Transaction Management in SOA based System

Rohit Barotia
Research Scholar,
Jagannath Universirty Jaipur

Abhay Upadhaya, PhD
Associate Professor,
Department of A.B.S.T.
University of Rajasthan

Himanshu Pareek
Guest Faculty
Commerce College
University of Rajasthan, Jaipur

ABSTRACT

In SOA-based system there is a big problem of integration of applications and business transactions, which often results into mismatched trust among domains and inactive during the long period. These activities put a big challenge to traditional ACID transaction processing. To prove this theoretically we presented three successive theoretical analyses which are familiar to the present transaction theories; including the classical ACID model the extended transaction model with ACID relaxations. These theoretical analyses are architecturally capable of playing the role of a loosely-coupled, pluggable middleware, overlaying heterogeneous legacy systems.

Keywords

ACID, EAI, ERP, SOA

1. INTRODUCTION

In the recent times SOA (service oriented architecture) emerged as a strong competitor to conventional integrated solutions like EAI (Enterprise Application Integration) and ERP (Enterprise Resource Planning), and is which is claimed by many, to be the solution for the problem of companies system integration. It would be the strong recommendation that the earlier versions of ERP solution should be removed with a new and comprehensive ERP system, Instead of being deconstructed; SOA adopts a softer and incremental approach to system integration legacy functionalities are wrapped into web services. EAI (Enterprise Application Integration) solutions are the proprietary product of an individual and cannot be integrated with other solutions. SOA-provides interoperability, interoperability is an unrivalled choice when we do not have the facility of starting from scratch and forcing everyone to throw away their previous investments in the legacy systems' infrastructure, training, etc., and when an open standard-based solution is preferred to a proprietary solution. Attaining global agreement is additionally dependent on the ability of web services to provide interoperable transaction mechanisms despite partial failures experienced in individual services. The transaction problem is a distributed system problem, since any distributed transaction needs a mechanism to reach global agreement in the face of partial failures. So, is the answer to web service transactions what we have been using for distributed transaction processing over the last two decades, but this time by applying XML and SOAP rather than the traditional platform-dependent message passing.

2. SOA BASED SYSTEM INTEGRATION

SOA (Service Oriented architecture) can be viewed as the service oriented architecture with a standard-driven approach to integrating heterogeneous legacy applications. Service-oriented integration is appropriate under the assumption that leveraging legacy logic is preferable to replacing it. In SOA web services are integrated and these services are builds the application, web services are the internet enabled and services oriented integration component in system integration setting. A web service can be used to abstract the application logic that is locked in existing legacy applications. A web service can also compose other web services to form a service-oriented process flow; still another service type can be used for the coordination of other web services.

In regards to legacy applications, being "legacy" does not necessarily mean that an application employs older technology, e.g. mainframe. In our project, an application deployed on a modern Java EE platform can also be deemed "legacy", when there is the need for it to communicate with business functionality deployed on a different platform. SOA usually extends the applications' existing multi-tier architecture by introducing a logical service integration layer. Service-oriented design is a extension of object- and component-based design. The most common service-oriented design principles include loose coupling, autonomy, discoverability, reuse, contract based design, abstraction, statelessness.

3. SYSTEM INTEGRATION APPROACH

The primary reason behind system integration is for two or more applications to collaborate on a certain task. This collaboration can be as simple as one application retrieving a value stored in another's database. There are three system integration approaches with increasing complexity levels. We assume that all legacy applications have a data-tier and an application-logic tier, among other tiers.

a. Data-level point-to-point integration

In data integration level, data from 'A' application is accessed by 'B' application without involving A's application logic, can be classified as being among the early system integration attempts. Incompatible database platforms stop the use of conventional data access technologies such as remote JDBC or ADO connections. Interoperability can be achieved by placing a wrapper service, as an extra service integration layer with existing data layer. Wrapper service can be used as a central data access controller from the service client. If designed with a common service interface, the use of the wrapper service can result in improved performance by getting fine-grained data traffic.

3.2 Application-level point-to-point integration

In application-level integration, direct data access is not an option. Application 'A' and 'B' exclusively communicate via their application-logic tier. The assumption of mismatched application platforms prevents the use of traditional remote invocation technologies such as Java RMI, RPC or .NET. The integration components such as proxy service or wrapper service can be used with the applications layer. As the proxy services can be easily auto generated because the proxy service interface easily mixed with the functionality from which the proxy is derived, like java methods. Although XML is mixed with the proxy services, this method uses PRC-centric message exchange pattern, as SOAP is designed to support both the tightly coupled RPC-centric, and the loosely coupled document-centric, message exchange patterns. A wrapper service is used custom-developed and designed to representation the coarser-grained legacy logic.

3.3 Process-level incorporation

Process-level incorporation uses new automated business processes by integrating existing applications, or sub-processes. A business process integrating disparate and distributed legacy functionalities is also known as a distributed workflow. To achieve process-level integration, it is not only enough to place wrapper or proxy services in a service integration layer on top of the legacy application layers. As in the point-to-point application-level integration model. We also need a resource to store and execute the business rules which lead the workflow. Traditionally, process-level integration is achieved through either proprietary point-to-point integration models or a proprietary hub-and-spoke EAI architecture.

4. CLASSICAL TRANSACTION PROCESSING MODELS

Transactions are a fundamental concept in building reliable distributed applications. A transaction is the grouping of a set of operations so that they constitute an indivisible, logical unit of work. Discussions of transaction processing models, we will briefly state the failure model.

4.1 Failure models

Transactions are a basic concept in the development of distributed applications. A transaction is the group of a set of operations so that they represent an indivisible, logical unit of work. In order to set an unambiguous situation for later discussions of transaction processing models, we will briefly state the failure model assumed in this paper.

- **Logical Failure:** The idea of placing multiple operations within a single transactional scope is to safeguard the system against various degrees of logical failures, such as lost update, inconsistent retrieval, dirty read, premature write, etc.
- **Omission failures**: a transaction processing system should be capable of dealing with omission failures in the form of process crashes, disk failures, or communication failures. In traditional transaction processing systems, these omission failures are masked by assuming a stable storage and a stable processor.
- **Disk Crash:** Disk crash is covered by stable storage. After a disk crash, it is the job of a recovery manager to keep on all committed updates from a recovery file to disk; this can be done through RAID (Redundant Array of Inexpensive/Independent Disks). A recovery file is a logical concept.
- **Process crash omission failures:** Process crash omission failure means replacing a crashed process with a new process that is reinstated from stable storage and other processes.
- **Communication omission failure:** There should be reliable point to point communication channel; we assume that reliable communication exists. In real-world SOA, reliable communication is often handled by another protocol on the SOA stack, namely WS-Reliable Messaging.

5. CLASSICAL TRANSACTION MODEL

According to the Classical transaction theory, transactions should put on view a serial equivalence means any concurrent execution must have the same effect as a serial execution and failure atomicity means the effects are atomic in the event of a server crash. In this paper these requirements are collectively referred to as reliability guarantees. Transaction we refer to the collection of reliability for transactions as the ACID properties:

1. **Atomicity**: Atomicity means the effects of all operations are reflected in the Transaction or none are.
2. **Consistency:** Consistency means a transaction must bring the system from one consistent state to another.
3. **Isolation**: The effects of the operations are not visible outside the Transaction until it completes successfully. Each transaction appears as if it executes in isolation.
4. **Durability:** once a transaction successfully completes, the changes it has made will survive system failures.

ACID Properties guarantees about: (1) single-machine transactions, (2) distributed transactions with the synchronous RPC/RMI communication paradigm (3) nested transactions.

5.1 Single-machine transactions

In single machine transaction model Recovery Manager is responsible for ensuring the atomicity and durability properties of transactions. Resource and transaction is managed by a database Transaction Manager. Consistency is and is dealt with at the application level. Isolation assures you in varying degrees by a concurrency control mechanism, such as locking, optimistic concurrency control, or timestamp-based concurrency control. DBMS's allow for choosing between different read uncommitted, read committed, repeatable read, serializable (isolation level) to match different data access patterns. Object-oriented languages have those facilities which allow client application to communicate with database monitor through OOPS API. The System. Transactions namespace in .NET 2.0 Framework is a case in point.

5.2 One-phase atomic commit protocol

In one phase atomic control protocol transaction model assumes that all resources are under the control of a single Transaction Manager. One-phase atomic commit protocol assumes that abort operation is completed as an atomic step for all objects, data, etc. participating in the transaction.

6. DISTRIBUTED TRANSACTIONS

A distributed transaction accesses objects managed by multiple, distributed servers. The classical model for distributed transactions - both database-level and application-level - uses a single Transaction Coordinator, and multiple Resource Managers. While the Transaction manager initiates and coordinates an atomic commit protocol, each individual

Resource Manager manages its local resource and responds to the Transaction manager.

6.1 The two-phase atomic commit protocol

The one-phase protocol can be fitted in a single-machine environment, because it is insufficient for distributed transactions. In two phase commit protocol the first preparation phase, the user are asked to give information about the transaction to be commit. An actual commit is not carried out until the second, completion phase, when a joint commit decision has been reached.

5.1 The three-phase commit protocol

In this protocol, a transaction can be survived with the with the collaborative decision. The three-phase commit protocol is non blocking because a running process can finish without introducing a globally inconsistent state. This can be done by synchronization points on the communicating state machines at the server's and Participants' processes, respectively. Theoretically the three-phase commit protocol's algorithm is superior to any other protocol but it is seldom used in practice. This is due to the fact that blocking the two-phase commit protocol very rarely occurs, and three phase protocol is complicated to implement

5.2 Nested distributed transactions

A nested distributed transaction can have a hierarchical tree-like structure. A parent transaction can execute more dependent transactions on the parent transaction. It provides you extra concurrency which is more flexible in a sense that the root transaction manager can choose to commit the whole transaction even when some of the child transaction have failed. Locking resources in nested distributed transactions are subject to more understated rules.

7. EXTENDED TRANSACTION MODEL

The distributed two-phase commit protocol is a well-formed, ACID protocol. ACID behavior is too expensive in terms of security.

7.1 Relaxing the ACID properties

Some researcher has proposed ways to relax the ACID properties. This section outlines a number of the possible ACID relaxations.

In larger enterprise systems, the results of the distributed system often involve complicated business logic which is difficult to integrate. In an extended transaction model, the atomicity property can be relaxed

1. Participants can only be included in the final results.
2. Participants can be participated in the local transaction commit without waiting for the global transaction.

8. WEB SERVICE TRANSACTION PROCESSING MODEL

It's a need to adopt the classical and extended transaction processing models in a service-oriented architecture. It is a challenge to understand the use of SOA impacts on transaction and why the ACID transaction model is not sufficient.

8.1 Impact of web services on transaction management

It becomes more complex if we provide less efficient solution as solution compared to transaction management at the application or database level because web services usually wrap around common functionality in legacy systems which otherwise would not be able to communicate. These legacy systems execute in different environment with the implementation of ACID-style two-phase or three-phase commit. If all the services are managed within a single transaction, web services have no role in the execution. Transaction management at the web service level should only be considered when using web services to integrate disparate systems. Interoperability is provided, by wrapping legacy systems behind a web service interface, between otherwise incompatible systems. Examples of such systems are Web Sphere, CORBA, EJB, .NET, SAP etc. The figure also display the possibility for transaction coordination middleware to work in conjunction with business process workflows expressed in Business Process Execution Language. The purpose is to coordinate all the layers to reach a common decision whether to commit, rollback, or compensate the changes done according the business workflow. We assume that web services are rightfully employed.

8.2 A reference model for web service transaction management

We have constructed a reference model for web service transaction management. This model serves the purpose of summarizing a theoretical analysis and serving a transaction which focus on building a service oriented middleware. This model represents three distributed "actors" in web transaction.

Service Container Framework: This framework handles the subsidy applications well as the document handling. On the behalf of these services the service container framework initiates the transaction

The Coordination Framework: This framework act as the extension of the web services protocols. Functionalities are as follows:

a) Activation service through which we start a new transaction
context, so it is referred to as *Coordination Context.*

b) Registration service through which a participant joins an existing transaction scope.

c) Completion service through which transaction is terminated.

The Legacy Server: The Legacy Server is a server through which a legacy server is hosted. Local resources are accessed through a local Resource Manager API. The resource manager is responsible to set the remote user to capture its local resource.

8.3 Three Phases In The Lifecycle Of A Transactional Service

The three phases of lifecycle are drawn inside the Service Container; this proves the fact that all transaction services are "hosted" by Service Containers.

- In the **Enlistment phase** two types of messages are exchanged between the user and a server.

1) Activation: The User initiates the transaction and issues a request to the Activation service for creating a new Coordination Context.

2) Registration: user wishes to join an transaction scope send requests to the Registration service.

- In the **Execution phase**, the application logic is executed, involving the invocation of one or more child services running on legacy servers.
- In the **Termination phase**, specific transaction protocols will be driven to their termination.

9. CONCLUSION

In we have analyzed the classical transaction model and surveyed four

Variants of the commit mechanism: single-machine one-phase commit, distributed two-phase commit, distributed three-phase commit and nested two phases commit. We conclude that the classical model is a good fit for single machine transactions and short-lived distributed transactions across closely integrated trust domains. The classical transaction model has the strength of providing the safe and strict ACID guarantees. However, its use of exclusive long-duration locks (pessimistic concurrency control) or tentative update versions (optimistic concurrency control) is in disharmony with asynchronous, long-running transactions, or transactions that require interim results to be visible to concurrent users. this paper we are trying to analyze the characteristics of transaction in SOA and how the transaction is different from traditional programming. In the world of web services where long-running transactions are more important rather than the exception, blocking transaction processing models such as the two-phase commit protocol are challenged. New transactional models are becoming dominant than ACID based transactions. Various combinations of the web services are required to use the various protocols to bridge the gap between execution environments. In this analysis the design criteria is given, which is the base for the reference model which can be applicable in the designing of web service transaction management middleware. In our theoretical analysis, we have aimed to spell out the new challenges SOA has posed in terms of the management of distributed transactions.

10. REFERENCES

[1] [AT-Interop, 2004]WS-Atomic Transaction Interop Scenarios. November, 2004.

[2] [BA-Interop, 2006] WS-Business Activity Interop Scenarios
1.1 November, 2006.

[3] [Bernstein, 1997] Philip A. Bernstein and Eric Newcomer.Principles of Transaction Processing. Morgan Kaufman.

[4] Business Transaction Protocol (BTP) Committee Specification (2002).

[5] [Chappell, 2004] David A Chappell. Enterprise Service Bus Theory in Practice. O'Reilly, USA.

[6] [Denzin, 1978] Norman K. Denzin. The research Act. McGraw Hill.

[7] [Ementor, 2006-b] Ementor. Use Case Model – CAP.

[8] [Ementor, 2006-c]Ementor. Data Model – CAP.

[9] [Erl, 2005] Thomas Erl. Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall. New Jersey, USA.

[10] [Fowler, 2003] Martin Fowler. UML distilled A brief guide to the standard object modeling Language. Pearson Education Inc., Boston.

[11] [Frank, 2006] Lars Frank. Databases with Relaxed ACID Properties. Copenhagen Business School Press.

[12] [Hasan, 2006] Jeffrey Hasan and Mauricio Duran. Expert Service-oriented Architecture in C# 2005. 2nd. A Press, California, USA.

[13] [IEEE, 1990] IEEE. Standard Glossary of Software Engineering Terminology. (IEEE Std. 610.12-1990). IEEE Computer Soc.

[14] [Kaye, 2003] Doug Kaye. Loosely Coupled - the MissingPieces of Web Services. RDS Press, California, USA.

[15] [Little, 2004] Mark Little, Jon Maron and Greg Pavlik. Java Transaction Processing.Prentice Hall.

[16] [Löwy, 2005]Juval Löwy. Introducing System. Transactions. Microsoft Press.

[17] [Löwy, 2007]Juval Löwy. Programming WCF Services. O'Reilly, USA.

[18] [Lublinsky, 2003] Boris Lublinsky and Dmitry Tyomkin. Dissecting Service-oriented Architectures. Business Integration Journal, USA.

[19] [McConnel, 2004] Steve McConnell. Code Example. Microsoft Press, Redmond, USA.

[20] [Tanenbaum, 2006] Andres S. Tanenbaum and Maarten Van Steen. Distributed Systems Principles and Paradigms. Pearson Prentice Hall, New Jersey, USA.

[21] [Project Tango] Sun's Project Tango - Web Services Interoperability Technologies

[22] **[**Møller, 2004**]** Kasper Møller. Integration of Applications.Thesis report. IT University of Copenhagen, Denmark.

[23] Myers, 1979] Glenford J. Myers. The Art of Software Testing. Wiley, New York, USA.

[24] Löwy, 2007] Juval Löwy. Programming WCF Services. O'Reilly, USA.

[25] [Frank, 2006] Lars Frank. Databases with Relaxed ACID Properties. Copenhagen Business School Press.