ABSTRACT

Web Service Atomic Transaction (WS-AT) is a standard used to implement distributed processing over the internet. Trustworthy coordination of transactions is essential to ensure proper running of web services. The smooth running of web services in spite of some faults is possible with the implementation of Byzantine fault tolerance mechanisms. An online banking application has been implemented, where all the core services (activation, registration, completion and coordinator) are replicated and protected by the Practical Byzantine Fault Tolerance technique (PBFT). The banking application has three components (client, bank and coordinator), where each of these three perform certain set of actions like account creations, fund transfers, loan requests and grants, and many more. PBFT coupled with the two-phase commit protocol, have been incorporated into the banking service to ensure fault free atomic transactions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In recent years, the use of the web services technology on the Internet has increased widely. This technology has enabled integration and collaboration of large businesses, thus automating the business processes. Good security and performance are required from distributed transactional web services, where data inconsistency is of major concern. Web Services Atomic Transactions (WS-AT) [10] was developed to provide interoperability among web services. It was developed by a coalition preceded by IBM and Microsoft. It has been recently accepted by the Organization for the Advancement of Structured Information Standards (OASIS) [10] as a web service standard. Activation, registration, completion and coordinator services are the core services provided by the WS-AT. Here, the coordinator offers these services to the initiator and the participants of a transaction. The Activation service creates a coordinator object for every transaction. The Registration service admits a participant into the transaction. The Completion service initiates the distributed commit of the transaction at the initiator's request. The Coordinator service coordinates the participants to commit or abort the transaction atomically [12]. It is extremely important to ensure that these services are trustworthy even with the occurrence of faults [8].

Every transaction in WS-AT does not require a Byzantine agreement, so the application of fault tolerance technique on each of them will decrease the performance. A better performance can be achieved by reducing the total number of Byzantine agreements in a transaction [2]. In the Byzantine fault tolerance mechanism every non-faulty transaction is included in the two-phase commit protocol [6]. The Two-Phase Commit (2PC) protocol is a standard distributed commit protocol [11] used in distributed transactions to achieve atomicity of operations. However, the 2PC protocol is designed with the assumptions that the coordinator and the participants are only subject to benign faults and that the coordinator can be recovered quickly.

This protocol does not work if the coordinator is subject to arbitrary faults, also referred to as Byzantine faults [8], because a faulty coordinator might send conflicting decisions to different participants to prevent them from terminating the transaction atomically (i.e., some participants may commit the transaction, while others abort it).

The problem of Byzantine fault tolerant (BFT) distributed commit for atomic transactions has been under research for the last two decades. This problem was first researched by Mohan *et al.* [9] and a solution was provided by integrating the Byzantine agreement (BA) and the 2PC protocol. The second phase of the 2PC protocol was replaced by a Byzantine agreement phase. This prevented the faulty coordinator from spreading conflicting messages to different participants without being detected. However, this approach was not practical because of three main shortcomings.

- It involved all participants in the Byzantine Agreement phase for each transaction. Therefore, the overhead of reaching a Byzantine agreement would be large.
- This method could only protect the members in the root cluster, but not the subordinate coordinators and the participants outside.
- It required the participants in the root cluster to have knowledge about all other participants in the same cluster, which prevents dynamic propagation of the transaction.

The BA is carried out among the coordinator replicas. The PBFT algorithm is used in this banking application because of its efficiency as proved by Castro *et al.* [2]. This algorithm ensures ordered atomic multicast of requests to a replicated server. Over the years modifications were done to this algorithm to ensure atomic distributed commits. The major change was done to the first phase, where the primary coordinator replica is required to use a decision certificate, which is a collection of the registration records and the votes it has collected from the participants, for accountability. This certificate helps the non faulty backup coordinator replica to verify the primary′s decision. The ease in implementation of PBFT is the reason why it will be used in our banking application. Most of the Byzantine fault tolerance algorithms can be used instead of PBFT without significant changes.

2. BACKGROUND

This chapter gives necessary information about the web services, their standards, atomic transactions and Byzantine fault tolerance. This will help to understand the background knowledge of our banking system.

2.1 Web Services Technology

2.1.1 Web Services

There is no formal definition for a web service as its interpretation varies. In simple words, a web service is a type of service deployed over the internet. The Web Services Technology refers to the set of standards that enable automated machine-to-machine interactions over the World Wide Web. The key technologies are eXtensible Markup Language (XML) [1], HyperText Transfer Protocol (HTTP) [5], Simple Object Access Protocol (SOAP) [7], Web Services Description Language (WSDL) [3], and Universal Description, Discovery and Integration (UDDI) [4]. From an architectural point of view, the web services platform consists of web services providers, web services consumers, and the UDDI registries, as shown in Figure 2.1 .

2.1.2 Standards

eXtensible Mark up Language (XML)

XML is designed to facilitate self-contained, structured data representation and transfer over the Internet. It allows users to define their own tags, making it easily extensible. XML messages enable different applications to communicate with each other over the network using a variety of transport-level protocols such as HTTP and SMTP. To invoke a web service, a user only needs to send an XML request message to the web services provider. The provider will then send an XML reply message back containing the results requested by the user. Typically, the XML messages must

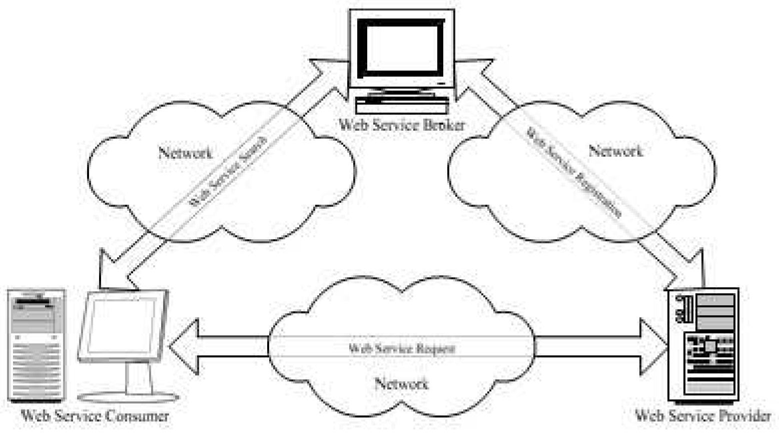

Fig. 2.1. Web service Architecture.

conform to the SOAP standard.

Simple Object Access Protocol (SOAP)

SOAP [7], a communication protocol for exchanging messages over the Internet, provides a standard modular packaging model, a data encoding method and a way to perform Remote Procedure Calls (RPCs). SOAP is easy to use and is easily extensible due to its use of XML as its message format. Like many public-domain application-level protocols, such as SMTP, a SOAP message contains a SOAP envelope and a SOAP body. A SOAP message often contains an optional SOAP header element and a fault element if an error is encountered by the sender of the SOAP message.

Universal Discovery Description and Integration (UDDI)

The UDDI registry service acts like yellow pages for business providers and consumers. Business owners can publish their web services to the UDDI registry; their partners and consumers can then locate the web services in need and obtain detailed information by searching the registry. There are three main components in UDDI, commonly referred to as white pages, yellow pages and green pages. The white pages provide web service providers information, such as name, address, contact information and identifiers. The yellow pages describe industrial categories based on standard taxonomies. The green pages present technical information regarding the web services. The UDDI also support several methods of searching, e.g., search by service providers location, by specified service types, etc.

Web Service Description Language (WSDL)

WSDL provides a structured approach to describe a web service based on an abstract model. For each web service, the corresponding WSDL document specifies the available operations, the messages involved in these operations, and a set of endpoints to reach the web service. WSDL is also extensible through the use of XML. In particular, it enables the binding of multiple different communication protocols and message formats.

2.2 Web Services Atomic Transactions (WS-AT)

The Web Services Atomic Transactions standard specifies two protocols, the 2PC protocol and the Completion protocol, and a set of services. These protocols and services together ensure automatic activation, registration, propagation, and atomic termination of a distributed transaction based on web services. The 2PC protocol is run between the participants and the coordinator, while the Completion protocol is run between the coordinator and the initiator. The Initiator is responsible for not only starting the distributed transaction but also ending it. The Participants register with the Coordinator when they start. The 2PC protocol commits a transaction in two phases. During the first phase (the Prepare phase), the Coordinator forwards a request to all of the Participants so that they can get ready to commit the transaction. If a Participant is able to commit the transaction, it responds with a Prepared vote. Otherwise, the Participant responds with an Abort vote. When a Participant responds with a Prepared vote, it enters the Ready state. A Participant must be prepared to either commit or abort the transaction. A Participant that has not responded with a Prepared vote can abort the transaction. When the Coordinator has received votes from every Participant, or after a specific time, the Coordinator starts the second phase (the Commit-Abort phase) by notifying the Participants of the result of the transaction. The Coordinator will commit a transaction if it has received Prepared votes from all of the Participants during the first phase. If not, it decides to abort the transaction.

The Coordinator side services are as follows:

Activation service

The Activation service creates a Coordinator object and a transaction context for each transaction. Essentially, the Activation service behaves like a factory object that creates Coordinator objects.

Registration service

The Registration service allows the Participants and the Initiator to register their endpoint references.

Completion service

The Completion service allows the Initiator to signal the start of the distributed commit.

Coordinator service

The Coordinator service runs the 2PC protocol, which ensures atomic commitment of the distributed transaction.

The Activation service is used for all transactions. It is provided by a single object, which is replicated in the implementation. When a distributed transaction is activated, a Coordinator object is created. The Coordinator object provides the Registration service, the Completion service and the Coordinator service. The transaction context contains a unique transaction id and an endpoint reference for the Registration service, and is included in all request messages sent during the transaction. The Coordinator object is replicated in our framework.

The Initiator side services are as follows:

CompletionInitiator service

The CompletionInitiator service is used by the Coordinator to inform the Initiator of the final outcome of the transaction, as part of the Completion protocol.

The Participant side services are as follows:

Participant service

The Participant service is used by the Coordinator to collect votes from, and to send the transaction outcome to, the Participant. The different phases of a distributed transaction are shown in Figure 2.2 for the online banking application. The bank provides an online banking web service that a customer can access. The transaction is started as a result of the customer invoking a web service of the bank to transfer an amount of money from one account to another.

2.3 Byzantine Fault Tolerance (BFT)

BFT refers to the ability of a system to tolerate Byzantine faults. A Byzantine fault is an arbitrary fault, which might be a hardware or a software fault caused by an intrusion into the system. BFT can be achieved by replicating the server and by ensuring that all server replicas reach agreement on each input despite Byzantine faulty replicas and clients. Such an agreement is referred to as a Byzantine Agreement

(BA) [8]. As mentioned earlier, the Practical Byzantine Fault Tolerance algorithm of Castro and Liskov [2] has been implemented in our online banking application.

The PBFT algorithm is executed by a set of *3f+1* replicas and tolerates *f* Byzantine faulty replicas. One of the replicas is the primary, and the rest of the replicas are the backups. The normal operation of the PBFT algorithm involves three phases. During the first phase i.e. the Pre-Prepare phase, the primary multicasts a Pre-Prepare message containing the clients request, the current view number, and the sequence number of the request to the backups. A backup verifies the Pre-Prepare message and ordering information by checking the signature, view number, and sequence number. If the backup accepts the message, it starts the second phase i.e. the Prepare phase by multicasting to the other replicas a Prepare message containing the ordering information and the digest of the request message being ordered. A replica waits until it has collected matching Prepare messages from *2f* other replicas. It then starts the third phase i.e. the Commit phase by multicasting a Commit message to the other replicas. The Commit phase ends when a replica has received matching Commit messages from *2f* other replicas. Now, the request message will be ordered and ready to be delivered to the server application.

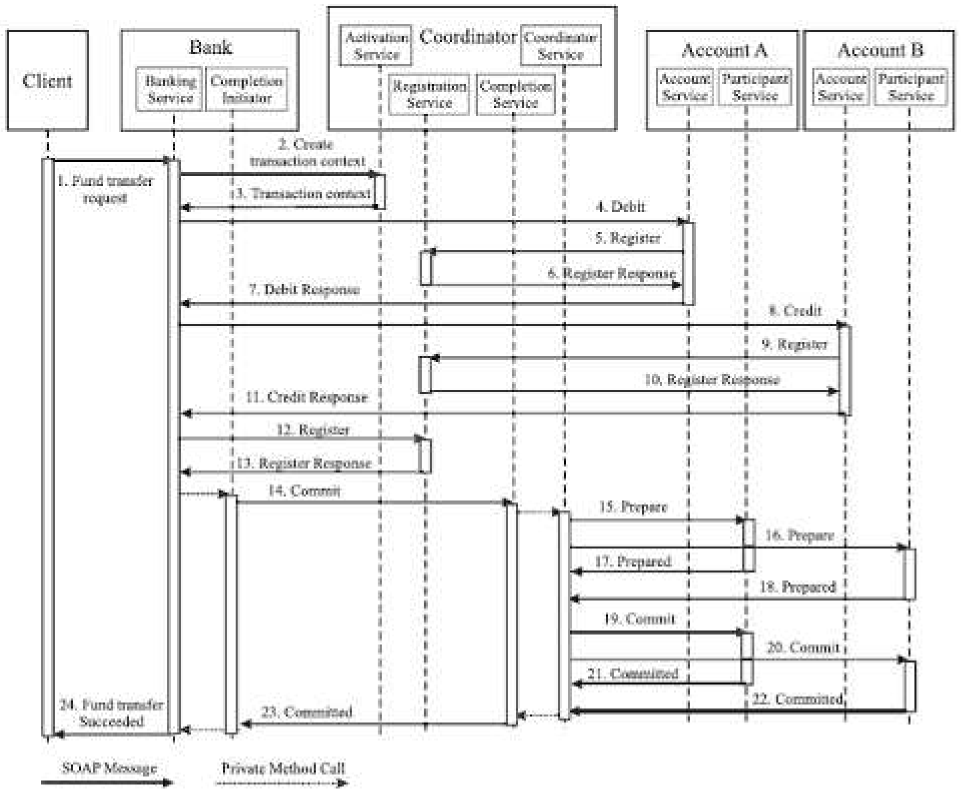

Fig. 2.2. Different phases in the banking application [12].

3. SYSTEM DESIGN

The online banking application under development offers various individual web services to the users. It is assumed that these web services are accessed through a browser. A distributed transaction is created on every user request for the coordination of web services. The web service provider is the Initiator of the transactions, which starts and terminates the transactions, and also propagates them to other Participants. A new Coordinator object is created for every transaction, which runs until the transaction ends. Participants are not replicated here, in the case of faulty ones, the coordinator replicas agree on the same outcome of the transaction. Practically, it is not possible to attain atomic commitment in the case of faulty participants. The 2PC protocol assumes the Participants to be non-faulty by default.

The software technologies used for the development are as follows:

- Operating System : Windows 7.
- Integrated Development Environment : Visual Studio 2010.
- Database : SQL Server 2005.
- Web Technology : ASP.NET, and ADO.NET.
- Programming language : C#

The online banking system will provide several functionalities. Table 3.1 lists the functionalities of the different users of the system. Figure 3.3 shows the complete functionalities of the system and the entities associated with them.

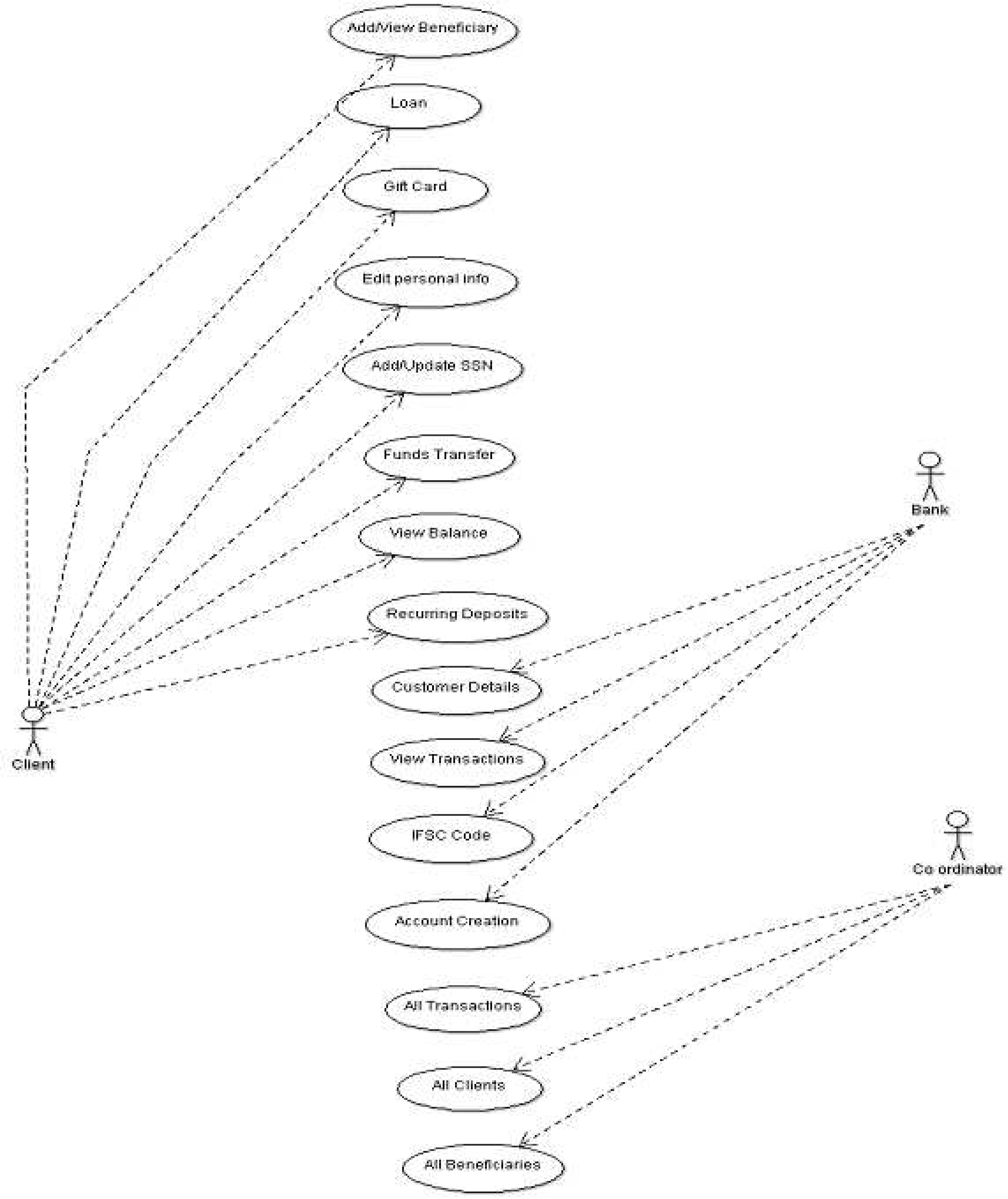

Fig. 3.3. Overview of the online banking application in the form of a use case diagram.

User	**Functionality**
Bank	Create bank accounts, coordinate the clients and their accounts.
Coordinator	Access complete client information.
Client	Transfer funds to the same bank or different bank, add or view beneficiary, update personal information, apply for a loan, perform recurring deposits and send complaints.

Table 3.1. Functionalities of the online banking system.

BIBLIOGRAPHY AND REFERENCES

[1] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., and Cowan, J. Extensible markup language (xml) 1.0 -w3c recommendation 16 august 2006, edited in place 29 september 2006. *World Wide Web Consortium(W3C), URL: http://www.w3.org/TR/2006/REC-xml-20060816/(06.08.2007)* (2006).

[2] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst. 20*, 4 (Nov. 2002), 398–461.

[3] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web services description language (wsdl) 1.1. w3c, 1.march 2001. *URL: http://www.w3c.org/TR/wsdl* (2001).

[4] Clement, L., Hately, A., von Riegen, C., and Rogers, T. Uddi version 3.0. 2 uddi spec technical committee draft, dated 20041019. *Organization for the Advancement of Structured Information Standards (OASIS), October 19* (2004).

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol–http/1.1,

1999.

[6] Gray, J., and Reuter, A. *Transaction Processing: Concepts and Techniques*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[7] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. Simple object access protocol (soap) 1.2. *World Wide Web Consortium* (2003).

[8] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 382–401.

[9] Mohan, C., Strong, R., and Finkelstein, S. Method for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In *Proceedings of the second annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1983), PODC '83, ACM, pp. 89–103.

[10] Newcomer, E., Robinson, I., Little, M., and Wilkinson, A. Web services atomic transaction (ws-atomic transaction). *Technical committee specification, OASIS (Organization for the Advancement of Structured Information Standards) 5* (2006).

[11] Staff, B. S. I. *Information Technology. Distributed Transaction Processing. the XA Specification.* B S I Standards, 1997.

[12] Zhang, H., Chai, H., Zhao, W., Melliar-Smith, P. M., and Moser, L. E. Trustworthy coordination of web services atomic transactions. *IEEE Transactions on Parallel and Distributed Systems 23*, 8 (2012), 1551–1565.