

In this thesis, we provide a methodology to fill the gap between design and implementation for Web Services transactions. We bring transaction concepts into the system design phase and reduce the time to write programs of Web Services transaction. Besides, we provide an Application Program Interface for design and implementation Web Services transaction systems. By our efforts, we make it easier to integrate Web Services applications as well.

1.3 Thesis Organization

The thesis is organized as follows. In Chapter2, we introduce our survey on Web Services-related and transaction-related theories and technologies. We also compare other Web Services transaction systems in the sub chapter of related work. In Chapter3, we illustrate the design of system architecture and each component. Chapter4 illustrates the design of transaction flows and transaction model in the system. In Chapter5, it illustrates implementation of system. Finally, Chapter6 present a conclusion and some future works.

CHAPTER 2
BACKGROUNDS

2.1 Backgrounds

In point of distributed systems, the main trends are “leveraging new technologies to solve existing problems” and “enhance the applicability of new technologies.” The issues of Web Services transactions have both properties of above. We introduce theses issues in this chapter and discuss about related works and current progress in industry vendors.

2.1.1 Web Services and Transactions Overview

Traditional transaction processing system are ready to process transactions with ACID model, including those distributed transaction systems using two phase commit protocols. Even so, ACID model is only suitable for the tightly-coupled distributed transaction system, but not workable for long-duration, loosely-coupled, asynchronous transaction systems. Such transaction architecture may need several hours or days to run, but the resources of participants are not allowed to be locked for such long period; when transaction errors occur or cancellations proceed, traditional rollback is not feasible. Therefore, long-lived, loosely-coupled, asynchronous transaction need to coordinate and conclude the transaction agreement at run time and isolation level must be relaxed.

Because Web Services can be accessed by any language, using any component model, and running on any platform, the Web Services transactions faces the problem of long-lived, loosely-coupled, asynchronous transactions. Web Services essentially is a loosely-coupled infrastructure with asynchronous and reliable messaging and dynamic invocation at run-time functionalities. Consequently, we can resolve the transaction problems described above by Web Services technology and at the same make up for the Web Services transactions.

Figure 2.1 shows the general Web Services Transaction architecture.

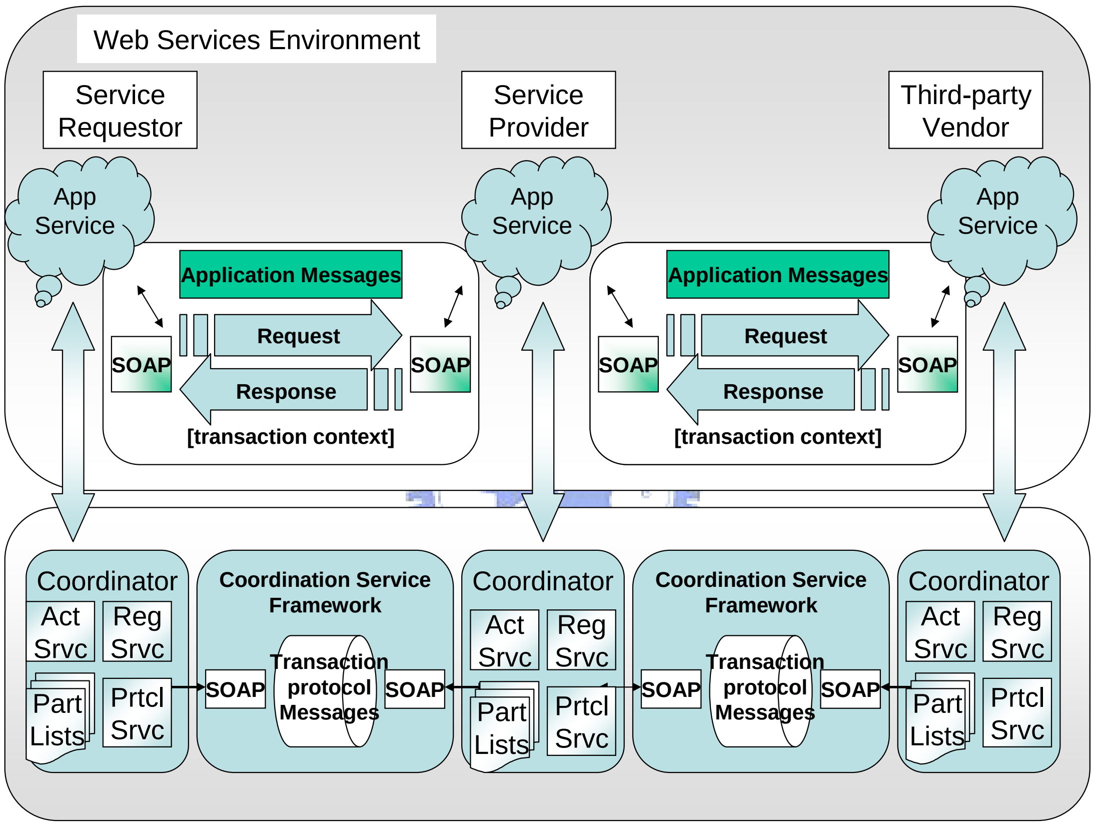


Figure 2-1 General Web Services Transaction Architecture

2.1.2 XML SOAP and WSDL

XML (Extensible Markup language) is a meta-language. It is also a high-structured and verifiable language. XML is as similar as HTML using tag and attribute. The most different between HTML and XML is that the tag and attribute of

XML can be customized by developers. All XML files follow the specification XML 1.0 defined in W3C Recommendation Feb’98. Web Services defines SOAP and WSDL by leveraging the architecture of XML.

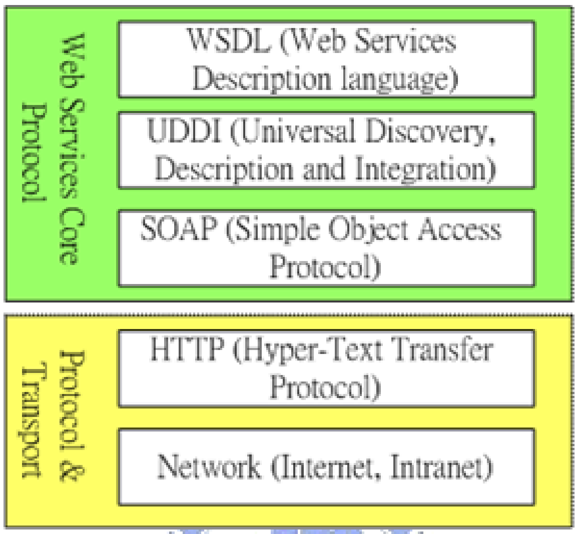

Web Services Core Protocol
WSDL (Web Services Description language)
UDDI (Universal Discovery, Description and Integration)
SOAP (Simple Object Access Protocol)
Protocol & Transport
HTTP (Hyper-Text Transfer Protocol)
Network (Internet, Intranet)

Figure 2-2 Basic Web Services Architecture

SOAP is a light-weight protocol applied on non-decentralized distributed systems. SOAP itself doesn’t define any information about applications; in the contrast, SOAP defines the framework of message exchanging for different application or module to communicate with each other. This protocol is based on XML, including three parts: 1. Envelope: define a framework to describe what parts should be contained and how to process these data. 2. Encoding rules: describe what are the data types used by applications. 3. Convention: present RPC (Remote Procedure Call) and response types. SOAP can be bind to most of transport protocols, and HTTP is now in widespread use. The most important benefit of HTTP protocol here is that it can transmit through firewalls without any obstruction.

WSDL leverages XML to describe the operation methods of a Web Service. WSDL defines some components to describe a service: 1. Type: defines a needed data type. 2. Message: define an abstract data type for transmission. 3. Operation: defines an action that can be achieved by a service. 4. PortType: defines the operations supported by a service. 5. Binding: defines the transport protocol and data types while using this service. 6. Port: defines the combination of the transport protocol and its network address. 7. Service: defines a collection of Web Services. WSDL can describe all related information about a service including how to use this service and what functionality this service provides. WSDL use the standards of HTTP and SOAP. By using WSDL, the operations of services can be known at any time when getting the location of WSDL file, and dynamic invocation at runtime thus can be achieved. Therefore, this becomes the greatest advantage of leveraging WSDL in Transaction Architecture design.

2.1.3 Atomic Transaction and Business Activity

Traditional two-phase atomic transaction is only with respect to the services involved. Atomic transaction simplifies application programming by hiding applications from system-generated exceptions, such as hardware failures or database deadlocks. Systems offering services may announce two-phase commit, but use some other intra-enterprise model like compensation. This freedom makes a simple two-phase commit (2PC) model more useful for long-running Internet computations and this part of application logic could be automated with transaction management.

A business activity uses multiple atomic transactions to move the long-running transaction from one consistent state to next one. Business activities leverage atomic transactions to handle system-generated exceptions while business activities handle application-generated exceptions themselves. Business activities implement long-running, compensation-based transactions and involve application business logics, while atomic transactions provide a way to easily return to consistent state. Therefore, ACID properties for long-lived business activities need to be relaxed.

2.1.4 ACID and Transaction Models

Compensation is based on the idea that a transaction is always allowed to commit, but its effect and actions can be cancelled after it has committed. Cancellation is an example of a compensating transaction. In compensating transaction, each real transaction has an associated compensating action, which describes a way to revert changes done by the real transaction and return to a previous consistent state. If any transaction aborts, the caller executes the corresponding compensating transaction for all the transactions that have previously committed.

As described in previous sub-chapter, a business activity in its nature is a long-running transaction and may have many problems with ACID properties, especially when compensating. In a long-running transaction, some activities may success and some may fail, though, transaction is still completed; we can’t just abort the whole transaction because of some failures, therefore the atomicity need to be relaxed. Consistency must be guaranteed at any time. A business activity typically span several minutes or even days and weeks, so we need to save the state between each steps in order to handle application-generated exceptions. Besides, we should avoid locking resources to maintain durability, but this need to lower the isolation level for releasing resources between steps. In conclusion, A and I need to be relaxed, and C, D are modified to be achieved.

At first, transactions came into use in DBMS (Database Management Systems) for the applicability and consistency of the data. As a result, several transaction models are developed for solving transaction problems, like Flat Transaction, Nested

Transaction and Distributed Transaction models, etc.

Figure 2-3 shows a transition diagram of Flat Transaction in the view of application.

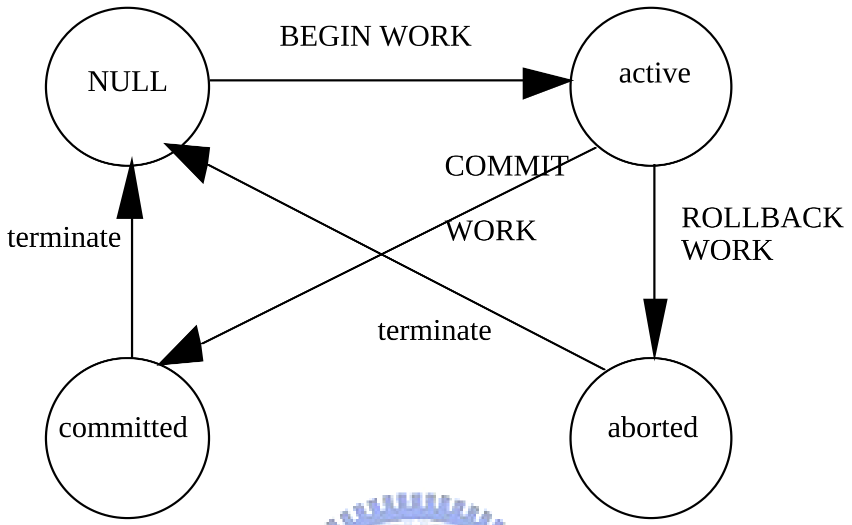


Figure 2-3 State-transition diagram for a Flat transaction as seen by the application

In a Flat Transaction, if we issue a rollback action, which is caused by a single failure. However, we have to gives up much more of the previous work than is necessary and explicitly cancel all the actions that are no longer useful. This is the most troubling thing. Furthermore, explicit cancellation might not be so easy, because some may cost a high cancellation fee. Next comes Chained Transaction, which is a workflow structure enhanced of Flat Transaction.

Nested Transaction is developed afterward. A nested transaction is a tree of transactions, the sub-trees of which are either nested or flat transactions. A sub-transaction's commit will take effect only when the parent transaction commits. The rollback of a transaction in the tree causes all its sub-transaction to roll back. This is the reason why sub-transactions have only ACI, but not D.

As a result of using single systems, like DBMS, ACID properties are guaranteed in traditional transactions. However, we need to consider much more in Distributed transactions. A distributed transaction is a flat transaction running in a distributed environment and guarantees ACID. There are much more uncertainties in the network environment, transaction in distributed environment would be more difficult to process. In order to make transactions in distributed environment more applicable, we apply the nested transaction model to distributed transactions; each transaction in a different node of the distributed environment is a sub-transaction of the nested transaction. This is what we are going to apply in our transaction architecture.

2.2 Related Industry Specifications

In order to build up a demonstration environment, we leverage several existing vendor specifications that are proposed related to transactions in Web Services environment. Before presenting the designed system, introducing these specifications and discuss about other related ones are necessary.

2.2.1 Business Transaction Protocol (BTP)

OASIS (Organization for the Advancement of Structured Information Standards) BTP is proposed by HP (Hewlett-Packard Development Company), Oracle (Oracle Corp.) and BEA (BEA Systems, Inc) for B2B transactions in the loosely-coupled domain, such as Web Services. BTP is not designed specifically for Web Services transactions; at first, its goal is to apply to heterogeneous environments and make the transaction protocol with general XML messaging. After a while, WS-C/AT/BA is designed specifically for Web Services and constructed the foundation of Web Services basic definition. In contrast with

WS-C/AT/BA, BTP is less suitable than WS-C/AT/BA for the Web Services tractions.

2.2.2 WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity

IBM, Microsoft and BEA propose these specifications and devide the coordination framework from the transaction management. WS-Coordination describes how to create, register and coordinate all the transaction activities among multiple Web Services; meanwhile, it provides coordination protocols services, which belongs to different coordination rules, while creating and registering transaction activities. WS-AtomicTrasaction and WS-BusinessActivites is based on foundation of traditional transaction, including functionalities of business logic and non-functional transactions inside applications. In addition, WS-BA defines how to manage the relations of business activities in different scopes and also allow nested scopes.

The Web Service architecture provides a set of modular protocol building blocks that can be composed in varying ways to create protocols to particular applications. The protocols present in WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity are mechanisms to create activities, join into them, and reach common agreement on the outcome of joint operations. These specifications provide a basis on which to build interoperable, distributed applications that desire to coordinate joint work.

2.2.3 WS-Composite Application Framework (WS-CAF)

WS-CAF is jointly proposed by Arjuna (Arjuna Technologies), Fujitsu (Fujitsu Limited), IONA (IONA Technologies), Oracle and Sun (Sun Microsystems) to the OASIS consortium. WS-CAF mainly defines the lack parts of the previous two specifications, and is called the superset of WS-C/AT/BA. It enhances the Context Services by dividing it from the Coordination Framework and provides transaction protocols and model different from those in BTP and WS-C/AT/BA. The design of Context Services here is a very useful framework and will become a reference of our architecture.

CHAPTER 3
TRANSACTION ARCHITECTURE

In terms of architecture design, our goal is to design a transaction architecture based on Web Services, providing basic transaction functionalities; meanwhile, we will provide an implementation design reference for the developers. Moreover, we design a complete system with reference to the existing specifications and supplement those are not strong enough or deficient. According to the comparison of the three main specifications, WS-C/AT/BA are still short in some areas, such as the flow of compensation transactions. WS-C/AT/BA are the main specifications we are going to support in our system, and others specifications are references for the system design as well.

Figure 3.1 shows the transaction work flow.

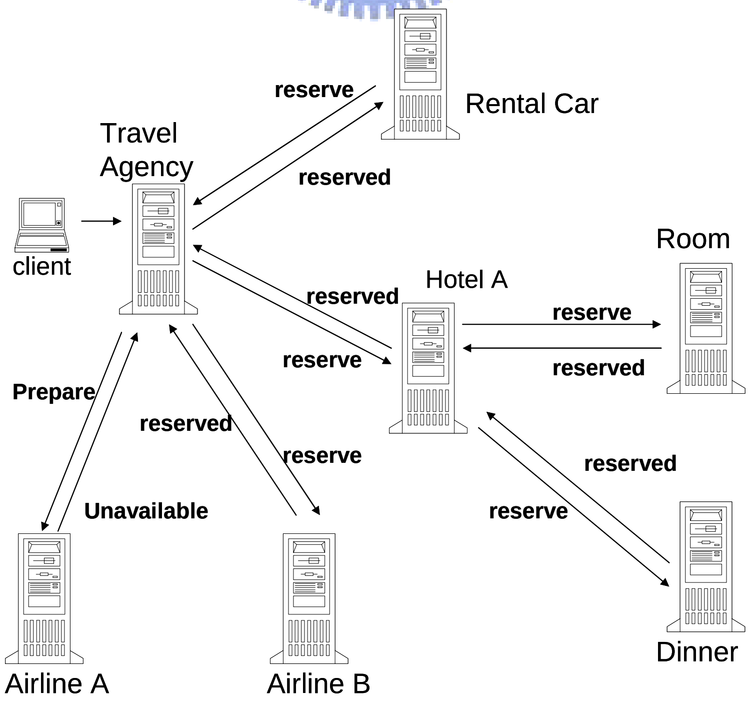


Figure 3-1 Transaction Work Flow

3.1 Web Services Transaction Environment Overview

In the environment of Web Services transaction, user access services through Web Services systems. When a user issues one request, corresponding Web Services begin to create and register a transaction in its coordinator, and pass its transaction context to the associated Web Services for transaction coordination. Because these Web Services are constructed on the basis of our transaction architecture, we call them Transaction-Aware Web Services. We will discuss this later.

The coordinator which creates the coordination context first is called the root coordinator; the other coordinators which join the same transaction are called subordinate coordinators. A coordinator may be a root and a subordinate at the same time. Moreover, transaction themselves may involves in multiple transactions. The transaction at the root is called top-level transaction, the others are called sub-transactions. A transaction's predecessor is called parent; a sub-transaction at the next lower level is also called a child. Such transaction is so-called nested transaction.

On top of the Web Services transaction architecture is the environment we build for transaction coordination, coordinators mainly provides activation service, registration service, and coordination services.

3.2 System Architecture

In point of system architecture, we design a stack as follows. The bottom layer is a Web Application Server built within a XML SOAP Engine, providing an execution environment for Web Services applications. Based on this, we develop some transaction coordinators port types as ActivationService, RegistrationService, RegistrationRequester and all coordinaton services for WS-AT and WS-BA transaction protocols.

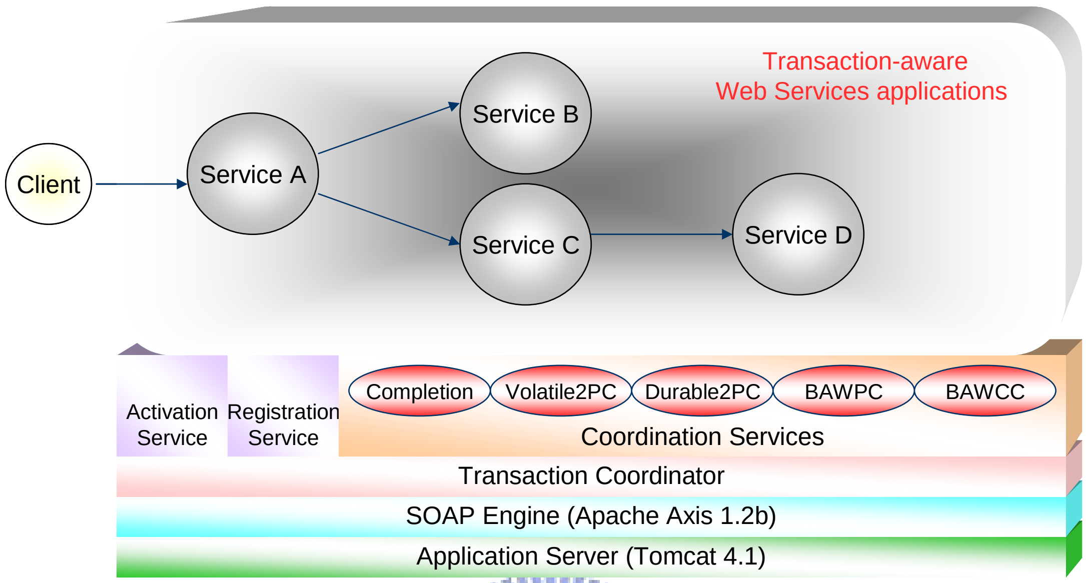


Figure 3-2 System Architecture

On the basis of the Web Services transaction architecture, the functionalities of coordinators are maintaining transaction connections, protocol participants management, protocol subordinators management, protocol services management, transaction state negotiation and protocol messages management.

3.3 Transaction-Aware Web Services

We have mentioned above about Transaction-Aware Web Services, which are Web Services applications defined to run on top of our transaction architecture and provide the functions of atomic transaction or business activity or both. Transaction-Aware Web Services are able to process transaction protocol related messages, and provide ActivationRequester, RegistrationRequester, and WS-AT and

WS-BA protocol participant services port types.

In the environment of Web Services transaction architecture, we divide Transaction-Aware Web Services into two types, while one is atomic transaction applications and the other is business activity applications.

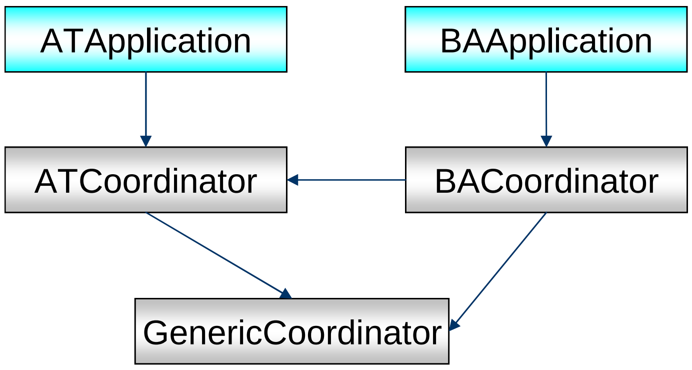


Figure 3-3 Class diagram of Transaction-Aware Web Services

From the Figure 3-3, we can see the class relations of transaction-aware Web Services. We design a base class named GenericCoordinator, which covers all the basic functions that a coordinator should provide, such as creation coordination context and registration services. Both the coordinators of atomic transaction and business activity inherit the GenericCoordinator, and therefore have all the basic functions of generic coordination, called ATCoordinator and BACoordinator respecitvely. Moreover, applications of atomic transaction and business activity as well inherit the ATCoordinator and BACoordinator respectively, and therefore they have built-in coordinator. We will introduce the coordinators and the applications of atomic transaction and business activity in the following two subsections.

3.3.1 Atomic Transaction Application

ATCoordinator is a coordinator which provides generic coordination functions and deals with atomic transaction coordination. Atomic transaction coordinator provides atomic transaction protocols services, such as Completion protocol service, Volatile2PC protocol service and Duable2PC protocol service. By providing these three protocol services, ATCoordinator can be inherited by atomic transaction applications. Atomic transaction application then can provide atomic transaction specific services, such as AT applications designed as a resource manager for a transaction processing monitor or a transaction system.

3.3.2 Business Activity Application

BACoordinator is a coordinator which provides generic coordination functions and deals with business activity coordination and sometimes atomic transaction coordination as well. Business activity coordinator provides business activity protocols services, such as BusinessAgreementWithCoordinatorCompletion protocol service and BusinessAgreementWithParticipantCompletion protocol service. BACoordinator is derived from the GenericCoordinator, but it can be derived from ATCoordinator too to have the atomic transaction coordination functions. By providing these two protocol services, BACoordinator can be inherited by business activity applications. BA application then can provide business activity specific services, such as BA applications designed as a transaction processing monitor, a transaction system or any other Web Services providing transaction services.

3.4 Context Management Framework

In this section, we illustrate about the context management mechanisms designed in our transaction architecture. The term “context” is not well defined in general technical specifications. In brief, context is a way to achieve correlation between different programs, applications or platforms. For example, the cookies used in Web browsers, single session sign-on systems, and transaction systems are examples of sharing common persistent data mechanism. For these reasons and techniques, common documents or temporarily working data can be shared between application programs.

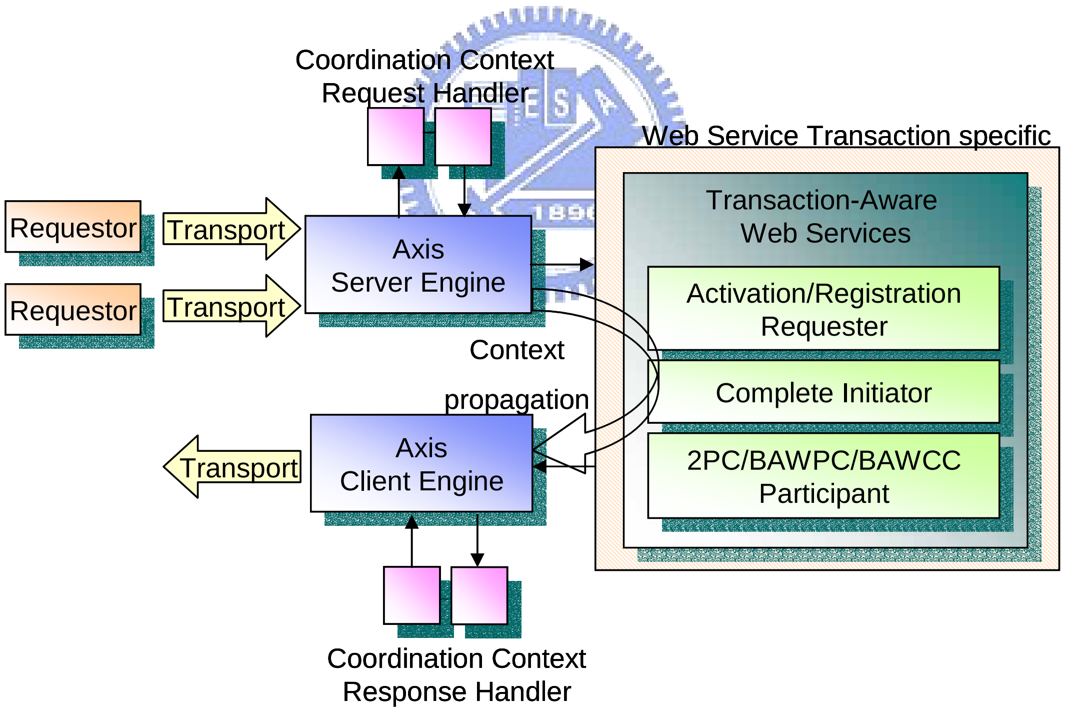


Figure 3-4 Context management framework of Transaction-Aware Web Services

Although WS-C mentions the use of Context in the transaction coordination, it doesn’t describe the usage scenario and where to use very clearly, and neither describes the management and maintenance of context of different activities. In the system we design, we use the message handler chain in SOAP Engine to manage and maintain contexts. As the figure 3-4 shows, we could see that the context management framework is designed to be embedded in the AXIS SOAP Engine. We leverage the SOAP processing mechanism, when the server engine is processing request messages in the request message processing chain, CoordinationContextHandler retrieves the context message shown in the request SOAP headers and store temporarily in the Server Engine; If Transaction-Aware Web Services need to know the context messages, they can get context messages from the Server Engine. Once when Transaction-Aware Web Services need to issue application requests to other Transaction-Aware Web Services, they can put the coordination context into the SOAP Client Engine, CoordinationContextHandler in the request message processing chain of Client Engine will retrieve the context messages which stored in it previously by Transaction-Aware Web Services, and add the context message to the request SOAP header by request message processing chain, and then transmit the request SOAP messages. By this framework we design, context sharing and management are easy and won't take more efforts for Transaction-Aware Web Services.

3.5 Activity State Management

Business activity is a series of business state transition, achieved by designed business functionalities, and sometimes are called business processes. A business process is usually composed by multiple business transactions; the business transactions in the Web Services environment, to be more precisely, are transaction and interaction processes of different organizations for negotiating certain common agreement deals. The outcome of a transaction will be affected consistently by all of the changes of one business state. We must ensure that all of the state transitions history, including applications states and transaction coordination contexts, would be recorded reliably within the processing coordinator and between involving coordinators. Furthermore, the nested message management we used in the transaction architecture makes all messages in different scopes managed by their own root coordinators of the scopes, and decides the correct state and messages of next transition.

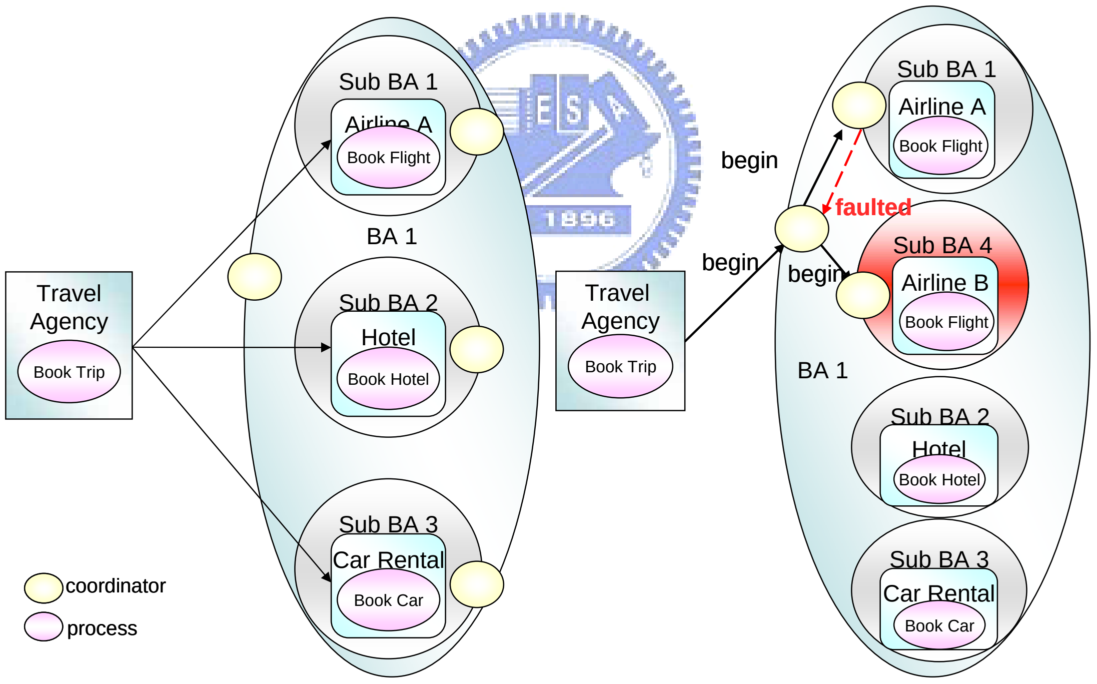


Figure 3-5 Transaction flow control

To give an example, we can see from the figure 3-5, when we are issuing a transaction request over a travel agency Web Services system, the travel agency Web Services then communicates with the other three involving Web Services, airline, hotel and car rent services, by exchanging transaction messages. As described in the paragraph above, all business transaction states are maintained by their own transaction coordinators, and the entire business activity states are maintained by the coordinator of travel agency. When any errors occur in the process of sub-transaction 1, the airline A turns out one failure and then notifies coordinator A an exception. The coordinator A receives this exception and tells the coordinator of travel agency to make him decide the next state of transaction. Later, the root coordinator notifies A to cancel the original sub-transaction 1 and begin to enter compensation state. Afterward, the root coordinator continues to process new compensate sub-transaction of airline reservation and transmit the new protocol message to the airline B. Since then, we can understand a scenario of compensation and error recovery in the activity state management of our transaction architecture.

CHAPTER 4
IMPLEMENTATION ISSUES

In this chapter, we will discuss how transaction architectures are implemented based on Web Services architecture and WS-Transaction specifications. The implementation provides the modularity of system components and a development environment which reduces development time and requires minimum efforts for Transaction-Aware Web Services applications. At the beginning of the system implementation phase, we spend a lot of time on the understanding and familiarity of AXIS architecture, therefore we are going to discuss about the AXIS as well. Furthermore, we will introduce implementation details of each components and related design issues.

4.1 Implementation Overview

Inside our system implementation, we use Java programming language to develop application programs. We use Apache Tomcat 4.1 [29] of Apache Software Foundation as the Application Server and Apache Extensible Interaction System (Apache AXIS) 1.2b [27] as SOAP Engine for processing XML Web Services.

Our architecture design and protocol implementation is based on WS-C/AT/BA. We define all transaction related interface according to WS-C/AT/BA WSDL definitions; XML schemas refer to the definitions of WS-C/AT/BA as well and Java data types are generated by WSDL2Java tool of AXIS. Besides, the idea of context management originates from WS-Context in the specification of WS-CAF; however, we design the context management framework and activity state management as some event handlers in the transaction architecture rather than services.

Because we use the definitions of WSDL PortTypes in WS-C/AT/BA for transaction related program interfaces, our implementation primarily is composed of many Coordinators and Transaction-Aware Web Services. We use a Java class to implement multiple Port Types; leveraging the polymorphism of Java in Web Services implementation makes Web Services implemented by a single class and deployed as multiple Port Types. This implementation provides a flexibility way for Web Services to be dynamically invoked at runtime.

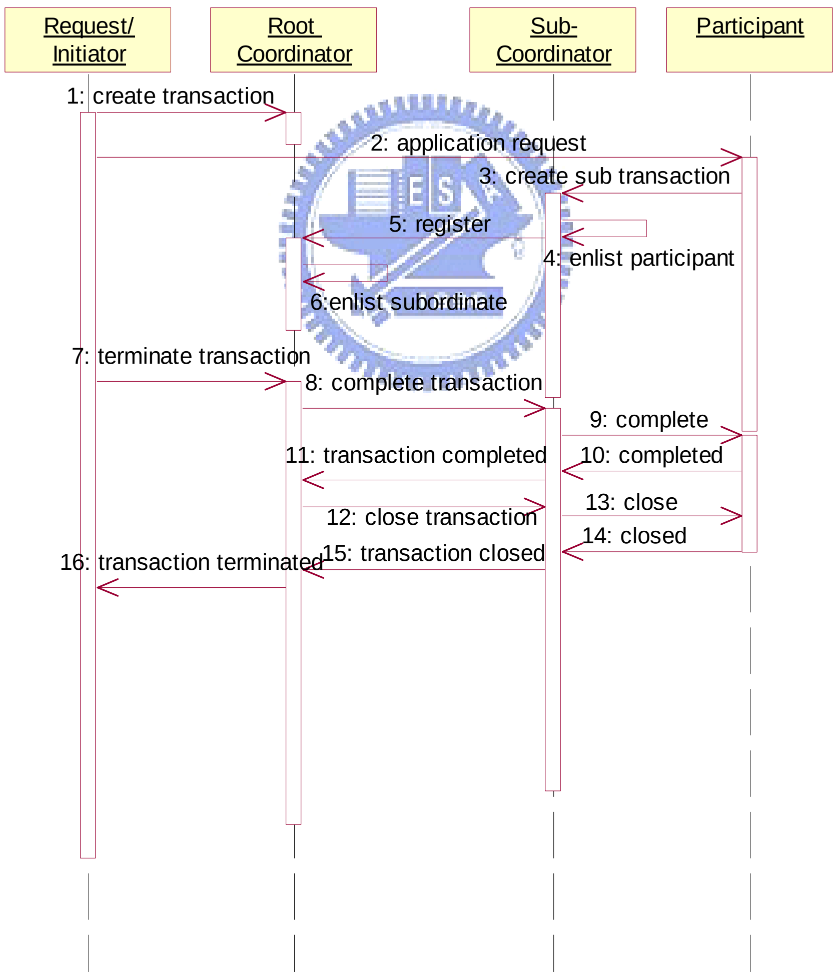


Figure 4-1 Sequence Diagram of Transaction Coordination Protocols

Figure 4-1 shows the sequence diagram when requester or initiator, root coordinator, sub-coordinator and participant are in proceeding of a transaction and the message flows among them. This diagram mainly shows the part of messages of transaction coordination protocols during creation , registration and interposition of an activity.

Figure 4-2 shows the message content when coordinator is processing an atomic transaction, i.e. the coordinator notifies the participants to go to prepare phase and get ready for the commitment.

```
<env:Envelope ...>
  <env:Header>
    <wsa:ReplyTo>http://localhost:8080/Part1</wsa:ReplyTo>
   <wsa:Action>http://schemas.xmlsoap.org/ws/2003/09/wscoor#Prepare</wsa:Action>
   <tx:activityID>urn:3fd333f8-b723-4acd-9d40-16a21fa5d1f8</tx:activityID>
   <tx:scopeID>urn:d1de41df-844f-4c60-b984-2ec5c9b22aba</tx:scopeID>
  </env:Header>
  <env:Body>
    <wsat:Prepare/>
  </env:Body>
</env:Envelope>
```

Figure 4-2 Protocol Message of Atomic Transaction from a Coordinator

4.2 Context Management

We have discussed about the design of context management framework in the previous chapter. However, there are many difficulties to maintain persistent data and correlated information within AXIS architecture. AXIS, as we have known, is an event-driven XML SOAP processing engine for Web Services applications; the design of AXIS is based on Tomcat servlet engine, which is a web application container and doesn't have any persistent storage for certain web applications to maintain their data. Nevertheless, Web Services are only able to put and get data elements as objects to and from the SOAP message bodies, but have difficulties to get and put data elements as objects to and from the SOAP message headers. Therefore, how to store and manage context information becomes an important issue within the implementation system. Axis is organized with a series of Handlers; MessageContext object, which contains request and response messages and some associated properties, will be passed to each Handler invocation.

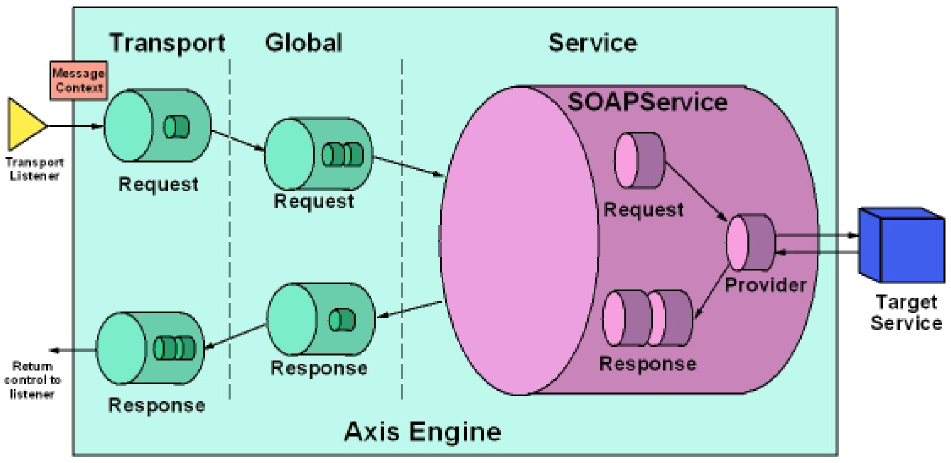
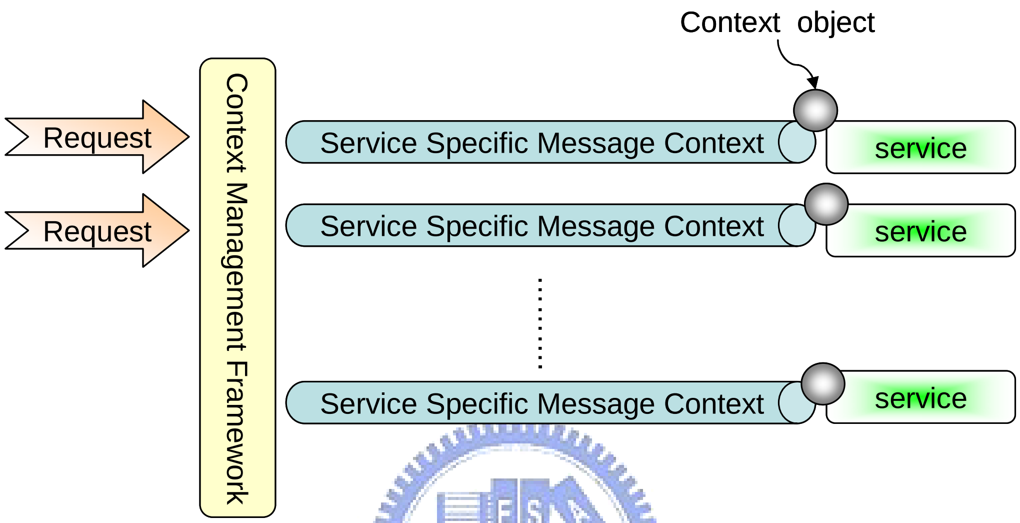


Figure 4-3 Server Message Path of AXIS Engine

source: ws.apache.com/axis/

The message path of the server is shown in the figure 4-3; a Transport Listener will create a MessageContext object and invoke the Axis framework, and the target Web Service is also a handler of the processing chain. Therefore, we can design one CoordinationContext Request Handler within the Server request chain to retrieve the context header from the request message and store it into the MessageContext object; once when the MessageContext object is passed to the target services, the target services can get the CoordinationContext object from the MessageContext at any time without extra efforts for processing the SOAP headers by themselves and context header messages won't be lost during the processing time. The figure 4-4 shows the Transaction Context Management Framework of the server-side.



Context object
Request
Request
Context Management Framework
Service Specific Message Context
service
Service Specific Message Context
service
Service Specific Message Context
service

Figure 4-4 Server-side Transaction Context Management Framework

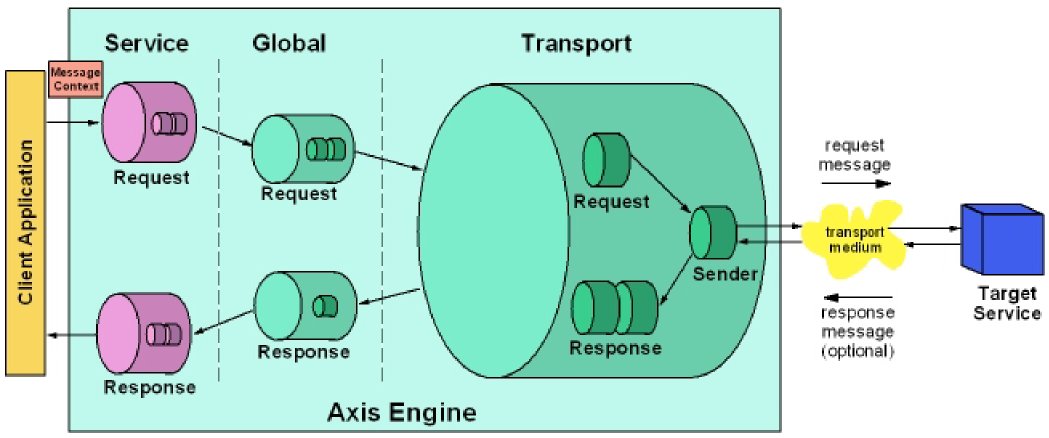


Figure 4-5 Client Message Path of AXIS Engine

source: ws.apache.com/axis/

The figure 4-5 shows the message path of the client; application code of a client will generate a MessageContext object and invoke the Axis processing framework, and the target Web Service is also a handler of the processing chain. Therefore, we can design another CoordinationContext Request Handler within the Client request chain to get the CoordinationContext object from the MessageContext object if exists any and turn the CoordinationContext object into a SOAP header element in the SOAP request message; if client application puts a CoordinationContext object into the Client engine of Axis and issues any requests to Transaction-Aware Web Services, the CoordinationContext object will be processed by CoordinationContextHanlder as described above; the client applications can put the CoordinationContext object into the request message headers freely without processing the SOAP headers by themselves and context objects won’t be lost during the processing time. The figure 4-5 shows the Transaction Context Management Framework of the client-side.

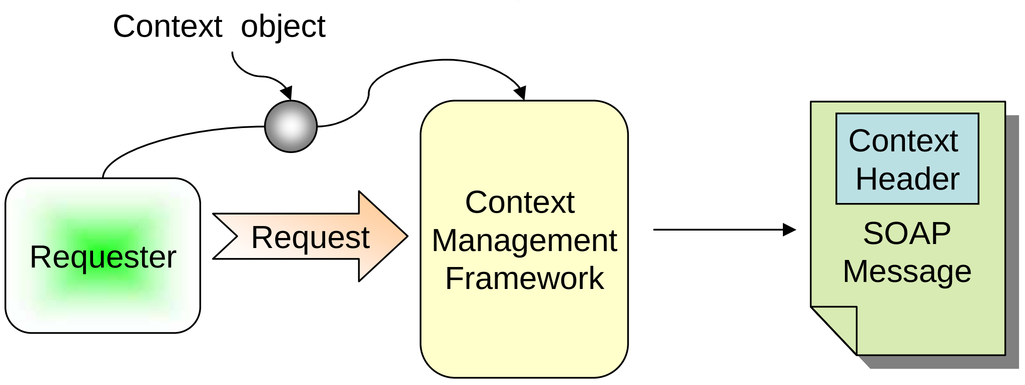


Figure 4-6 Client-side Transaction Context Management Framework

Since then, we are finally able to provide a context propagation mechanism for the transaction architecture to maintain and manage persistent and correlated data among those many multiple Transaction-Aware Web Services.

4.3 Participant Lists Management

Since each joiner, transaction, and sub-transaction has an individual identifier, we need to manage them among the coordinators and record those identifiers respectively and separately for each protocol, in case that any exception and state transition of the transaction occurs.

The figure 4-7 shows the entire participant lists maps existing in atomic transaction and business activity coordinators. ATCoordinator maintains the Volatile2PC and Durable2PC protocol participant lists maps while BACoordinator maintains the BusinessAgreementWithCoordinatorCompletion and BusinessAgreementWithParticipantCompletion protocol participant lists maps. Completion protocol doesn't need a participant lists map because there is only one initiator as participant role.

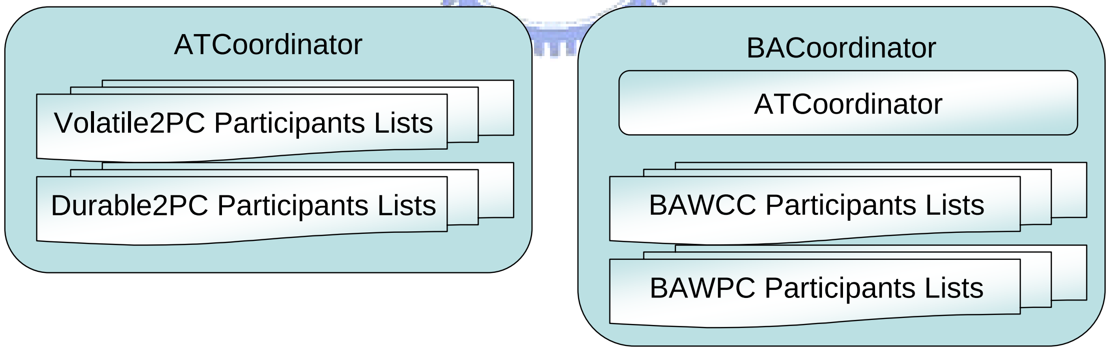


Figure 4-7 Protocol Participants Lists of Transaction Coordinators

We record the transaction identifier as the key and the joined participants' address list as the value of the key/value pair in the map for all protocols respectively. When a coordinator receives a register request after authentication, it get the list of the transaction according to the transaction identifier, which is get from the context object in the request header; if the list not exists in the coordinator, create a new one, and then store the port references, i.e. the address of the participants, into the list and put the list back to that protocol lists map.

4.4 Run-Time Scenario

The run time scenario of the transaction architecture will be described as follows. When a client is going to issue a transaction request to a transaction-aware Web Service, it first create a coordination context with its coordinator and get a coordination context object returned; next, the client issues a application request with a coordination context to a transaction-aware Web Service. On receiving the request with a context header, the target services begin to interpose itself as a subordinate of the created transaction and register to the previous coordinator for its root coordinator. After these creation and registration flows, transaction protocol messages are then able to be transmitted and form a protocol instance.

The figure 4-8 shows the creation and delegation scenario of a transaction activity.

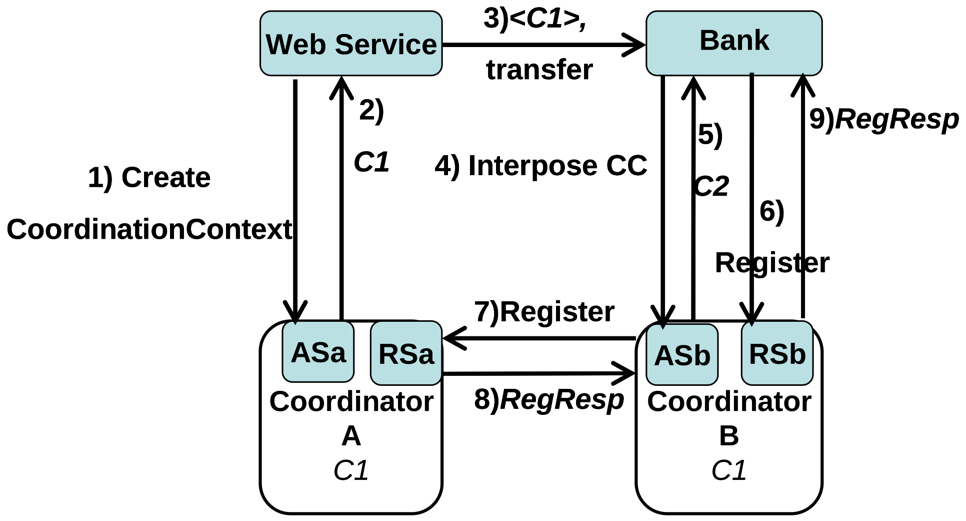


Figure 4-8 Creation and Delegation of a Transaction Activity

When an atomic transaction is being processed between coordinators and transaction-aware Web Services, there will be three kinds of participants: Completion initiator, Volatile2PC participants and Durable2PC participants. If all processes of each participants are finished, the Completion initiator, i.e. the application, begin to commit the atomic transaction activity and send a commit message to the coordinator. Then the root coordinator starts the prepare phase of the 2PC protocol and send a prepare message to each Volatile2PC participants; after each Volatile2PC participant returns a prepared message, the coordinator sends a prepare message to all Durable2PC participants. When all prepared messages are returned from the Durable2PC participants, the coordinator decides to commit the transaction and sends committed message to the completion initiator and commit messages to all Volatile2PC and Durable2PC participants at the same time.

The figure 4-9 illustrates the two phase commit protocol as a state transition diagram.

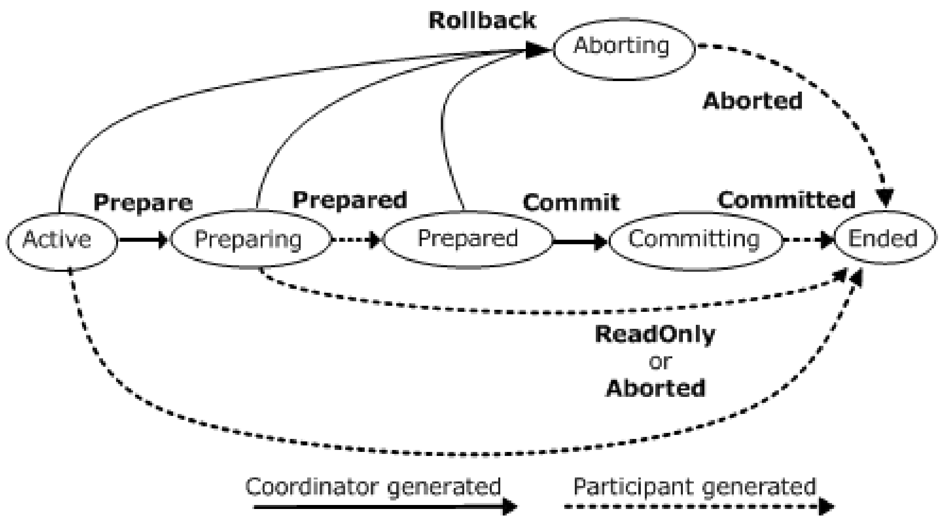


Figure 4-9 Two Phase Commit Protocol

source: msdn.microsoft.com

When a business activity is being processed between coordinators and transaction-aware Web Services, there will be two kinds of participants: BusinessAgreementWithCoordinatorCompletion participants and BusinessAgreementWithParticipantCompletion participants. The previous ones need the coordinator to tell them when to complete their works while the second ones decide when to complete their works by themselves. When BAWPC participants finish their works, they signal the coordinator by sending a completed message and enter the competed state; when BAWCC participants finish their works, they wait for the coordinator's complete message, and then complete their processes and signal the coordinator by sending a completed message and enter the competed state. If all processes of each participants are finished, the root coordinator decide to close the transaction and notifies all participants with the close message. If any exception or errors occur within a participant, the root coordinator will decide to begin a compensation activity and notify the participant with a compensate message; meanwhile, the root coordinator initiates another sub-transaction to replace the original work of the compensated participant. Sometimes the compensation takes more time and money to complete.

The figure 4-10 illustrates the business agreement with coordinator Completion protocol as a state transition diagram.

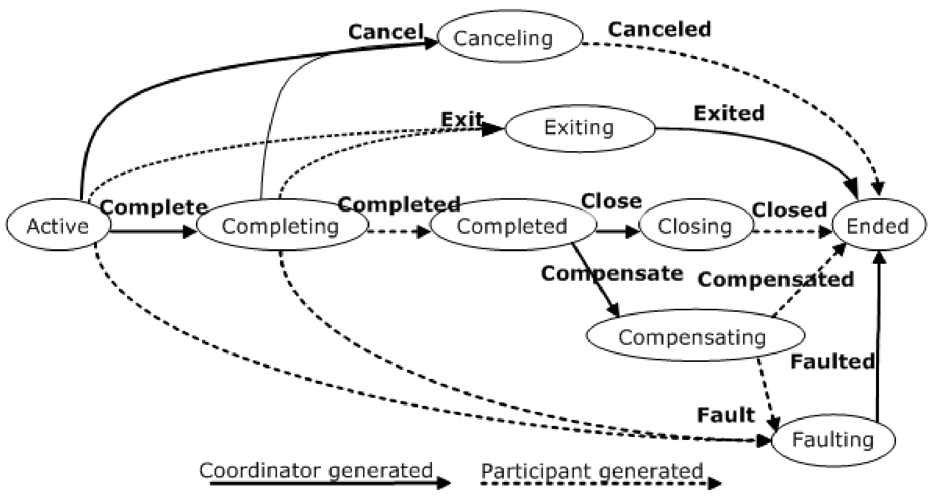


Figure 4-10 Business Agreement With Coordinator Completion Protocol

source: msdn.microsoft.com

4.5 Transaction Application Development Model

Dynamic service invocation is an important technique in Web Services architecture. However, not all Web Services applications achieve this characteristic, especially in Web Services transaction systems. We design the transaction architecture to provide an environment for transaction system developers to make transaction-aware Web Services dynamically invoked. The PortTypes conventions defined in the WSDL of WS-C/AT/BA help us to achieve this goal; we design all transaction protocols and interfaces according to the specification and thus make all protocol messages exchanged among specified interfaces.

With the architecture we implement, developers of transaction-aware Web Services can develop and deploy Web Services transaction systems easily and fast for the following reasons: 1) we design the class hierarchy as a single-rooted architecture; all classes in the architecture inherit the "Coordinator" class; 2) one single class implementing multiple interfaces (PortTypes) can make Web Services application deployment in the Axis architecture faster, easier and multi-deployable;

3) one application code can be multiple port types and doesn’t have to write different classes for different port types; this saves development time and application protocols between different application codes within a single system; 4) services are running in the application scope of the Axis architecture; this makes the applications become available and won’t be down since they are initiated once, besides, easy to maintain persistent data, such as participants lists, in the coordinators and transaction-aware web services.

If the developers are going to make the new developed Web Services transaction system connected with the legacy transaction systems, they can just package the existing systems by interfaces of the architecture we defined.

CHAPTER 5
DISCUSSION AND RELATED WORKS

According to existing specifications of Web Services transaction, there are several vendor products for reference: 1). IBM Corp. implements a application programming interface package of WS-AT for WebSphere platform, but not including WS-BA yet. 2). As for BTP, HP Company has developed a transaction system called HP-WST [21] to support BTP in the first year of BTP and is open, but no further development in the later years. 3). WS-CAF has a compatible product published by Arjuna Technologies, called XML Transaction Service [22]. Arjuna declares to support both WS-C/AT/BA and WS-CAF; however, it is not open. 4). Choreology Corp. from England developed a product called Cohesions [20], supporting BTP and WS-C/T (the old version of WS- specifications).

As we can learn from the previous paragraph, the implementations from the vendors are not completed yet and none is for open platform.

	BTP	WS-C/T	WS-CAF	Me
Complete Year	2000	2004	2003	2004
SOAP based Transaction Protocol	No	Yes	Yes	Yes
WSDL based Transaction interface	No	Yes	No	Yes
Context Management Framework	No	No	Yes	Yes
Implementations	Old	Vendor	Vendor	Open

Figure 5-1 Comparisons between Our Transaction Architecture and Related Works

The figure 5-1 shows comparisons between our transaction architecture and the vendor specifications. There are four important characteristics: SOAP based transaction protocol, WSDL based transaction interface, context management framework and implementations. Our transaction architecture is obviously the winner of all; it has all of the characteristics while others don't have all.

Our transaction architecture achieves the following features as well:

1). Loosely-coupled, asynchronous communications, dynamic composition architecture

2). Verified and open for Web Services transactions

3). Suited for heterogeneous, long-lived transactions

4). Compatible with WSDL, SOAP, WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity by leveraging WS-C/T and WS-CAF and integrating the pros and minimize the cons of both

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

In thesis, we proposed a demonstrative system for building transaction architecture in Web Services environment. This will improve the interoperability among various platforms and systems with B2B application integration. Besides, the independent of transaction architecture from inside of an application is designed with references of industry specifications; our experience of implementation will help the developers of Web Services transaction-related systems in implementation. Nevertheless, with the movement of XML and Web Services, transaction architecture in Web Services environment must become standardized. We have compared several commons and differences of those industry specifications, and we will dedicate to all industry specifications or implementation of standards. Finally, we will provide a complete development flow and design tool for developers. Before we finish this thesis, there is still not any complete reference product or implementation of WS-C/AT/BA. The goal of our researches chiefly is to design an open architecture to provide developers in design of Web Services transaction. Developers can implement their desired systems by referring to your design and implementation experiences.

6.2 Future Work

6.2.1 Web Services Composability

Service composition allows developers to “compose” services that exchange SOAP messages and define their interface in WSDL and WS-Policy into an aggregate solution. The aggregate is a composed Web service. WS-Policy enables a service to specify what it expects of callers and how it implements its interface. The BPEL4WS (Business Process Execution Language for Web Services specification) together supports service composition. Supporting WS-Policy and BPEL4WS must become a goal in the future.

6.2.2 Web Services Interoperability

Interoperability issues are very important in Web Services environment. Currently, our system does not compatible with the definition of WS-I Basic Profile [27]. This profile defines all basic constraints about Web Services interoperability to various platforms or systems, such as Microsoft .NET and J2EE Web Services. Document/Literal type messaging is an example of such issues.

6.2.3 Services Assurance and Reliable Messaging

Because of the Web Services environment is cross organizations and enterprises, we need authentication and message integrity, confidentiality, trust and privacy. WS-Security support secure Web Services with existing security models, such as Kerberos, X509, etc. WS-Trust defines a Security Token Service (STS), which is a distinguished Web Service that issues, exchanges and validates security tokens. In addition, we need WS-ReliableMessaging for building unique identifiers, sequence numbers and retransmission.

CHAPTER 7
REFERENCE

[1]. World Wide Web, http://www.w3.org/.

[2]. J. Gray and A. Reuter, “Transaction Processing: Concepts and Techniques,” Morgan-Kaufmann Publishers, 1993.

[3]. Web Services, http://www.w3.org/2002/ws/.

[4]. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “Web Services Description Language (WSDL) 1.1,” W3C, Note 15, 2001, www.w3.org/TR/wsdl.

[5]. “Simple Object Access Protocol (SOAP) 1.1,” http://www.w3.org/TR/soap/, W3C Note, May 2000.

[6]. Microsoft Corporation et al., “Web Services Coordination (WS-Coordination),” http://msdn.microsoft.com/library/en-us/dnglobspec/html/wscoor.asp.

[7]. IBM Corporation et al., “Web Services Atomic Transaction (WS-AtomicTransaction),” http://msdn.microsoft.com/library/en-us/dnglobspec/html/wsat.asp.

[8]. BEA Systems et al., “Web Services Business Activity Framework (WS-BusinessActivity),” http://msdn.microsoft.com/library/en-us/dnglobspec/html/wsba.asp.

[9]. Microsoft Corporation, “White Paper: Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity,” http://msdn.microsoft.com/library/en-us/dnwebsrv/html/wsacoord.asp.

[10]. 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.

[11]. Organization for the Advancement of Structured Information Standards, “Business Transaction Protocol,” committee specification, http://www.oasis-open.org/committees/business-transactions/documents/specification/2002-06-03.BTP_cttee_spec_1.0.pdf, 26 June, 2002.

[12]. Organization for the Advancement of Structured Information Standards, “OASIS Web Services Composite Application Framework,” http://www.oasis-open.org/committees/documents.php?wg_abbrev=ws-caf.

[13]. Brahim Medjahed, Boualem Benatallah, Athman Bouguettaya, Anne H. H. Ngu, Ahmed K. Elmagarmid, “Business-to-business interactions: issues and enabling technologies”, The VLDB Journal (2003) 12: 59-85.

[14]. Keisuke Yano, Hirotaka Hara, Sanya Uehara, “Collaboration management framework for integrating B-to-B and Internal Process”, Proceedings of the Sixth International Enterprise Distributed Object Computing Conferene (EDOC’02), 2002.

[15]. Asit Dan, “Downloadable Service Contracts for Disconnected Transactions”, Proceedings of the 12th Internet Workshop on Research Issues in Data Engineering: Engineering e-Commerce/e-Business Systems (RIDE’02), 2002.

[16]. Deron Liang, Satish K. Tripathi, “Performance Analysis of Long-Lived Transaction Processing Systems with Rollbacks and Aborts”, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 5, October, 1996.

[17]. 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’02w), 2002.

[18]. Takashi Hatashima, Daigoro Yokozeki, Masataka Suzuki, Kouji Tokumaru, “WebServices Processing Platform – eCo-Flow”, Proceedings of the 2002 Symposium on Applications and the Internet (SAINT’02w), 2002.

[19]. Mark Little, “Transactions and Web Services”, Communications of the ACM, Vol. 46, No. 10, October, 2003.

[20]. Michel P. Papazoglou, “Web Services and Business Transactions,” http://maximus.uvt.nl/sigsoc/pub/Papazoglou%20-%20Web%20services%20and%20business%20transactions.pdf.

[21]. IBM Corporation, “Web Services Atomic Transaction for WebSphere Application Server,” http://www.alphaworks.ibm.com/tech/wsat/.

[22]. Choreology Ltd, “Choreology Cohesions,” http://www.choreology.com/products/index.htm.

[23]. Hewlett-Packard, “HP web services transactions HP-WST,” http://www.hpmiddleware.com/downloads/pdf/wst_specsheet.pdf.

[24]. Arjuna Technologies, “XML Transaction Service,” http://www.arjuna.com/products/arjunaxts.

[25]. The OMG Group, “Activity Service specification,” http://cgi.omg.org/cgi-bin/doc?orbos/2000-06-19.

[26]. Sun Microsystems, “Java Transaction API specification,” http://java.sun.com/products/jta/index.html.

[27]. Web Services Interoperability Organization, “Basic Profile 1.1 WG Draft”, http://www.ws-i.org/Profiles/Basic/2003-12/BasicProfile-1.1.pdf, Dec. 19, 2003.

[28]. Apache Software Foundation, “Jakarta Project Tomcat,” http://jakarta.apache.org/tomcat/index.html.

[29]. Apache Software Foundation, “Extensible Interaction System (Apache AXIS),” http://ws.apache.org/axis/.

[30]. The Open Group, “Distributed Transaction Processing: The XA Specification,” http://www.opengroup.org/products/publications/catalog/c193.htm, 1992.