Computing, Information Systems, Development Informatics & Allied Research
ISBN 978-2257-44-7(Print) ISSN 2167-1710 (online)
Vol.3 No.1 March 2012

Towards an Enhanced Protocol for Improving Transactional Support in Interoperable Service Oriented Application-Based (SOA-Based) Systems

Laud Charles Ochei
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria
laudcharles@yahoo.com

ABSTRACT
When using a shared database for distributed transactions, it is often difficult to connect business processes and software components running on disparate platforms into a single transaction. For instance, one platform may add or update data, and then another platform later access the changed or added data. This severely limits transactional capabilities across platforms. This situation becomes more acute when concurrent transactions with interleaving operations spans across different applications and resources. Addressing this problem in an open, dynamic and distributed environment of web services poses special challenges, and still remains an open issue. Following the broad adoption and use of the standard Web Services Transaction Protocols, requirements have grown for the addition of extended protocols to handle problems that exist within the context of interoperable service-oriented applications. Most extensions to the current standard WS-Transaction Protocols still lack proper mechanisms for error-handling, concurrency control, transaction recovery, consolidation of multiple transaction calls into a single call, and secure reporting and tracing for suspicious activities. In this research, we will first extend the current standard WS-Transaction Framework, and then propose an enhanced protocol (that can be deployed within the extended framework) to improve transactional and security support for asynchronous applications in a distributed environment. A hybrid methodology which incorporates service-oriented engineering and rapid application development will be used to develop a procurement system (which represents an interoperable service-oriented application) that integrates our proposed protocol. We will empirically evaluate and compare the performance of the enhanced protocol with other conventional distributed protocols (such as 2PL) in terms of QoS parameters (throughput, response time, and resource utilization), availability of the application, database consistency, and effect of locking on latency, among other factors.

Keywords: Database, interoperability, security, concurrent transaction, web services, protocol, service-oriented

1. INTRODUCTION

Many organizations already operate large enterprise web applications based on Java, .NET or PHP platforms that are deployed in a distributed environment. Most of these web applications are developed from services, which are, reusable software components. A service is a natural development of a software component where the component model is, in essence, the set of standards associated with web services (Sommerville, 2007). An example of this type of application is the traditional common e-commerce application, where for example, a purchase order has to be submitted across multiple systems.

Managing transactions in a distributed environment can be very challenging especially when you have concurrent transactions with interleaving operations that spans across different applications and resources. It is often very difficult to connect concurrent business processes running on disparate platforms into a single transaction. For example, one platform could insert or update data, and then another platform later access the changed or added data. This severely limits the transactional capabilities for cross-platform business process management (Gabhart, 2004).

Addressing this problem in an open and dynamic environment of web services especially when concurrent transaction calls spans across different applications and resources in a distributed environment poses special challenges, and still remains an open issue. Transactional and security support is vitally important, first, to ensure data integrity when managing transactions in a distributed in a distributed environment (Marina et al, 2006). Quality of service is also important in improving transactional and security support for business operations.

Monitoring the QoS across large-scale, distributed, heterogeneous applications is very challenging, because software and hardware components of the system are prone to errors. Nevertheless, establishing this infrastructure is critical to reduce and possibly prevent system downtime. Assuming that there is a problem is an online procurement system that is made up of services (components) from Java EE, .NET, PHP and legacy system commonly deployed in various businesses or composed of a set of federated services distributed over the internet. Hours of service interruptions often translate into millions of dollars in lost revenue. Without a proper system management infrastructure in place, the troubleshooting process can consume days or weeks before the problem is identified and fixed, degrading overall service levels. The ultimate goal of system management is to ensure that the quality of service and operating requirements of all business applications are satisfied.

A procurement system (composed of services) might be considered secure when a customer makes a purchase order using a service implemented on Java EE. This system may not necessarily be secure when order service is exchanging sensitive business information with another service (say the inventory system) implemented on a different platform say .NET. This is because the interoperable application is exposed to security vulnerabilities if one of the services (either the sender or recipient) is exploited or is being attacked by hackers. There are historic incidents of vulnerabilities on the .NET platform and the Java platform. These incidents can become a critical problem that causes financial loss or disruption to the business service. The computer security institute gives the following estimates concerning financial loss and business disruptions.

Table 1: Incidences/Year/Cost

Cause of Loss and Disruption	Year	Estimated Amount
Virus, unauthorized access, and theft of proprietary information	2005	$130 million
Denial of service attacks	2003	$65 million
Denial of service attacks	2005	$7.3 million
Average loss per incident for proprietary information theft	2003	$355552 million

Source: Computer Security Institute, 2005.

For a procurement system, improving transactional and security support translates to ensuring reliable delivery of service request and return of the processing result, fast system performance, and highly available interoperable system. Improving transactional and security support also entails having an acceptable response time for user experience. This includes the flow of order request including the response time for order validation. It also entails having an efficient error handling mechanism; putting in place a common logging system for logging and alerting error messages and database failures.

Considering the peculiarities of most procurements systems which are basically interoperable SOA-based systems, and the need to improve transactional and security support for these applications, most developers and companies are focusing on how to extend the current standard WS-Transaction Framework, and then to develop an enhanced protocol (that can be deployed within the extended framework). This research aims at developing an enhanced protocol that can be integrated within an interoperable SOA-based system such as a procurement system to improve transactional and security support. A hybrid methodology which incorporates service-oriented engineering and rapid application development will be used to develop a procurement system (which represents an interoperable service-oriented application) that integrates our proposed protocol. We will empirically evaluate and compare the performance of the enhanced protocol with other conventional distributed protocols (such as 2PL) in terms of QoS parameters (throughput, response time, and resource utilization), availability of the application, database consistency, and effect of locking on latency, among other factors.

In this research, we will first extend the current standard WS-Transaction Framework, and then develop an enhanced protocol (that can be deployed within the extended framework). The enhanced protocol will incorporate a mechanism for error-handling, concurrency control, consolidating multiple transaction calls into a single call, and secure tracing and reporting of business transactions for suspicious activities. These mechanisms will directly address issues such as QoS (throughput, response time, and resource utilization), availability of the application, database consistency, and effect of locking on latency, among others.

Thereafter, a hybrid methodology which incorporates service-oriented engineering and rapid application development will be used to develop a procurement system (which represents an interoperable service-oriented application) that integrates our enhance protocol. When the system is implemented, it will help to significantly improve transactional and security support for procurements systems and other interoperable SOA-based systems. In this research, the words request, message, process, transaction, operation are used interchangeable except explicitly stated.

2. BACKGROUND OF THE STUDY

Many organizations operate large enterprise web applications (such as web-service based applications), which involve distributed transactions that spans across different applications and resources. Apart from this, these applications involve related transactions (or task) that are loosely-coupled and carried out over long periods of time. Managing a transaction distributed across multiple systems significantly complicates an implementation to adhere strictly to the basic ACID tenets. For example, at the system level, isolation is usually implemented with locks. Distributed transactions that span across different applications and resources unavoidable involve a situation where one transaction locks the data record of another transaction. Even if, the transaction does not fail, resources might be locked longer than necessary and this will be unacceptable to most organizations. For performance reasons, so-called isolation levels allow this property to be relaxed for particular applications (Marina et al, 2006; Wenbing et al, 2006).

Traditional protocols such as strict two-phase locking protocol (2PL) and the two-phase locking protocol (2PC) are examples of protocols that are based on locking mechanisms (Bernstein et al, 1987; Ozsu and Valduriez, 1999). Thus, traditional ACID (Atomicity, Consistency, Isolation and Durability) properties of a transaction management system have to be relaxed for web services-based transactions. In particular, atomicity and isolation properties are usually relaxed in existing transaction protocols

in the web service environment; i.e. some activities in a transaction can commit their results before the whole transaction commits and the results of some activities can be seen before the whole transaction completes. However, several problems arise because of the relaxation of the ACID properties such that a readdressing of the transaction management problem for web services environment is required (Alrafai, 2006). First, transactional dependencies can emerge among independent transactions, which need to be addresses when compensation is needed in order to avoid inconsistency problems. To handle such dependencies, support for concurrency control mechanism is needed. In addition, when several concurrent transactions call web services running in different platforms , transactional support is required to ensure that the outcome of all transactions are consistent.
Assuming there is service failure due to the fact that a transactions or sub-transactions have failed. This situation points to the fact that there is need for a recovery mechanism to recover those transactions that have failed in the web services environment. The current specifications does not support these mechanisms at all, therefore there is a need to extend the specifications of the standard web service transaction protocol to provide an enhanced protocol for supporting the concurrency control mechanisms and other mechanisms(Alrafai et al, 2006). There is a clear justification for an enhanced protocol to manage concurrent transactions in a web service environment:

- Some extended protocols address only one part of the problem, such as the recovery problem (Cabrera et al[1, 2, 3], 05).

- Some protocols involve several transaction coordinators that communicate with each other to handle certain mechanisms such as transactional dependencies and other concurrency control problems [Choi, 2005; Haller et al, 2005).

- There are protocols that even handle some of these mechanisms but in different ways. For example, dependency graphs are use by some protocols to handle concurrency while others use transaction-wait for graph.

- There are some mechanisms that are very useful for proper management of concurrent transactions in web service environment but are usually overlooked by many extended protocols and in some cases these mechanisms may be implemented differently.

Again, when you consider that fact that in a distributed environment, transaction spans across different applications (Java EE, .NET and PHP) and resources (database and file systems), such an enhanced protocol is particularly needed to improve transactional and security support for asynchronous applications in a distributed environment. Most of the extended protocols such as the one proposed by Alrafai (2006) focused on introducing a mechanism for concurrency control. Wenbing (2005) proposed a protocol which was basically a reservation based extended transaction protocol aimed at addressing the limitations of compensation-based extended protocols. There are several other extended transaction protocols that target distributed transactions and web services transactions.

There are other very useful mechanisms that are not usually considered by many other extended protocols. An example of such a mechanism is the one that attempts to consolidate multiple transactions calls into a single call. Another example is the one that allows for secure reporting and tracing of transactions for suspicious activities. Table 1 provides a summary of some of these protocols by highlighting their main focus and shortcomings.
Our proposed protocol will incorporate: (1) error-handling mechanism that ensures the request either returns back, times out, is resubmitted, or an error is propagated up the invocation chain or there is a custom notification or callback; (2) concurrency control mechanism to enable detecting and managing conflicts that arise out of concurrent transactions that spans across multiple applications and resources; (3) mechanism to consolidate multiple transactions calls into a single call to reduce the number of operations within an atomic transaction; and (4) security control mechanism to address the challenge of handling a common security process logic, and secure reporting and tracing transactions for suspicious activities.

We will empirically evaluate and compare the performance of the proposed protocol with other conventional distributed protocols (such as 2PL) in terms of throughput, response time and resource utilization, among other factors.

3. DEFINITION OF RELATED TERMS

In this section, we will attempt to explain some of the key terms and concepts that are used in this research study.

Transaction: A transaction is a sequence of information that is treated as an individual unit and follows the "ACID" (atomicity, consistency, isolation and durability) test. A transaction must succeed or fail as a unit, following the atomic rule of "all or nothing." A transaction must be consistent, leaving both sides in a valid state. A transaction is isolated, unaware of or not seen by other concurrently executing transactions. And a transaction must be durable: once it succeeds it must persist (JNBridge, 2011).
A transaction is also defined as a unit of work that results in either success or failure and fulfils Atomic, Consistent, Isolated, and durable (ACID) principle (Marina et al, 2006).

Distributed transaction: A distributed transaction is the one that spans across different applications (Java EE, .NET and PHP) and resource (database and file systems). Managing a transaction distributed across multiple systems significantly complicates an implementation to adhere to the basic ACID tenets. Transactional support is very important in distributed transactions because of the need to preserve data integrity (Marina et al, 2006).

Interoperability: Laudati P. et al (2003) defines interoperability as the *ability to communicate or transfer data between functional units running on different platforms, implemented in different technologies, using industry standard or widely accepted data description and communication protocols.*
Interoperability as a process comes to play when we want to ensure that applications built on one platform connects to those created on the other. This research is not about interoperability; rather it is about providing support for interoperable SOA-applications in a distributed environment.
Services Oriented Architecture (SOA)
Services oriented architectures (SOA) are a way of developing distributed systems where the components of these systems are stand-alone services (or web services) (Sommerville, 2007).

Services as Reusable Software Components: Services are a natural development of software components where the component model is, in essence, the set of standards associated with web services (Adams, 2006; Sommervile, 2007).

A service can therefore be defined as a *loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.*

Services can be developed for reuse in a service-oriented application. It has much in common with component based development. The development of software using services is based on the idea that you compose and services to create new, composite services. These may be integrated with a web user interface to create a web application or may be used as components in some other service composition. The services involved in the composition may be specifically developed for the application may be business services developed within a company or may be services from some external provider (Sommervile, 2007). The implication of web services is that, in a real sense, the Web itself hosts your applications, with components scattered across multiple servers, your own and potentially many others, all working in concert(Adams, 2006). This is the approach that we will use to develop the procurement system that is used distributed transaction scenario.

Concurrency: Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other.
Because computations in a concurrent system can interact with each other while they are executing, the number of possible execution paths in the system can be extremely large, and the resulting outcome can be indeterminate. Concurrent use of shared resources can be a source of indeterminacy leading to issues such as deadlock, and starvation. A number of mathematical models have been developed for general concurrent computation including Petri nets, process calculi, the Parallel Random Access Machine model and the Actor model.

The design of concurrent systems often entails finding reliable techniques for coordinating their execution, data exchange, memory allocation, and execution scheduling to minimize response time and maximize throughput (Wikipedia, “concurrency”, 2011).

Concurrent Transactions: Sometimes it is useful for an application to have multiple independent connections called concurrent transactions. Using concurrent transactions, an application can connect to several databases at the same time, and can establish several distinct connections to the same database.

For example, suppose you are creating an application that allows a user to run SQL statements against one database, and keeps a log of the activities performed in a second database. Because the log must be kept up to date, it is necessary to issue a COMMIT statement after each update of the log, but you do not want the user's SQL statements affected by commits for the log. This is a perfect situation for concurrent transactions. In your application, create two contexts: one connects to the user's database and is used for all the user's SQL; the other connects to the log database and is used for updating the log. With this design, when you commit a change to the log database, you do not affect the user's current unit of work.

Another benefit of concurrent transactions is that if the work on the cursors in one connection is rolled back, it has no affect on the cursors in other connections. After the rollback in the one connection, both the work done and the cursor positions are still maintained in the other connections.
Lock: A lock is a system object associated with a shared resource such as a data item of an elementary type, a row in a database, or a page of memory. In a database, a lock on a database object (a data-access lock) may need to be acquired by a transaction before accessing the object. Correct use of locks prevents undesired, incorrect or inconsistent operations on shared resources by other concurrent transactions (Wikipedia, “Two-phase locking”, 2011).

Markov Model: In probability theory, a Markov model is a stochastic model that assumes the Markov property. The simplest Markov model is the Markov chain. A Markov chain is a mathematical system that undergoes transitions from one state to another (from a finite or countable number of possible states) in a chainlike manner. It is a random process characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property. Markov chains have many applications as statistical models of real-world

processes. The PageRank of a webpage as used by Google is defined by a Markov chain. Markov models have also been used to analyze web navigation behavior of users (Wikipedia, "Markov Chain", 2011).

4. RELATED WORK

Transactional and security support is important when talking about management of concurrent transactions in a distributed environment. Most research agrees that the traditional ACID transactions are unsuitable for application in distributed environment (especially web service transactions)(Alrafai et al, 2006; Wenbing et al, 2005, Marina et al, 2006; Reddy, 2003; Little and Freund, 2003).

Hence, there are a lot of protocols that have been proposed to address the limitations of traditional ACID transactions. Some of these protocols are still anchored on traditional ACID properties of a transaction management system, although with some relaxed properties while others are extensions to the standard web service transaction framework (or protocols). Alrafai(2006) address the problem by proposing an extension to the standard framework for web service transactions to enable detecting and handling transactional dependencies between concurrent business transactions. Thereafter, he then presented an optimistic protocol for concurrency control that can be deployed in a fully distributed fashion within the proposed framework. The main advantage of his approach to concurrency control is that is does not require any direct communication or information exchange between independent transactions. This approach has its cost because it requires two times the number of exchanged messages to reach globally correct execution; and so this is simply a trade-off relation between the cost of the number exchanged messages and the security privacy properties that can be ensured using it. Many protocols that support concurrency control work rely heavily on algorithms to detect and handle transactional dependencies between concurrent business transactions.

These algorithms are used to build transaction-wait-for-graph (TWFG) (directed graph whose nodes represent transactions, and arcs represents the wait-for relationships).
Performance studies indicate that a major component of cost of running the detection algorithms is wasteful (occurs in the absence of deadlock)(Krishna, 2003; Shyu et al, 1990; Sinha and Natarajan, 1985). Other examples of research work addressing the concurrency control problem in web services transaction is the work done by Haller et al; Choi et al, 2006. In all of these works, there is a general consensus that concurrency control should be handled by transaction coordinators who in turn maintain and update local partial views of the global serialization graph by direct communication. Several approaches have been used by different researchers to evaluate the performance of protocol/architectures of applications that run in a distributions environment. It is generally accepted that QoS related parameters are a key to evaluating the performance of these applications (Alrafai, 2006; Reddy, 2003; Marina et al, 2006; Wenbing, 2008).
We have discussed some of these approaches in the section on experimental evaluation, and will not be repeated here.

5. MOTIVATION OF THE STUDY

Modern large-scale enterprise applications encompass concurrent transactions with interleaving operations that cut across multiple platforms and resources and there is need to improve transactional and security support for these applications.

5.1 Motivating Scenario

The following example demonstrates the problem of managing transactions in a web service environment, and that motivated the need to improve transactional and security support. Figure 1 shows an overview of the features of the procurement system.

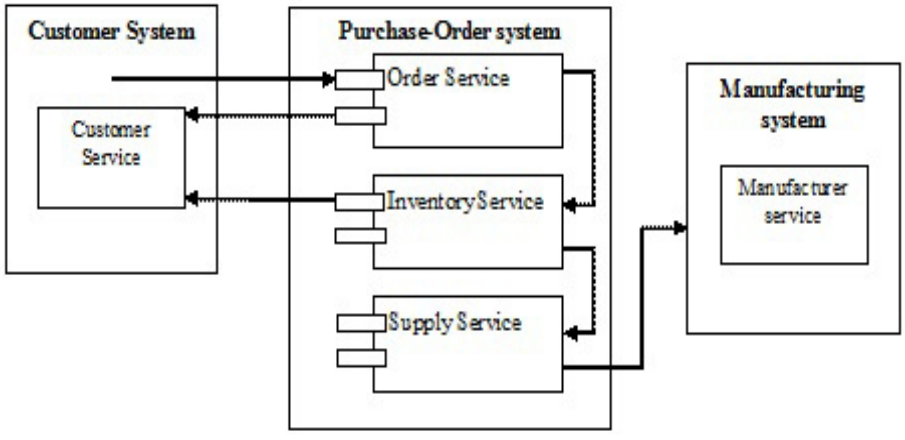

Fig. 1. Deployment Diagram for the Procurement System

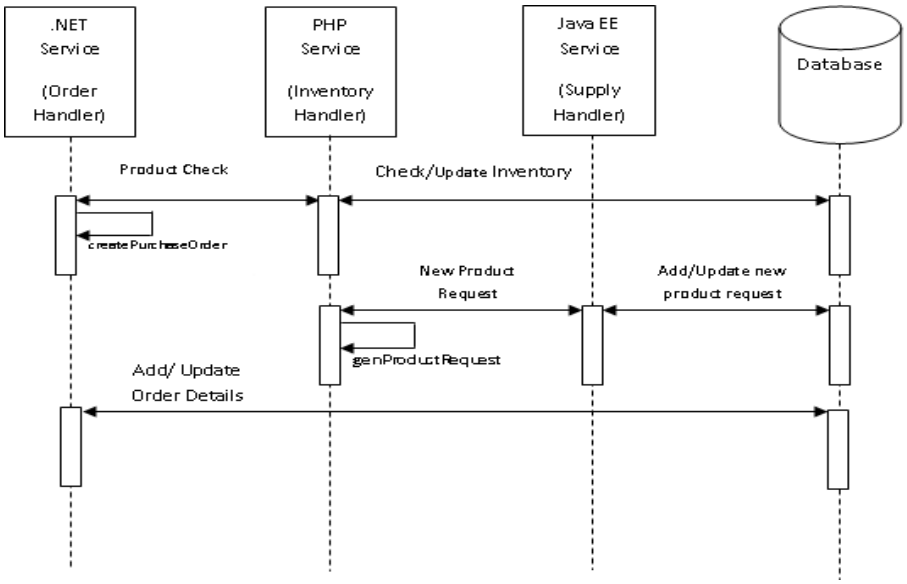

Fig. 2. An expended view of the Purchase-Order subsystem

Our distributed transaction scenario (or use case) is very simple and straightforward. It has the following features:
(i) The procurement system is an interoperable SOA-based application. There are three sub-systems within the procurement system – Customers system, Purchase-Order system and Manufacturing system.

(ii) The procurement system is modeled after a loosely coupled SOA architecture, where the Purchase-Order sub-system exists within its own organizational context and is implemented as tightly-coupled subsystem. This organization manages the purchase order system and is responsible for the operation of all its components.

(iii) Although the focus of the research is not on interoperability, this scenario also draws attention to the challenges of interoperability management in a distributed environment.

(iv) There are concurrent transactions that spans across different applications and resources, and for a transaction to be completed there may be different operations that are involved and each of these operations may be handled by the different subsystems.

(v) The purchase order system is trying to portray a short-lived transaction that is deployed within the boundaries of the same corporation. Again it is expected that the level of trust among the participants is fairly high. These are in fact, features of the WS-AtomicTransactions.

(vii) The purchase order subsystem is composed of three services(developed independently and specifically for the application): order services- implemented using .NET platform, the inventory service- implemented using implemented using the PHP platform, and the supply services- implemented using the Java EE platform. The purchase order subsystem (i.e. all the web services) is connected to a shared database -SQL Server database. The services could run on single or different computer systems with the same operating systems and application servers.

5.2 Description of the Distributed Transaction Scenario

Suppose there are two dependent processes (order service, inventory service) and one independent process (supply service) which have to collaborate in order to place an order for a product P1 in a procurement system. The product count for P1 must be above 500 units; otherwise a new product request is generated. The initial product count for P1 is 600 units. A customer initiates a transaction T1 to order for 500 units of P1. The order service sends a call to the inventory service to confirm if P1 is available. Once P1 is confirmed to be available, the inventory service updates its stock inventory and the purchase order is created. From time to time, the supply service checks the database to generate a new product request when the product count for P1 is below 500. This sort of process severally limits transactional capabilities across platform because a different service (platform) is adding and updating inventory while another independent platform (service) is generating a new product request.

Assuming that this order is later cancelled while T1 has not yet been committed (may be due to service failure or delay), and meanwhile the product count for P1 had already been decremented. It is possible for other concurrent transaction (T2, T3,…,Tn) to make an order based on the assumption that 600 units of P1 is still in stock. These transactions are not allowed in a procurement system, as it would result in potentially overselling P1. Again, assuming the purchase order is not cancelled but

there was an error (may be due to inventory service failure) or a delay that prevented an update on the product count of P1, request for product count information from other concurrent transactions will be denied (or locked) for a long time until service is restored and the supply service will not also be able to generate new product request.

Unaware that inventory service may be down or that P1 may be out of stock, several independent transactions (T1, T2, T3,…,Tn) may attempt to place order for P1. Many of those transactions would definitely fail and this would result in a situation where the database is left in an inconsistent state. When service is restored eventually, it will be very difficult to recover those transactions that have failed because no logging information was maintained. It will also be difficult to report and trace for suspicious activities due to the absence of a central logging mechanism in the system.

This scenario shows that order service depends on the inventory service, which verifies that the selected product is in stock. The inventory service also depends on the supply services, which verifies that the product count is low and hence a new product request is generated.

Current web service transaction standards do not address most of these problems - dependencies and recovery issues. There are no efficient models to express these problems and there are no efficient protocols to handle such problems.

6. STATEMENT OF PROBLEM

When using a shared database in a distributed environment, it is often difficult to connect concurrent business processes and software components running on disparate platforms into a single transaction; which severely limits transactional capabilities across platforms. Therefore one of the most serious challenges is how to improve transactional and security support for applications in a distributed environment. Technically, the standard framework for web services transaction still lacks mechanisms that can improve transactional support for interoperable applications, and so there is need to extend the standard framework for web service transactions to make provisions these mechanisms. The motivating scenario described in the previous section points the following specific problems:

1. Transactional and security capabilities of the system could be severely hampered due to concurrent multiple transaction calls. The number of failed transactions and locks contention involved in the system could build up if there are error-handling mechanisms to resolve transaction conflicts on time.
2. Resources could be locked or blocked for a long time when concurrent transactions are trying to access the resource.
3. The database might be left in an inconsistent state if the probability of failed transaction is high and there is no mechanism in the system to handle errors.
4. There is difficulty in recovering failed transactions in case of service failure, and also in reporting and tracing for suspicious activities due to the absence of a central logging mechanism in the system

7. RESEARCH QUESTIONS

The research questions that emanates from the foregoing are listed below:

1. What are the shortcomings of the standard WS-transaction protocol and other extended protocols; how do we extend it to improve transactional support; and what are the mechanisms for incorporation into the proposed extended protocol?
2. To what extent can the incorporation of a mechanism for error-handling, concurrency control, consolidating multiple transaction calls, and secure tracing improve transactional support for interoperable SOA-based systems?
3. How can we integrate our proposed protocol into a sample program to improve transactional support for interoperable SOA-based system?
4. Is there any significant difference between our proposed protocol (when integrated with an interoperable SOA application and a conventional application) and the conventional distributed 2PL?

7.1 RESEARCH HYPOTHESIS

Hypothesis 1:
Is there any significant difference between our proposed protocol and the conventional distributed 2PL in terms of improving transactional and security support for interoperable SOA-based applications?

Hypothesis 2:
Is there any significant difference between integrating our proposed protocol with an interoperable SOA-application and a normal SOA-based application?

8. AIMS AND OBJECTIVES OF THE STUDY

The aim of this research is to develop an enhanced protocol to that will significantly improve transactional and security support for asynchronous applications that need to interoperate in a web service environment. The specific objectives of the research are:

1. To improve the Quality of Service (QoS) and performance of the system in terms of response time, throughput and resource utilization.

2. To reduce the length of locking and blocking time involved before being able to access a resource.

3. To minimize the probability (and the number) of locks contentions, transaction conflicts, failed transactions, and multiple transaction calls in the system.

4. To reduce the probability that the databases (that stores data, logging and error information) might be left in an inconsistent state.

The specific objectives are:

1. To analyze and evaluate the current standard framework for web services transactions and other extended protocols with a view to identifying its problems.

2. To extend the standard framework for web service transaction protocol and then propose an enhanced protocol for use within the extended framework.

3. To develop and implement an enhanced protocol by integrating it with a sample program which represents an interoperable SOA-based application.

4. To evaluate and compare the performance of the enhanced protocol with other conventional distributed protocols (such as 2PL) in terms of QoS parameters (throughput, response time, and resource utilization), availability of the application, database consistency, and effect of locking on latency, among other factors.

9. PURPOSE AND SIGNIFICANCE OF THE STUDY

This research will provide a protocol that will help developers to either manage transactions in a web service environment or, better still design a system that does not suffer from problems associated with concurrent transactions in web service environment. This research study will benefit practicing web developers and companies in the following ways:
Practicing Web Developers: The research work will extremely be useful to practicing web developers because it will outline how to improve transactional and security support for transactions in web service environment.

Companies: Architects and IT managers in companies can get a brief overview of how this protocol is relevant to their operating environments and also how it can facilitate their architectural planning.

When the protocol is integrated into an interoperable SOA-based application, it will help in:

- Recovering transactions in case of service failure.
- Improving the Quality of Service (QoS) and performance of the system in terms of response time, throughput and resource utilization.
- Ensuring that the database is left in a consistent state after a distributed transaction.

10. SCOPE AND LIMITATION OF THE STUDY

This research assumes that concurrent transaction spans across different applications (.NET, Java EE, PHP) and resources (databases and file systems) and there is need for transactional support. For instance, you have an existing data repository that you need to access from more than one application running on multiple platforms. This research assumes that the transaction is short-lived (not a long living transaction) and is deployed in an intranet system (where the boundaries are within the same corporation). Therefore, it is expected that the level of trust among transaction participants is high. This research study uses a sample web application – a procurement system that is composed of web services.

This application is deployed in a distributed environment. The procurement system is only a sample use case implementation to demonstrate a distributed transaction scenario and does not illustrate recommended practice for implementing security in a production application. This application will be implemented first on a stand-alone computer and thereafter in an Intranet system.

Three platforms will involved in developing three different services that takes part in the procurement system - .NET implement the ordering service, Java EE implements the supply service, and PHP platform implements the inventory service.

Each of the service in the distributed transaction scenario may also be regarded as software components that reside on different machines. Within the proposed protocol, the web service should be seen as a means of support and not as an interoperability solution.

11. RESEARCH METHODOLOGY

The methodology to take in the research summaries into the following steps:

(a) The current standard protocol and other extended protocols will be analyzed and evaluated with a view to identifying its problems. These problems will be categorized into groups which will then translate into the mechanisms that will be incorporated into our proposed protocol.

(b) Extend the standard framework for web service transaction by first presenting the standard framework for web service transaction, and then highlight where and how our proposed protocol will be integrated into the framework. Thereafter we will develop a protocol for use within the extended framework.
We will present an algorithm that supports the protocol; give a formal definition of the protocol and a proof of the definition/algorithm for correctness.

(c) Develop a sample program (purchase order system) that integrates our proposed protocol. The sample program will be an interoperable SOA-based application. The following development tools will be used to develop the sample program

(i) Visual Studio 2010 IDE – for the .NET service
(ii) NetBeans 6.8 IDE – the Java service
(iii) XAMPP – for the PHP service
(iv) The database will be SQL Server 2010.

(d) Run experiments with the sample program that integrates our protocol in order to capture parameters. These parameters will be used to empirically evaluate and compare the performance of the proposed protocol with other conventional distributed protocols (such as 2PL) in terms of Quality of Service (QoS) parameters, availability of the SOA-based application, and consistency of the database and the effect of locking on latency, among other factors. In particular, we will use a discrete-time Markov model to estimate the probability that all of some large number of the transactions in the system complete successfully and also the probability that the database might have been left in an inconsistent state.

12. IDENTIFYING THE MECHANISMS FOR INCORPORATION INTO OUR PROPOSED PROTOCOL

Before making extensions to the standard web service transaction framework, it is vitally important to identify specific problems in the distributed transaction scenario. Thereafter, you then figure out a mechanism that will be incorporated into the proposed protocol to solve the problems. The problems that were pointed to by our motivating scenario can be grouped into four categories namely: error-handling, concurrency control, consolidating multiple transactions calls, secure reporting and tracing. These four categories translate into the mechanisms that will be incorporated into the proposed protocol. Let us now explain how each of these mechanisms fits into our motivating scenario. We will also explain the strategies for solving this problem.

(1) Mechanism for error-handling

In the Procurement System, the purchase order is being saved into the database. Once the purchase order transaction commits, all changes made to the data are permanent, and critical information will not be lost. If an error occurs prior to modifications committed to the database, changes should roll back, thus leaving the system in the original state. Being able to roll back or at least notify the errors to recover the system to its original state is an important part of the process. This is where an effective error-handling mechanism comes into play. An error-handling mechanism should be developed to ensure that the request either returns back, times out, is resubmitted, or an error is propagated up the invocation chain. There could also be a custom notification or callback mechanism.

Transaction systems do not usually give guarantee of consistency. In our distributed transaction system, if the product count does not decrement correctly, the supply processing system may not request products on time. In order to maintain consistency across the system by building this business requirement can be built into the error handling mechanism either at the application logic or by enforcing corresponding constraints at the database level.

(2) Mechanism for concurrency control

In the Procurement system example, when decrementing the products counter, a write lock should be set to avoid potentially overselling a product. An application should deny request for the product count information while this product count is being modified. Only after all changes are committed or rolled back should the data be available to the rest of the requests. This is

where an effective concurrency control mechanism comes into play to reduce the number of locks in the system and to also reduce the locking time involved before being able to access the resource. We will also try to include other strategies recommended by Kanjilal(2010) to handle concurrency.

These include:

- Reducing the length of time of the transaction (i.e. not having transactions that run for a long time).
- Performing certain operations at the end of a transaction instead during a transaction
- Avoiding user input that needs to commit during transactions
- Ensuring that all transactions are either committed or rollback after a specified time
- Ensuring that the resources are access in the same order
- Proper utilization of isolation levels to help minimize locking

(3) Mechanism for consolidating multiple transactions calls into a single call

In the Procurement System, there are situations where there could be periods where a very large number of users and/or transactions are either trying to use or are using the system. In other words, high data driven application usually have large number of concurrent users and operations. Because transactions in a concurrent system can interact with each other while they are executing, the number of possible execution paths in the system can be extremely large, and the resulting outcome can be indeterminate. Concurrent use of the shared resources (in this case, the database) can be a source of indeterminacy leading to issues such as deadlock and starvation. The solution will be to develop a mechanism to consolidate multiple transactions calls into a single call or try to reduce the number of transactions and sub-transactions within the system.

(4) Mechanism for secure reporting and tracing

In the Procurement system, the transactions spans across multiple platforms and resources and the system may suffer from security problems. Sometimes the security policy that is designed to address this problem may also be implemented at different levels. The challenge then is how to bring the security processing logic into a single component, and then implement it in a distributed environment.
Again, in our procurement system scenario, there is no way to know how many transactions have succeeded or failed, why they have failed, how it failed, when it failed, and at which terminal or machine. This is simply because no transaction logging information is maintained. The administrator needs this information to trace and report transactions for suspicious activities. It might also be very difficult to recover failed transactions in case of service failure if there is no central logging system.

12. CONCLUDING REMARKS

In this research, we will provide a protocol that significantly improves transactional and security for asynchronous applications in a distributed environment especially when concurrent transaction calls spans across multiple resources. When the protocol is integrated into an interoperable SOA-based application, it will help in recovering transactions in case of service failure, improving the Quality of Service (QoS) and performance of the system in terms of response time, throughput and resource utilization, and ensuring that the database is left in a consistent state after a distributed transaction.

REFERENCE

1. Adams, David.(2006): ASP.NET 2.0 Tutorial. Retrieved on January 30, 2008 at http://www.maconstateit.net/tutorials/ASPNET20/default.htm

2. Alrifai M, Dolog P, Balke W, Nejdl W. (2009): Distributed Management of Concurrent Web Service Transactions. IEEE Transactions on Services Computing, vol. 2, no. 4, pp. 289-302, October-December, 2009.

3. Alrifai M., Dolog P., and Nejdi W. (2006): "Transactions Concurrency Control in Web Service Environment," *Web Services, European Conference on*, pp. 109-118, Fourth IEEE European Conference on Web Services (ECOWS'06), 2006.

4. Cabrera L,F; Copeland, G; Feingold, M; Freund, R.W; Freund, T; Johnson, J; Sean Joyce, S; Kaler, Chris; Klein, J; Langworthy, D; Little, M; Nadalin, A; Newcomer E; Orchard, D; Ian Robinson, I; Shewchuk, J; Tony Storey, T(2005): Web Services Atomic Transaction (WS-AtomicTransaction). Retrieved on October 31, 2011 from http://specs.xmlsoap.org/ws/2004/10/wsat/wsat.pdf

5. Cabrera L,F; Copeland, G; Feingold, M; Freund, R.W; Freund, T; Johnson, J; Sean Joyce, S; Kaler, Chris; Klein, J; Langworthy, D; Little, M; Nadalin, A; Newcomer E; Orchard, D; Ian Robinson, I; Shewchuk, J; Tony Storey, T(2005): Web Services Business Activity Framework (WS-BusinessActivity). Retrieved on October 31, 2011 from http://specs.xmlsoap.org/ws/2004/10/wsba/wsba.pdf

6. Cabrera L,F; Copeland, G; Feingold, M; Freund, R.W; Freund, T; Johnson, J; Sean Joyce, S; Kaler, Chris; Klein, J;

Langworthy, D; Little, M; Nadalin, A; Newcomer E; Orchard, D; Ian Robinson, I; Shewchuk, J; Tony Storey, T(2005): Web Services Coordination (WS-Coordination). Retrieved on October 31, 2011 from http://specs.xmlsoap.org/ws/2004/10/wscoor/wscoor.pdf

7. Cabrera L,F; Copeland, G; Feingold, M; Freund, R.W; Freund, T; Johnson, J; Sean Joyce, S; Kaler, Chris; Klein, J; Langworthy, D; Little, M; Nadalin, A; Newcomer E; Orchard, D; Ian Robinson, I; Shewchuk, J; Tony Storey, T(2003): *Web services composite application framework* (ws-caf), 2003, published at http://developers.sun.com/techtopics/webservices

8. Böttcher, S.; Gruenwald, L. and Obermeier, S.(2006): Reducing Sub-transaction Aborts and Blocking Time Within Atomic Commit Protocols.

9. Dhawan, P(2002): Microsoft Developer Network. Performance Comparison: .NET Remoting vs. ASP.NET Web Services. Retrieved on October 28, 2011 from http://msdn.microsoft.com/en-us/library/ms978411.aspx

10. G. Weikum(1991): "Principles and realization strategies of multilevel transaction management," *ACM Transactions on Database Systems*, vol. 16, no. 1, 1991, pp. 132-180.

11. Gabhart K. (2004): Java/.NET Interoperability via Shared Databases and Enterprise Messaging. Retrieved on January 31, 2011 from http://www.devx.com/interop/Article/19952/0/page/2.

12. Guest S. (2004): Microsoft .Net And Java: Achieving Interoperability. Devx.Com. Retrieved On January 4, 2010 From http://www.devx.com/interop/article/19928/0/page/1

13. H. Garcia-Molina and K. Salem, "Sagas," *Proceedings of the ACM SIGMOD Conference*, San Francisco, CA, 1987, pp. 249-259

14. Hogg, K; Chilcott, P.; Nolan, M.; and Srinivasan, B.(2004): An Evaluation of Web service in the design of B2B Application. 27th Australasian Computer Science Conference, The university of Otago, Dunedin, New Zealand. Conferences in Research and Practice in Information Technology, Vol 26. V. Estivill-Castro, Ed.

15. JnBridge(2011): .NET-to-Java and Java-to-.NET Cross-platform Transactions with 2-Phase Commit. JNBridge LLC., Boulder, Colorado, USA. Retrieved on February 2, 2011 from http://www.jnbridge.com/jnbpro-transactions.htm.

16. K. Haller, H. Schuldt, and C. Türker. *Decentralized coordination of transactional processes in peer to peer environments*. ACM Press, in Proc. of the 14th ACM Intl. Conference on Information and Knowledge Management (CIKM 2005), pages 36--43, Bremen, Germany, Nov. 2005.

17. Kanjilal J. (2010): Concurrency handling techniques in ADO.NET. Retrieved on January 14, 2011 from http://www.developerfusion.com/article/84418/concurrency-handling-techniques-in-adonet/

18. Knifsend F; Louis R; Nilesh Junnarkar, tugdual Grall, Heidi Buelow, Clemen Utschig, Sachin A. Agarwal, Jesus Rodriguez(2006): Interoperating J2EE and Microsoft.NET Applications – What Standards to Adopt? An Oracle White paper. Oracle.com.

19. Laudati P.; Loeffler W.;, David Aiken, Arkitec, Keith Organ, Arkitec, Anthony Steven, Mike Preradovic, Wayne Citrin, Peter Clift,(2003): Application Interoperability: Microsoft .NET and J2EE. Microsoft Corporation.

20. Little, M And Freund, T(2003): A Comparison Of Web Services Transaction Protocols: A Comparative Analysis Of Ws-C/Ws-Tx And Oasis Btp. .Retrieved On November 1, 2011 From Http://Www.Ibm.Com/Developerworks/Webservices/Library/Ws-Comproto/

21. Little, M(2006): A history of Extended Transactions. Extended transaction models for business process management: from CORBA to web services. Retrieved on November 1, 2011 from http://www.infoq.com/articles/history-of-extended-transactions

22. Majumdar, B; Mysore, U; Sahoo, L; Saxena, S. (2007): Load Testing Web Services. SYS-CON Media, Inc. SOA Web Services Journal. Retrieved on October 28, 2011 from Majumdar, B; Mysore, U; Sahoo, L; Saxena, S. (2007): Load Testing Web Services. SYS-CON Media, Inc. SOA Web Services Journal. Retrieved on October 28, 2011 from http://soa.sys-con.com/node/284564

23. Mark Little (2002): Web services transactions: past, present and future. Arjuna.com. Retrieved on October 31, 2011 from citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123...

24. Microsoft Encarta Dictionaries(2007): Microsoft Encarta and Student Program Manager, One Microsoft Way, Redmond, WA 98052-6399,U.S.A. Microsoft Corporation

25. Mitchell, B.(2011): QoS. About.com Guide. Retrieved on October 29, 2011 from http://compnetworking.about.com/od/networkdesign/g/bldef_qos.htm

26. MSDN Library(2010): ASP.NET 4. MSDN Library. Microsoft cooperation. Retrieved on October 19,2010 from http://msdn.microsoft.com/en-us/library/ee532866.aspx

27. OASIS *Business transaction protocol(2004):* Retrieved on October 31, 2011 from http://oasis-open.org/committees/download.php/2077/BTP_Primer_v1.0.20020605.pdf

28. OASIS Web Service Business Activity (WS-BusinessActivity) http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.pdf

29. OASIS Web Service Coordination (WS-Coordination), http://docs.oasis-open.org/ws-tx/wscoor/2006/06

30. OASIS(2007): Web Service Atomic Transaction (WS-AtomicTransaction). Retrieved on November 1, 2011 from http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf

31. Paul, D., Henskens, F. and Hannaford, M. (2007): Isolation and Web Services Transactions. Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE Computer Society.

32. Reddy P. and Bhalla S. (2003): Asynchronous operations in distribute Concurrency Control. IEEE Transactions on Knowledge and Data engineering. Vol. 15, No.3. Retrieved on February 14, 2011 from http://www.bme.ogi.edu/~hayest/cse541/Readings/reddy-distributed-concurrency-control-2003.pdf

33. S. Choi, H. Jang, H. Kim, J. Kim, S. Kim, J. Song, and Y. Lee. *Maintaining consistency under isolation relaxation of web services transactions*. In Proc. of WISE 2005, New York, USA, Nov. 2005.

34. Shyu S. C; Li, V.O.K. and Weng, C. P. (1990): "Performance Analysis of Static Locking in Distributed Database Systems," IEEE Trans. Computers, vol. 39, no. 6, pp. 741-751, June 1990.

35. Sinha, M.K and Natarajan, N.(1985): "A Priority Based Distributed Deadlock Detection Algorithm," IEEE Trans. Software Eng., vol. 11, pp. 67-80, Jan. 1985.

36. Sommerville, I. (2007): Software Engineering(Eight Edition).Pearson Education Limited, Harlow, England.

37. Tan, K, and Mustapha S. (2009): Measuring Availability of Mobile Web Services. Asian Research center, British communication telecommunication.

38. Wan Nurhayati, Rahman Wan AB, Meziane Farid(2008): Challenges to Describe QoS Requirement for Web Services Quality Prediction to Support Web Services Interoperability in Electronic commerce. Communications of the IBIMA, Volume 2008.

39. Wenbing Zhao, L. E. Moser and P. M. Melliar-Smith (2005): A reservation-based extended Transaction protocol. *Proceedings of the IEEE International Conference on Web Services*, Orlando, FL, July 2005.

40. Wenbing Zhao, L. E. Moser and P. M. Melliar-Smith (2005): Unification of transactions and replication in three-tier architectures based on CORBA. *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, 2005, pp. 20-33.

41. Wenbing Zhao, *Moser* L. E., and P. M. Melliar-Smith(2006): A Reservation-Based Extended Transaction Protocol, IEEE Transactions On Parallel And Distributed Systems 1.

42. Wikipedia (2010): Concurrency (computer science). Wikimedia Inc. Retrieved on May 1, 2010 from http://en.wikipedia.org/wiki/concurrency

43. Wikipedia (2010): Concurrency control. Wikimedia Inc. Retrieved on May 1, 2010 from http://en.wikipedia.org/wiki/concurrency control

44. Wikipedia (2010): Service-oriented architecture. Wikimedia Inc. Retrieved on May 1, 2010 from http://en.wikipedia.org/wiki/Service-oriented_architecture

45. Wikipedia (2011): Quality of Service. Wikimedia Inc. Retrieved on October 29, 2011 from http://en.wikipedia.org/wiki/Quality_of_service

46. Wikipedia(2011): Markov Chain. Wikimedia Foundation Inc. Retrieved on November 1, 2011 from http://en.wikipedia.org/wiki/Markov_chain

47. Wikipedia(2011): Two-phase commit protocol. Wikimedia Foundation Inc. Retrieved on November 1, 2011 from http://en.wikipedia.org/wiki/Two-phase_commit_protocol

48. Wikipedia(2011): Two-phase locking. Wikimedia Foundation Inc. Retrieved on November 1, 2011 from http://en.wikipedia.org/wiki/Two-phase_locking

49. Wohlstadter E.; Tai, S.; Mikalsen, T.; Diament, J.; Rouvellou, I.(2006): A service-oriented Middleware for Runtime Web Services Interoperability. International conference on web services(ICWS'06) , 2006.

50. Suninter(2005): Sun Facilitates Interoperability Between Java Technology and .NET Via Open Source Web Services Implementation. www.sun .com/smi /Press/sunflash/2005-11/sunflas.20051104.1.html

51. Schafer, M.; Dolog P., and Nejdi W. (2006): An Environment for Flexible Advanced Compensation of Web Services Transactions. ICWE 2007.

52. Riegen, M.; Husemann, M.; Fink, S. and Norbert Ritter, N. (2007): Rule-Based Coordination of Distributed Web Service Transactions. *IEEE Transactions on Services Computing*, vol. 99, no. 2, pp. 60-72, , 5555.

53. Prithwish Kangsabanik, P.; Yadav D. S.; Mall R.; Majumdar A. K.(2006): Performance Analysis of long-lived cooperative transactions in active DBMS. Data and Knowledge Engineering. Available online at Sciencedirect.com

54. Munoz-Escoi, F. D.; Pla, J.; Ruiz-Fuertes, M.I.; Irun-Briz, L.; Decker, H.; Armendaiz-Inigo J.E. and Gonzalez de Mendivil, J. R. (2003): Managing Transaction Conflicts in Middleware-based Database Replication Architectures.

55. Obermeier,S.; Böttcher, S.; Hett, M.; Chrysanthis, P. and George Samaras(2006):

56. Blocking reduction for distributed transaction processing within MANETs

57. Paul, D., Henskens, F. and Hannaford, M. (2007): Isolation and Web Services Transactions. Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE Computer Society.

58. Prithwish Kangsabanik, P.; Yadav D. S.; Mall R.; Majumdar A. K.(2006): Performance Analysis of long-lived cooperative transactions in active DBMS. Data and Knowledge Engineering. Available online at Sciencedirect.com
59. Riegen, M.; Husemann, M.; Fink, S. and Norbert Ritter, N. (2007): Rule-Based Coordination of Distributed Web Service Transactions. *IEEE Transactions on Services Computing*, vol. 99, no. 2, pp. 60-72, , 5555.

60. Schafer, M.; Dolog P., and Nejdi W. (2006): An Environment for Flexible Advanced Compensation of Web Services Transactions. ICWE 2007.

61. M. Madiajagan, and B. Vijayakumar(2006): Interoperability in Component Based Software Development. World Academy of Science, Engineering and Technology 22 2006.

Author's Bio

Laud Charles Ochei holds a Bachelor's degree and Master's degree from University of Uyo and University of Benin in Nigeria respectively. He is presenty on a doctoral programme at the Department of computer Science, University of Benin, Benin City, Nigeria. his PhD at the University of Benin where he researches into enhancing supports for inter-operable service-oriented applications. Currently a lecturer at the University of Port Harcourt, he is also engaged in a number of community service and youth development.