Architectural Framework for Improving QoS of Service for Interoperable Service-Oriented Systems

[1]Ochei, L.C.; [2]Ogunsakin, R.; [3]Wobidi, Echebiri
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria.
E-mail: [1]laud.ochei@gmail.com; [2]rotimi.ogunsakin@gmail.com; [3]echebs62@yahoo.com

ABSTRACT

This paper presents an architectural framework for analyzing the performance of interoperable service-oriented systems. Previous research has focused on the architectures for evaluating the performance of service-oriented systems, and there are serious limitations in the transactional capabilities of these architectures. This paper proposes an enhanced architectural framework for improving transactional and security support for interoperable SOA-based systems and improve the quality of service of the system. First, we present two traditional web service transactions framework. Second, we present our proposed transaction model and show how it maps into the traditional frameworks. Third, we show how to extend the traditional web service transaction models to incorporate interoperable SOA-based systems using a procurement system as a case study. Extensive evaluations using architectural analysis shows that the architecture satisfies the analysis goals, and system requirements. Thus, the proposed architecture is recommended for system designers and developers to enhance transactional support for interoperable service-oriented systems.

Keywords: Architecture, Interoperability, Service-oriented system, cloud deployment, performance

CISDI Journal Reference Format
Ochei, L.C., Ogunsakin, R., Wobidi, Echebiri (2020): Architectural Framework for Improving QoS of Service for Interoperable Service-Oriented systems. Computing, Information Systems, Development Informatics & Allied Research Journal. Vol 11 No 2, Pp 149-170
Available online at www.isteams.net.cisdijournal

1. INTRODUCTION

A large variety of large-scale enterprise transactions systems exist on the internet, ranging from web servers to e-mail servers, streaming media services, and e-commerce servers. Most of these systems are built with web services, using the classical client-server architecture, where multiple clients concurrently access these services. Under this situation, such systems are bound to face varying workloads, where for instance, an e-commerce system may experience a heavier workload at peak hours of usage (Arlitt and Jin, 1999). In the extreme form, a heavy workload may induce server thrashing and service unavailability, with underlying economic costs (Malrait et al, 2009). These costs are estimated at up to US$2.0 million/hours for Telecoms and Financial companies. Apart from the resulting cost implication, the transactional and security capabilities of the systems are severely limited due to concurrent transactions with interleaving operations that span across different applications and shared resources.

Quality of service (QoS) is important in improving transactional and security support for business operations in a large enterprise transaction system. Monitoring the QoS across SOA-based systems is very challenging because software and hardware components of the system are prone to errors [Ref]. Nevertheless, setting up this infrastructure is critical to reduce and prevent system downtime.

Eliminating application downtime poses a significant challenge for IT organizations. This is particularly true for applications that must provide continuous or near-continuous operations to users who increasingly expect uninterrupted availability of online services. Any outage of an application or website, even if that outage is scheduled or planned, has an impact on the revenue and reputation of the business. Put simply, downtime equals lost revenue. For example, the Computer Security Institute reported a worldwide financial loss and business disruptions of circa (equivalent to US$130 million loss), that resulted from viruses, unauthorized access, and theft of proprietary information in 2005 (Computer Security Institute, 2005).

Motivated by the challenges highlighted above, this paper presents an architectural framework to provide transactional and security support for interoperable service-oriented systems. Transactional and security support is vitally important, first, to ensure data integrity when managing transactions in a distributed environment (Marina et al, 2006) and also to improve performance and quality of service of the system.

The main contributions of the paper are:
(i) an architecture for integrating the design model into a web service distributed transaction scenario to improve transactional and security support for the system.
(ii) novel distributed transactional and security support (DTSS) Algorithm for enhancing transactional support for an interoperable SOA-based system.

The rest of the paper is organized as follows: Section 3 discusses the transactional and security support for interoperable SOA-based systems. Section 4 presents a high-level model of the proposed system. Section 5 presents extensions to existing frameworks for web service transactions. Section 6 is the evaluation of the architecture. Section 7 concludes the paper with recommendations for future work.

2. TRANSACTIONAL AND SECURITY SUPPORT FOR INTEROPERABLE SOA-BASED SYSTEM

This section presents a scenario which captures a transactional and security for interoperable SOA-based systems.

2.1. Motivating Scenario

The following motivating scenario is adapted from our previous work (Ochei, 2012). The scenario demonstrates the problem of managing interoperable service-oriented systems in a distributed transaction scenario and motivates the need to enhance transactional and security support. Figure 1 shows an overview of the features of a procurement system (which represents an interoperable SOA-based system), while Figure 2 shows an expanded view of the purchase-order sub-system.

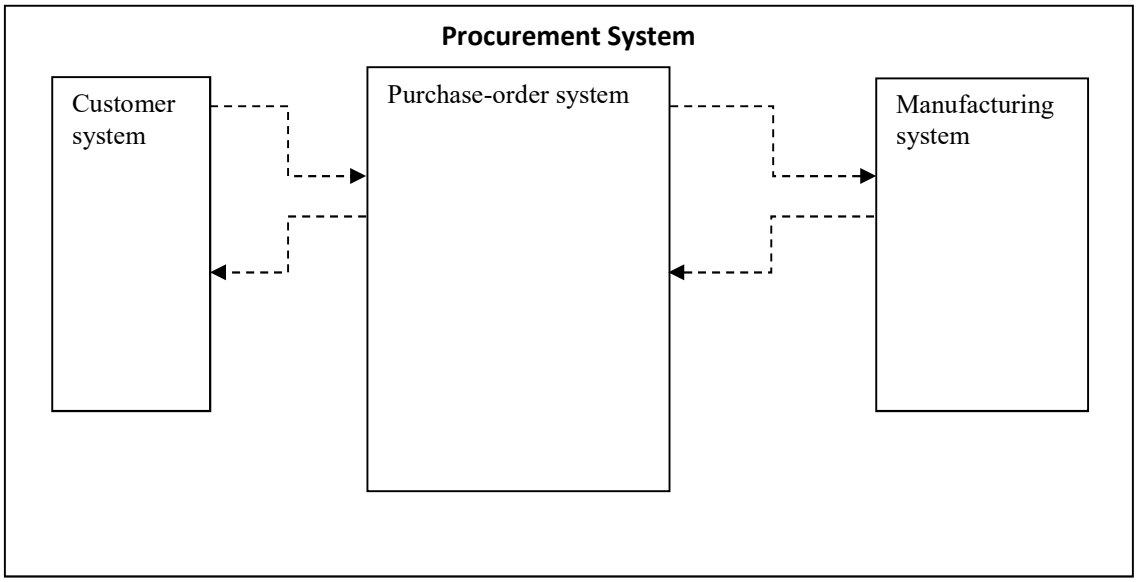

Figure 1: Deployment Diagram for the Procurement System

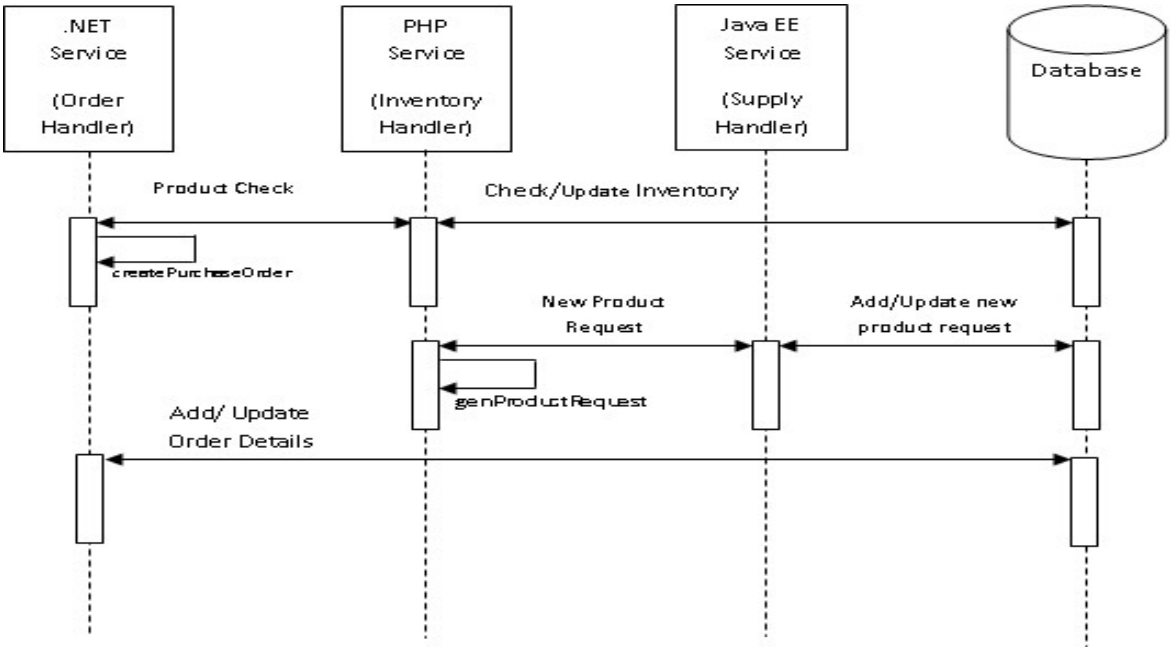

Figure 2: An expended view of the Purchase-Order sub-systems (sequence diagram for message flow between the three service handlers)

Our distributed transaction scenario 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 modelled after a loosely coupled SOA architecture, where the Purchase-Order sub-system exists within its organizational context and is implemented as a tightly coupled subsystem. This organization manages the purchase order system and is responsible for the operation of all its components.
iii. Although, the above scenario does not demonstrate how to make two or more systems to interoperate, instead it draws attention to the challenges of interoperability management in a distributed environment.
iv. There are concurrent transactions that span 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 figure above (Figure 1 and Figure 2) assumes that a transaction is used to coordinate multiple operations that span across multiple platforms and control the commit or abort of any changes submitted to the underlying data source.
vi. 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-Atomic Transactions.
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 the PHP platform, and the supply services- implemented using the Java EE platform. The purchase order sub-system (i.e. all the web services) is connected to a shared database – the SQL Server database. The services could run on single or different computer systems with the same operating systems and application servers.
viii. The databases are shared by two or more components within the system.

2.2.1 Description of the Distributed Transaction Scenario

In the diagram shown in Figure 2, there are two dependent processes (order service and inventory service), and one independent process (supply service) which have to collaborate to place an order for a product P1 in a procurement system. 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. Immediately after updating the stock inventory, the supply service checks and compares the current product count of P1 with the reorder level of that P1. A new product request is immediately generated if the product count for P1 is below 500.

This sort of process severely limits transactional capabilities across platforms because a different service (platform) is adding and updating inventory while another independent service (platform) is accessing the changed inventory before 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 types of 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. It will also lead to very low quality of service (i.e., low throughput and high response time, system unavailability) guarantee to customers. 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 trace and report transactions 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 service, which generates a new product request when the product count is low. Current web service transaction standards do not address most of these problems - dependencies and recovery, and error-handling issues and low quality of service. There are no efficient models to express and also handle these problems. A robust architecture is required for this interoperable SOA-based system where there is a need for enhancing transactional and security support.

2.2 Challenges of the system

The motivating scenario described in the previous section points the following specific problems:

I. Many useful mechanisms for improving transactional support and security are either excluded or poorly implemented in the current web service transaction frameworks and other transaction models.
II. Most proposed frameworks and models do not express and address transactional and security support issues – dependencies, recovery, error-control, database consistency, availability, and secure tracking and reporting of suspicious transactions. Some proposed models cannot also be deployed within existing frameworks for web service transactions.
III. Transaction conflicts occur leading to failed transactions, locks contentions that could last for a long time.
IV. The database might be left in an inconsistent state.
V. Difficulty in recovering failed transactions in case of service failure.
VI. Difficulty in reporting and tracing transactions for suspicious activities.

2.3. Distributed Transactional Model Based on A Multilayered Architecture

In this section, we introduce an architecture that will sustain our enhanced model for improving transactional and security support for interoperable SOA-based systems. Figure 3 below shows the conceptual view of this multi-layered architecture.

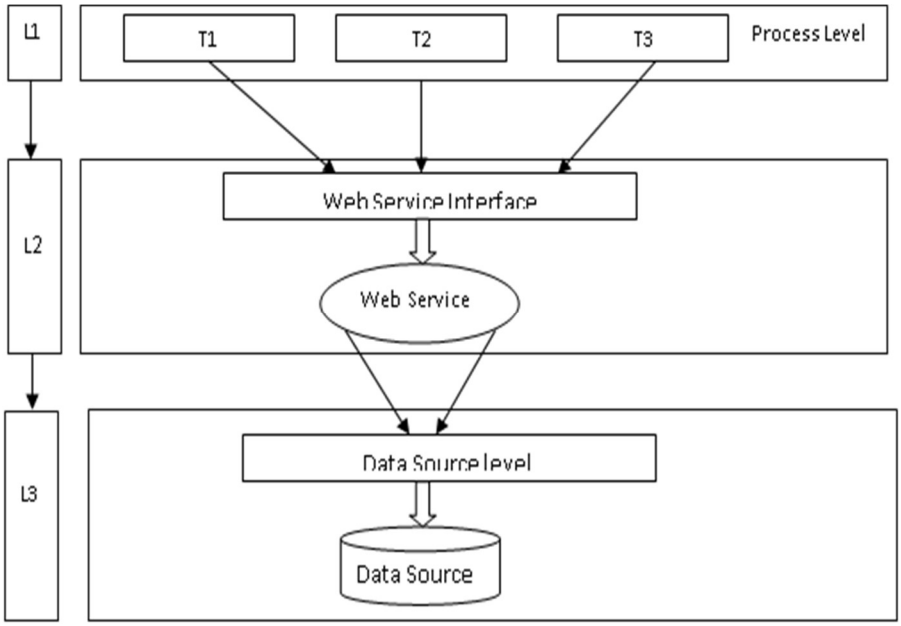

Figure 3. The multi-layered Architecture (Adapted from Alrafai, 2006)

The proposed multi-layered architecture can be modeled with the multilevel nested transaction model from (Weikum, 1991). This model fits into the multi-layered architectures where each layer has its own level-specific semantics of the set of operations. This model is a special case of the open nested transaction model with the requirement that all leaf nodes in a transaction tree correspond to operations at the particular level of abstraction, where the edges represent the implementation of each operation at level Li by a sequence of operations at the next lower level Li-1 (for i=1,…, n in bottom-up order of a n-level system). In the architecture shown in Figure 3, we have a 3-level system (L1= resource level, L2=service level and L3=process level).

The three levels in our scenario are as follows:

I. Process level - A set of business processes or activities that calls the procurement system.
II. Web service interface level- A set of web service operations that provides access to products available for customers to order.
III. Data source level -A database that stores and manages products

We assume that each process on the process level is mapped to one web service operation (e.g. check product availability/ generate new product request) on the web service interface level which is in turn mapped to a database transaction on the data source level. The consistency of the overall system can only be guaranteed, if the produced schedule at each level is guaranteed to be serializable (i.e., equivalent to some serial execution of the involved transactions) (Weikum, 1991). In this paper we restrict our focus on process level transactions with Web service level operations as elementary operations of these transactions.

2.3 Review of Related Works

There are several related research work that explores existing and proposed transactional model to manage web service transactions. Most research agrees that the traditional ACID transactions are unsuitable for application in a 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 and models 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.

Several works evaluate the performance of applications that implement these web service transaction frameworks. 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). Alrifai et al (2006) proposed an extension to the standard web service transaction framework to support concurrency control at the service level, and he also performed an experimental evaluation using simulation to validate that his proposed framework outperforms the conventional distributed 2PL in terms of response time and throughput. The performance was measured in terms of the average response time and overall throughput of the system.

Another approach for performance evaluation was done by Reddy et al (2003) where he aimed to study the effect of introducing delays in a distributed transaction. Reddy in his work set some parameters for execution that are consistent with the performance of the conventional two-phase locking method. The simulation result of the number of aborts (Na) and the total number of transactions (Nt) was measured, and then the Mean of Missing Rate (MMR) defined as the ratio of aborts over the total number of transactions was computed. A low value of MMR points to good system performance for comparisons. (Reddy et al, 2003).

Our research differs from the above works cited in many respects. Most of the above works cited above have addressed the issue of transactional support for web service transactions by extending the current standard framework for web service through incorporating certain mechanisms that will address a particular aspect such as concurrency control and transaction dependencies. In this research, we proposed a generic transactional model, and then show how it can be deployed within any framework for web service transactions. We also consider other useful mechanisms such as mechanism for improving QoS as well as a mechanism for secure tracking and reporting of transactions for suspicious activities.

3. ANALYSIS OF THE EXISTING SYSTEM

Two architectures will serve as our existing system. The first is the architecture of the standard web service transaction framework (as described by OASIS specifications) while the second will be the conventional enterprise transaction system built using web services. These architectures are themselves frameworks that can be extended to accommodate our enhanced model for improving transactional and security support for interoperable SOA-based systems.

3.1 Web Service Transaction Framework

In 2007, OASIS, the international standards consortium, announced that its members have approved Web Services Transaction (WS-Transaction) version 1.1 as an OASIS Standard, a status that signifies the highest level of ratification. More than 20 organizations collaborated to define the protocol framework for coordinating distributed application actions.

WS-Transaction describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols can be used to support a wide variety of applications that require consistent agreement on the outcome of distributed transactions. WS-Transaction is offered on a Royalty-Free basis, as provided under OASIS policies. The WS-Transaction OASIS Standard comprises three specifications: WS-Coordination; WS-AtomicTransaction; and WS-BusinessActivity. WS-Coordination enables an application service to create the context necessary for propagating an activity to other services. WS-Coordination defines two key concepts (see Figure 4):

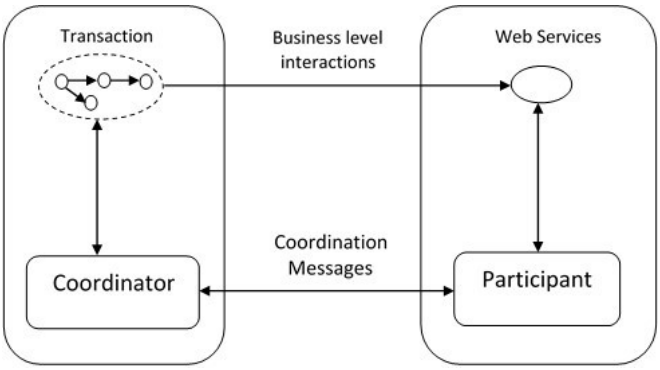

Figure 4. Standard Web Service Transaction Framework (Alrafai, 2006)

(i) The Coordinator
The coordinator is responsible for creating the context and coordinating the different partners according to the applied protocol. The coordinator role can be taken by the initiator of a distributed application or by a (trusted) third party.

(ii) The Participant
The participant is an entity that resides on the Web service provider side and represents an instance of the Web service that has been invoked within the distributed application. This entity is responsible for communicating with the coordinator according to the applied protocol on behalf of the Web service. Transaction protocol's messages can be exchanged and coordinated within this framework.

WS-AtomicTransaction defines agreement protocols for short-lived activities having the all-or-nothing property, and WS-BusinessActivity defines protocols for long-running transactions that require compensation-based agreement. Working together, these specifications enable existing transaction processing, workflow, and other systems to hide their proprietary protocols and operate in a heterogeneous environment. In this paper, we shall refer to these specifications as WS Transaction framework (the de-facto standard for web service transaction). Many issues were not addressed in the WS-Transaction framework and therefore several authors have proposed extensions to this framework to support, for example, concurrency control (Alrafai, 2006) as shown in Figure 5.

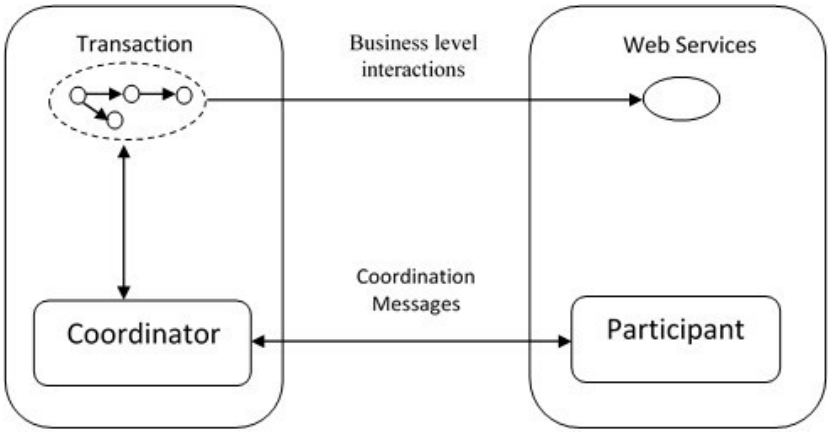

Figure 5. Web service transaction framework (adapted from Alrafai, 2006)

There are other frameworks in the web services domain. These include OASIS Web Services Composite Application Framework (WS-CAF) TC. However, we shall deal only with the web service transaction framework. In this study, we will also be proposing an extension to the web service transaction framework for improving transactional and security support (OASIS, 2012).

3.2 Conventional Enterprise Web Service Transaction Architecture

There are several enterprise transaction systems built using web services and which may not strictly conform to the web service transaction framework as described by OASIS specifications.

A typical enterprise transaction system built using Web services is shown in Figure 6. Application servers execute business logic and request database operations and are usually placed on an internal network separate from the Web servers for security purposes. A set of Web servers running (IIS and ASP.NET Development Server) accepts incoming requests and use a queuing resource manager such as Microsoft® Message Queue (MSMQ) to deliver them to application servers for processing. Client applications use a Web Services Description Language (WSDL)-generated proxy in order to submit operations to the Web server (Connolly, 2004).

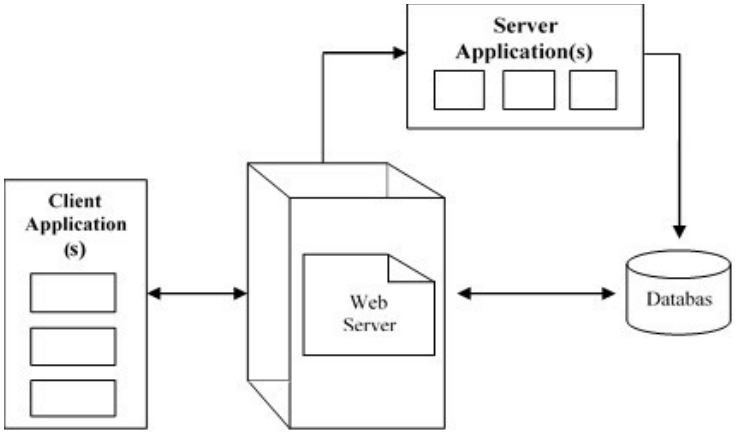

Figure 6. A Typical Web Services Transaction Architecture (Connoly, 2004)

Most client transactions will consist of multiple SOAP requests. SOAP is a wire protocol, a way to transfer data between distributed systems using XML technologies and protocols such as HTTP. For example, a financial market trade might consist of a SOAP request to submit an order and a subsequent request to check for its execution. Or a product purchase might consist of one request to submit an order, a second request to submit credit data, and a third request to obtain an order number for a completed order.

3.2 Challenges of the Existing Architectures

In this section we discuss the weaknesses that are inherent in existing frameworks for web service transaction, and hence raise the need for introducing various mechanisms that will enhance web service transactions. The WS-BusinessActivity coordination type (which is based on the open nested transaction model) relaxes the isolation property. It permits disclosing intermediate results by autonomous participants instead of locking local resources until the end of the (global) transaction. In the case of transaction abort, the effects of already committed sub-transactions are undone by means of compensating sub-transactions. When the number of transactions having access to intermediate results increases, the compensation of some operations becomes either too expensive or even impossible, and so the assumption that all service operations can always be compensated is not realistic (Alrifai, 2006).

This raises the need to a concurrency control mechanism for Web service transactions. Several extensions to the standard web service framework have been proposed and implemented by several authors including Alrifai(2006) to support control. Most concurrency control algorithms rely on building and maintaining dependency graphs in order to detect deadlocks in a distributed environment. However, performance studies indicate that a major component of cost of running the detection algorithms is wasteful (occurs in the absence of deadlock) especially when asynchronous operations are involved (Reddy, 2003; Shyu et al, 1990; Sinha and Natarajan, 1985).

Our motivating scenario pointed to several mechanisms that can be introduced into various frameworks for web service transactions. Let us now explain how each of these mechanisms fits into our motivating scenario and strategies for solving this problem.

(i) 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.

(ii) 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. Since 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.

(iii) Mechanism for secure reporting and tracing
In the Procurement system, the transactions span 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.

(iv) Lost or duplicate transactions: It is very possible for transactions to be lost in case of a disconnected network. So the transaction has to be queued in a queuing facility like the MS Message Queue to avoid loss or duplication.

4. HIGH-LEVEL MODEL OF THE PROPOSED SOLUTION

The figure below (Figure 7) shows a high-level model of the proposed transaction model.

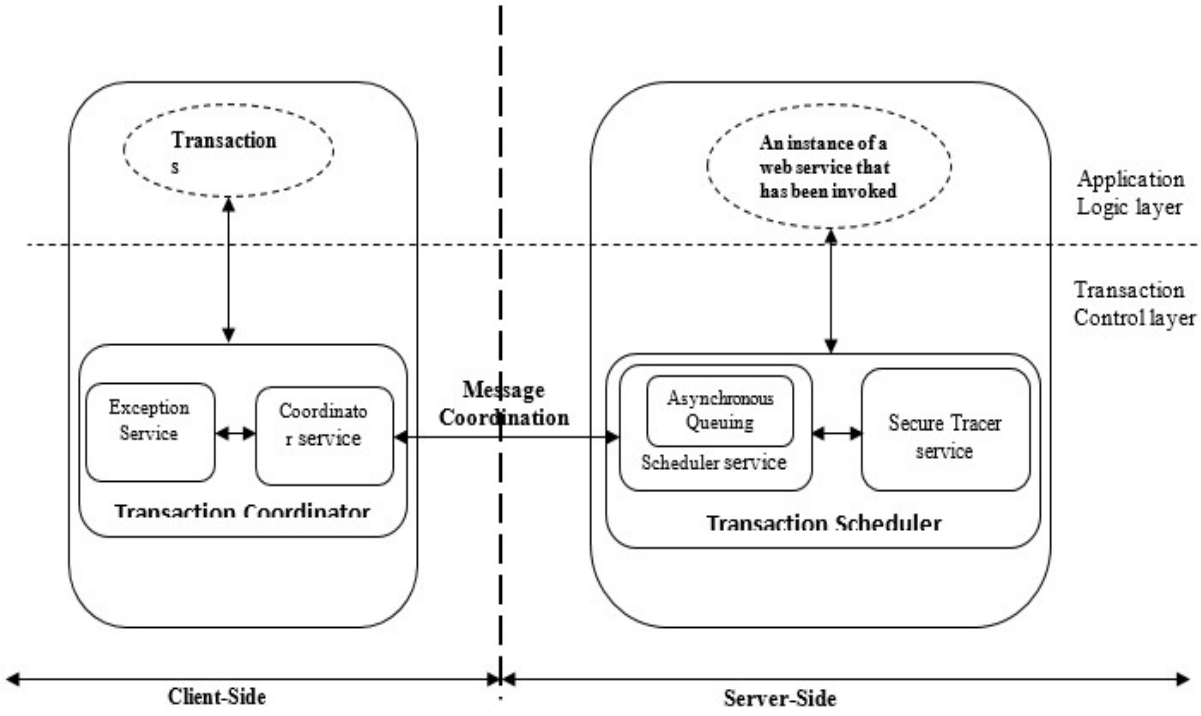

Figure 7. Generic View of Proposed Transactional and Security Model

This model separates the role of the service requester (corresponds to client-view) from the service provider (corresponding to the server-side). There are two main components of the transactional and security model, namely- transaction coordinator and transaction scheduler. The transaction coordinator contains components that can be introduced on the client's side while the transaction scheduler contains components that can be introduced on the server-side.

Before implementation, a developer has to decide whether he intends to extend the functionality of the framework on the client-side or on the server-side. From there, he can then choose the appropriate component to deploy in order to improve transactional and security support for the interoperable SOA-based system. We have introduced several components into our proposed transactional model. These components are mechanisms that will be integrated into the proposed framework to address the problems that were pointed to by our motivating scenario for this paper. These components are explained below:

(a) **The Transaction Coordinator**
The transaction coordinator is associated with the client side. It is mainly responsible for handling exceptions and coordinating the different transactions in the distributed application. It has two main components-

(i) Exception service: This service is responsible for defining exceptions and how these can be communicated to service users. For example, input messages may be incorrect and so we need exceptions that report incorrect input to the service client. The exception service is implemented on the client-side (service requester). It is recommended that all exception handling should be left to the user of the component (Sommervile, 2007). One of the most important functions of the exception service will be to take the service requesters message or object and apply the handler action, such as digital signing, encryption/decryption, or tracking the security actions for security audit or compliance reasons. In our implementation, the exception service is a process that can take the service request from the requester's object and process it. In essence, the secure object handler intercepts the service request and adds customs processing logic such as adding digital signature, encryption/decryption, or tracking the security actions for security audit or compliance reasons, to the incoming business data object.

(ii) The Coordinator service: This service is responsible for creating and coordinating the different partners according to the applied protocol.

(b) **The Transaction Scheduler**
The transaction scheduler is associated with the server-side and resides on the web service provider's side. It is mainly responsible for service-level concurrency control and security control. It has two components-

(i) Scheduler service: This service is responsible for checking if there are conflicting transactions such as transactional dependencies (i.e., concurrency conflicts) so that these can be properly handled. Several other extended frameworks have also proposed the introduction of this component to the standard web service transaction. There are several protocols and models that have been proposed for managing and scheduling concurrent web service transactions. Most of these protocols detect transactional dependencies (deadlocks) by building and maintaining dependency graphs and ensuring that it contains no cycles. For example, Alrafi(2006) proposed an extension to the standard web service transaction framework for supporting concurrency control by introducing the WS-Scheduler. The WS-Scheduler maintains a dependency graph and decides (based on the deployed concurrency control mechanism) when transactions are allowed to commit their activities and leaves the system.

Performance studies indicate that a major component of cost of running the detection algorithms is wasteful (occurs in the absence of deadlock) especially when asynchronous operations are involved (Reddy, 2003). In addressing this problem, Reddy proposes that instead of maintaining dependency graph as other protocols, we can make use of transaction-wait-for information to build a local access graph (LAGs). A LAG of T_i at site S_k contains conflicting edges of all transactions T_j such that both T_i and T_j have a conflict on some data item resident at S_k (Reddy et al, 2003). LAGs avoid significant inter-site message overhead and are convenient for deadlock detection because the possibility of deadlocks is eliminated by local computations.

In our implementation, we will adopt asynchronous queuing mechanism for the WS-Scheduler. This will enable us to eliminate the need for record locking and hence an elaborate protocol to handle concurrency control. If all requests are submitted through a single queue, the application can process its requests serially. And provided only one request is being processed at a time, the application does not need to simulate serialization by locking resources (Kaye, 2004). Local computations are eliminated which could also be a bottleneck on the overall performance of the system. (ii) Secure Tracer Service: This service allows the administrator to correlate the sender's and recipient messages in pairs and applies some security processing rules to identify any suspicious messages or trace the source details from the central logging repository. This service provides a secure tracing capability to meet audit control and compliance needs. It also provides the capability to identify suspicious messages that may have been tampered with or to scan for any problematic digital signature in the messages to allow suspicious or potential security vulnerabilities to be detected proactively.

The secure tracer is an approach recommended by Marina et al (2006), but is often ignored by most implementations. This strategy requires a central logging mechanism to pull both the sender and the recipient log messages together. Once the central logging mechanism (in our case, the shared database) is implemented, a simple message matching functionality can be built to correlate the sender and the recipient messages that refer to the same service request together.

5. EXTENDING FRAMEWORKS FOR WEB SERVICE TRANSACTION

In this section we will show how to extend framework for web service transaction frameworks. There are two frameworks that we will consider-
(i) The standard web service transactions (as described by OASIS)
(ii) The conventional enterprise transaction system built using web service

5.1 Standard Web Service Transaction Framework

Figure 8 shows how the different components of our proposed transactional model map to the standard web service transaction framework.

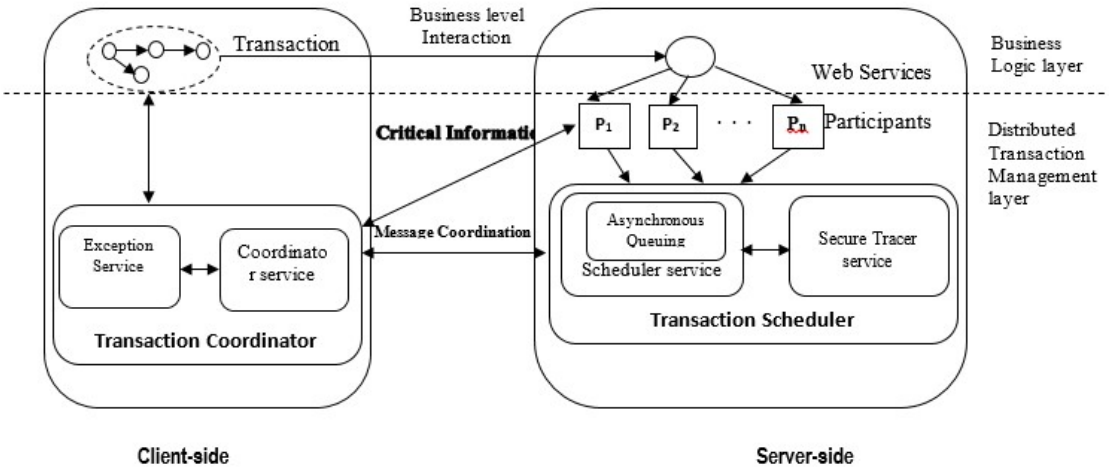

Figure 8: Extended Web Service Transaction Framework for Interoperable SOA-based Systems

5.1.1 Description of the Standard Web Services Transaction Framework

Our proposed framework resembles the extended web service transaction framework proposed by Alrifai et al (2006). Alrifai's framework was in fact an extension of the standard web service transaction protocol (as described by OASIS) for supporting currency control. However, our framework has a number of significant improvements which addresses each of the four-mechanisms proposed for incorporation into our proposed protocol. The framework has two (2) main views or sides; the transaction coordinator (which has two services- the exception service and the coordinator service) and transaction scheduler (which has two services- the scheduler service and secure tracer service). Each of these components (mechanisms) can be implemented as a service in the framework to offer transactional and security support for interoperable SOA-based systems in a distributed environment.

When a message is received from the coordinator, the exception service has to verify and validate the message. Once this is done, the message is then passed on to the coordinator service. The coordinator receives all validated messages and queues them properly using a queuing mechanism/application (such as Microsoft message queue, which ensure FIFO arrangement and also a guarantee that all messages are intact in case of disconnected mode of message communication). The coordinator passes the messages in this order to the scheduler service(within the transaction scheduler) which is responsible for checking if there are conflicting transactions such as transactional dependencies so that these can be properly handled before forwarding it possibly to the respective participants. The scheduler maintains all the possible states of the web services (these states maybe similar to the ones defined by the WS-BusinessActivity specification – completing, compensating, aborted, committed etc.). The secure tracer service records every transaction sent to the participants (service providers). This information can be used to trace and report the transaction for suspicious activities.

5.2 Conventional Enterprise Transaction System Built Using Web Services

This section shows how the proposed transaction maps to the conventional enterprise transaction system (see Figure 9). We assume that the sample transaction- that is, the procurement system is designed in line with the architecture of a conventional enterprise transaction system.

5.2.1 Description of the Enterprise Transaction System Built using Web Services

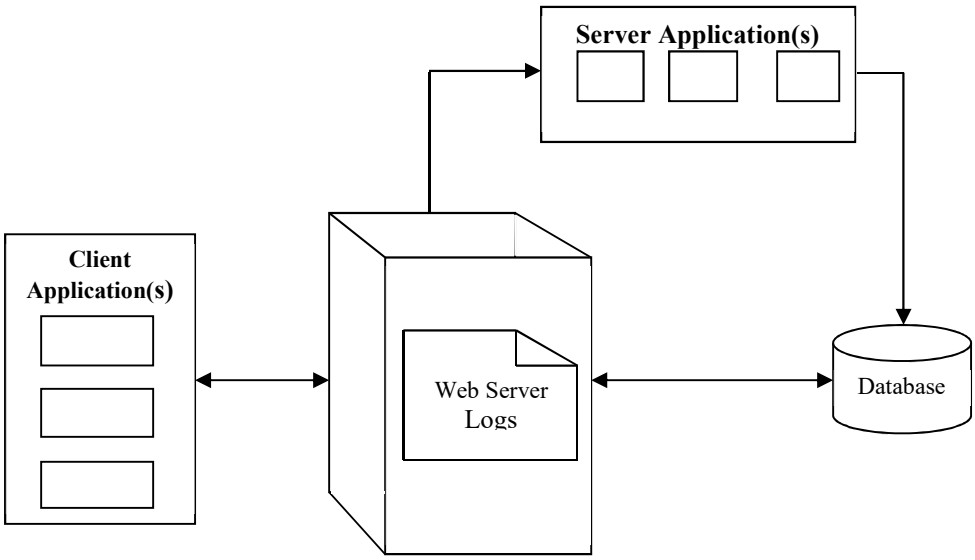

Figure 9. Conventional Web Services Transaction Framework (Source: Connolly, 2004)

Assuming that the client invokes a web service synchronously, an immediate response to each request is required, which implies forcing a two-way data exchange for every service interaction. When services need to carry out synchronous communication, both service and service requester must be available and ready to complete the data exchange. This can introduce reliability issues when either the service cannot guarantee its availability to receive the request message, or the service requester cannot guarantee its availability to receive the response to its request. Synchronous message exchanges can also impose processing overhead as the service requester needs to wait until it receives a response from its original request before proceeding to its next action. Prolonged responses can introduce latency by temporarily locking the consumer and service. Apart from this, synchronous communication can cause overload of services required to facilitate a great deal of concurrent access. This is because services are expected to process request as soon as they are received, and so usage thresholds can be more easily reached, thereby exposing the service to multi-consumer latency or overall failure (Erl, 2009).

6. EVALUATION

This section presents how our novel architecture was evaluated. This has been achieved by carrying out an architecture analysis and demonstrating how our model maps to the procurement system.

6.1 Architecture Analysis

This section presents an analysis of the architectural framework. In software architectural analysis, the software architecture is evaluated against some criteria." (Kazman, 2014). The three main drivers of this architectural analysis are quality control, risk reduction and cost/benefit. The aim of architecture evaluation is "to analyze the software architecture to identify potential risks and verify that the quality requirements have been addressed in the design. This means that architectural evaluation is an architectural analysis with a specific analysis goal, which is to determine whether the architecture design satisfies the requirements that are defined as architecturally significant requirements.

The goal of the architecture evaluation is stated as follows:
Provide an enhanced architectural framework that will significantly improve transactional and security support for interoperable SOA-based system by incorporating the following mechanisms:
(1) error-handling
(2) concurrency control
(3) consolidating multiple transactions call
(4) secure reporting and tracing for suspicious transactions

Architecture analysis can be performed manually or automatically. Manual analysis means that shareholders perform architecture analysis without the use of dedicated tools. The definition of one or more analysis goals is the starting point of each analysis. This paper adopted the four categories of analysis goals proposed by Taylor et al (2014) to analyze our proposed architecture. The four categories are: completeness, consistency, compatible and correctness.

Completeness
This is made up of external completeness (i.e., whether all system requirements have been addressed in the architectural design) and internal completeness (whether all important architectural elements have been defined and whether all design decisions have been made). For this architecture we can confirm that all the requirements specified in the goal have been addressed in the architecture. For example, we have included the Secure Tracer Service to allow the administrator to correlate the sender's and recipient messages in pairs and applies some security processing rules to identify any suspicious messages or trace the source details from the central logging repository.

Consistency
This entails checking whether the defined architecture contains contradicting information (e.g., inconsistent names, interfaces) or not. For this architecture, we have ensured that there are no redundant message and the flow of interactions between the different components are consistent throughout the system.

Compatibility
This entails checking whether an architecture adheres to design guidelines and constraints defined by architectural styles, and standards. For this architecture, we have checked the arrows uses to show exchange of messages are consistent with whether the messages are asynchronously or synchronously.

Correctness
This entails performing the analysis with respect to some artefact of reference, for example, the specific system requirements and implementation standards. For this architecture we checked that the architecture conforms to the standard web service transaction framework. This was achieved by mapping the components of the proposed architecture to the components on the standard framework after the traditional web service transaction framework was extended and deploying the architecture with the procurement system.

For example, the three independent processes process1, process 2 and process 3 sends request/transactions to the coordinator, which may reside on different machines and may also be implemented on different platforms. The coordinator is responsible for creating the context and coordinating the different partners according to the applied protocol. The coordinator role can be taken by the initiator of a distributed application or by a (trusted) third party.

6.2 Deployment of the Proposed Model with the Procurement System

In this section, we present a practical application of the proposed architecture. This is achieved by showing how the deployment of the proposed model (which is an extension to the conventional web service transaction architecture) will help to enhance transactional and security support for the procurement system (which represents an interoperable SOA-based system).

6.2.1 Deploying the Proposed Model with the Procurement System

This section shows how the proposed model can be deployed with the procurement based on the standard web service transaction (see Figure 10). We also describe specifically how each of the four mechanisms that we have introduced into the web service transaction framework can be properly utilized.

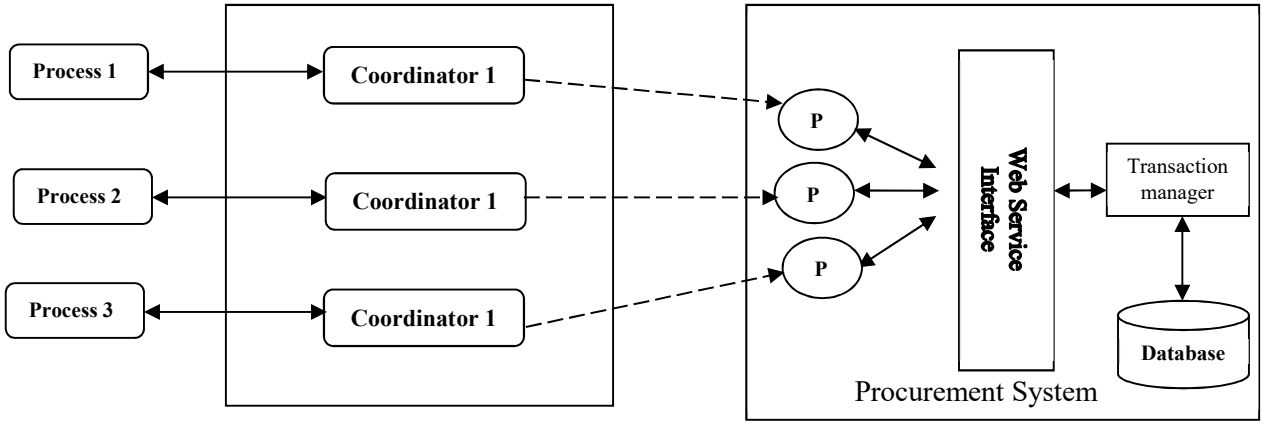

Figure 10. The Procurement system uses the Web Service Transaction Framework. The processes are concurrently and asynchronously involved in three different independent transactional contexts.

(i) Transaction Coordinator

The figure below (Figure 11) shows how three independent processes process1, process2 and process3 are trying to send request/transactions to the coordinator. The coordinators (processes) may reside on different machines and may also be implemented on different platforms. These processes are concurrent, distributed and asynchronous in nature.

Within the WS- transaction framework, there is no guarantee that all transactions passed from the coordinator to the participant are in a secure and reliable (failsafe) manner. This is because most design and implementations of distributed system assume that direct connections always exist. In reality, this is far from being true especially for service-oriented systems where loosely coupled systems take part in transactions that last for a long time. In a service-oriented system, processes communicate across heterogeneous networks and between computers and platform which may not always be connected. Apart from this, transactions are not communicated in a unified programming model compatible with other communications standards.

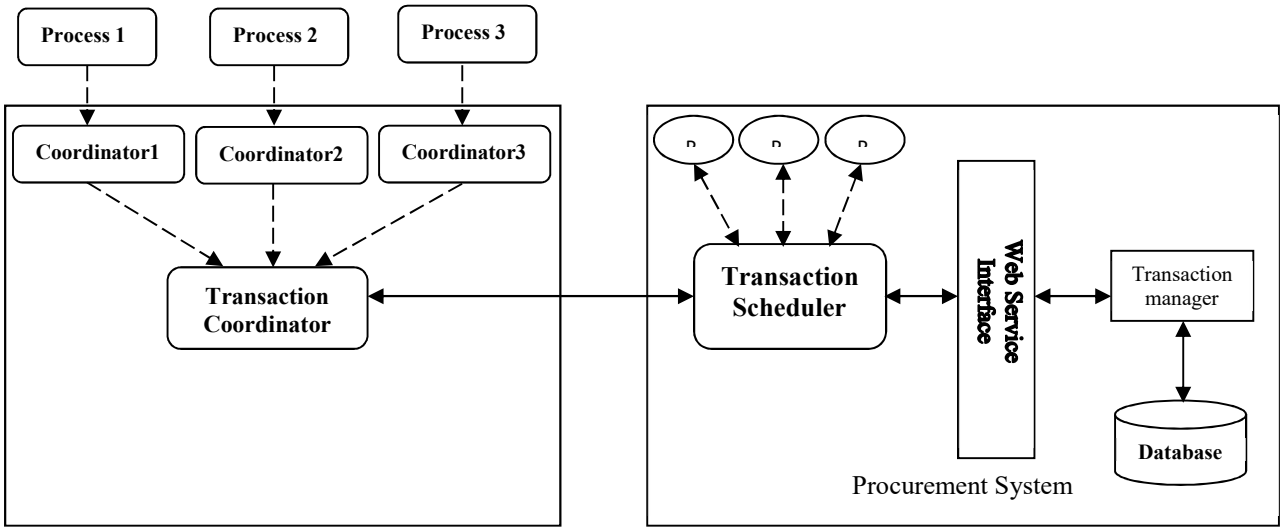

Figure 11. The extended web service transaction framework manages three contexts. Transaction Coordinator is deployed on the client's side (the initiator of the distributed transactions), while the transaction scheduler is deployed on the server-side (the service provider).

With the deployment of a WS-Transaction Coordinator as shown in the figure below, the problem of reliable delivery can be addressed properly so that messages can be delivered reliably between applications inside and outside the enterprise. The exception service will check and validate that all transactions carries with it all the necessary parameters according to the agreed protocol of the initiator of the distributed transaction. Coordinator service places the transactions that fail to reach their intended destination in a queue facility and then resends them once the destination is reachable. It also supports security and priority based messaging. Later, dead letter queues may be created for looking at transactions that timed out or failed for other reasons. The issue about transactions is the nature and contents of the messages sent. The content of the transactions can be simple messages or objects. These objects can be serialized and sent to the queue and then can be de-serialized and received by another application. Messages sent from coordinator to participant (client to server) must be secure. A secure data transport mechanism ensures confidentiality and reduce the risk of principal security spoofing.

(ii) Transaction Scheduler

Again, three independent processes process1, process 2 and process 3 access the purchase order subsystem of the procurement system that was described in chapter one (motivating scenario). The three processes are coordinated by three autonomous coordinators Coordinator 1, Coordinator 2 and Coordinator 3. These coordinators are represented locally (within the procurement system) by three participants P1, P2 and P3. Potential transactional dependencies are not visible to the three processes and therefore cannot be detected and handled.

Figure 11 shows what happens when the transaction scheduler is deployed, transactional dependencies can be detected and handled properly. Apart from this, the transaction scheduler will also allow for secure tracking and reporting of transactions for suspicious activities. This will be accomplished by the scheduler service and the secure tracker service.

In the extended framework, processes can no longer contact the web scheduler directly. They are queued in the transaction coordinator before passing it to the web transaction scheduler)
At this point, we assume that there are no transactional dependencies between concurrent transactions and also there is no global waiting cycles.

6.2.2 Deploying the Proposed Model with the Procurement System

In this case, we assume that the procurement system is a conventional enterprise transaction system built using several web services components.

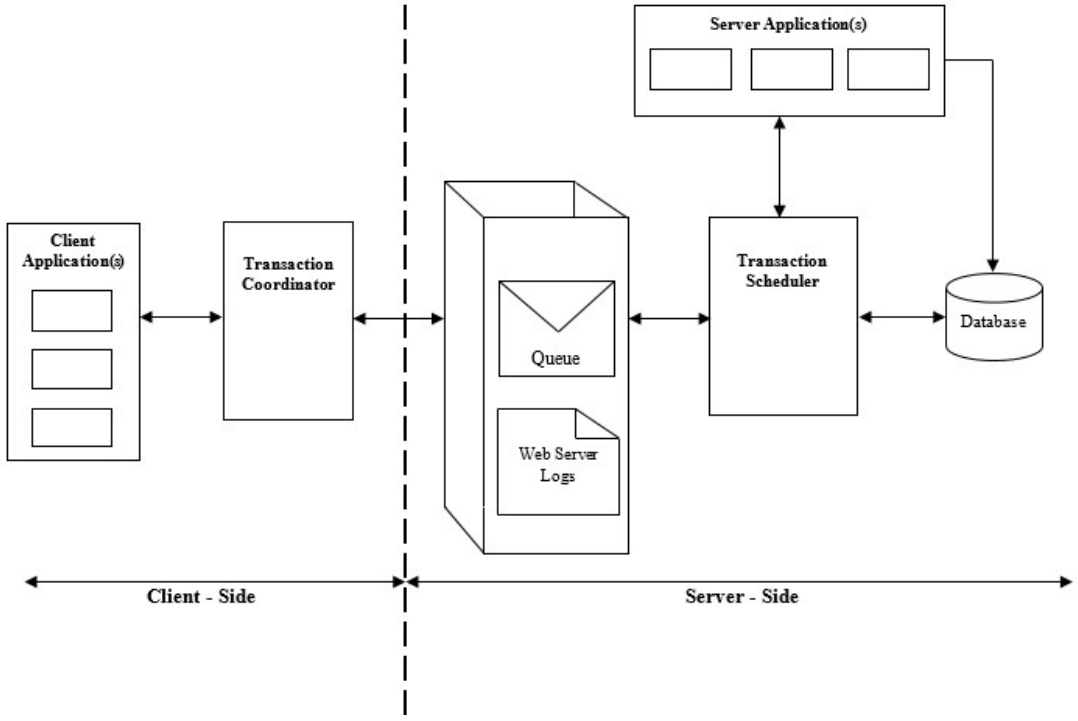

Figure 12. Deploying the proposed framework with convention enterprise transaction system built with Web services

With the deployment of the transaction coordinator after the client application, a service can exchange messages with its consumers via an intermediary buffer, allowing service and requester to process messages independently by remaining temporarily decoupled. An asynchronous queuing technology is introduced into the framework (architecture) as shown in Figure 12. The client application sends a message which is intercepted by the transaction coordinator and stored in an intermediary queue. The Transaction scheduler pulls the message from the queue and sends it to the application servers, and at the same time issues a log message to the database. The application servers then process the message and then issue a log message to the database (or some other central logging system). The central logging system in collaboration with a component in the transaction scheduler (secure tracer) can apply some security processing logic to ensure that the suspicious messages are flagged for administrator's attention. An option alert queue can be created for storing these alert messages.

Asynchronous queuing technology can be completely transparent, meaning that neither the requester nor the service may know that a queue was involved in a data exchange. This technology also ensures that Messages in transit are not lost should a system failure occur.

6. CONCLUSION

This paper has presented an enhanced architectural framework for improving transactional and security support for interoperable SOA-based systems, and to also improve the quality of service of the system. We presented the traditional web service transactions framework and thereafter showed how to extend the traditional web service transaction models to incorporate interoperable SOA-based systems using the procurement systems as a case study. Firstly, we presented two traditional web service transactions framework. Secondly, we presented our proposed transaction model and show how it maps to the traditional frameworks. Thirdly, we showed how to extend the traditional web service transaction models to incorporate interoperable SOA-based systems using the procurement system as a case study. Extensive evaluation of the architecture using architectural analysis shows that the architecture satisfies the analysis goals, and system requirements, and is thus recommended for system designers and developers to enhance transactional support for interoperable service-oriented systems. In future, we plan to investigate how an interoperable SOA-based system can be deployed on the cloud to enhance distributed transactions, especially in a multi-cloud environment.

REFERENCES

1. Arlitt, M and Jin, T.(1999). Workload Characterization of the 1998 World Cup Site. Technical Report HPL-1999-359R.10, HP Laboratories Palo Alto
2. 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.
3. Little, M (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...
4. Malrait, L, Marchnad, N., Bouchenak, S.(2009). Average Delay Guarantee in Server Systems Using Admission Control.
5. Menasce, D., Almeida V., Dowdy, L. (2004). Performance By Design: Computer Capacity Planning by Example. Pearson Education, Inc. New Jersey, USA.
6. Fisher, M., Sharma, S., Lai, R., & Moroney, L. (2006). *Java EE and. Net interoperability: integration strategies, patterns, and best practices*. Prentice Hall Professional.
7. Computer Security Institute (2005). Cited in Fisher, M., Sharma, S., Lai, R., & Moroney, L. (2006). *Java EE and. Net interoperability: integration strategies, patterns, and best practices*. Prentice Hall Professional. Page 403
8. Ochei, L. C (2012). Towards an Enhanced Protocol for Improving Transactional Support in Interoperable Service Oriented Application-Based (SOA-Based) Systems.
9. 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
10. 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.
11. 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.
12. Sommerville, I. (2007). Software Engineering (Eight Edition). Pearson Education Limited, Harlow, England.
13. Kaye, D.(2004). Web-Services Transactions: From Loosely Coupled - The Missing Pieces of Web Services. XML: Article. SYS-CON Media. Retrieved on August 28, 2012 from http://xml.sys-con.com/node/43755
14. Weikum, G.(1991). *Principles and realization strategies of multilevel transaction management*. ACM Transactions on Database Systems, vol. 16, no. 1, 1991, pp. 132-180.
15. Erl, Thomas (2009). SOA Design Patterns. Pearson Education, Inc. New Jersey, USA.

16. Buchgeher, G., & Weinreich, R. (2014). Continuous software architecture analysis. In *Agile Software Architecture* (pp. 161-188). Morgan Kaufmann.
17. Taylor RN, Medvidovic N, Dashofy EM. Software architecture: foundations, theory, and practice. New Jersey: Wiley; 2009.
18. Kazman R, Bass L, KleinM, Lattanze T, Northrop L. A basis for analyzing software architecture analysis methods. Software Qual J 2005;13:329–55, Kluwer Academic Publishers.