Journal of Behavioral Informatics
Digital Humanities & Development Research
www.behaviouralinformaticsjournal.info

Vol 6, No 1, March, 2020

Design Model for an Interoperable SOA-based System: A Case Study of Procurement System

[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 a design model for an interoperable SOA-based system based on a case study of a procurement system. We first describe a procurement system together with an architecture for the system. Thereafter, we show how the Transaction Coordinator component of our proposed architectural framework (that is, an extension of a standard web service transaction framework) maps to the architecture of the procurement systems to provide transactional and security support for the procurement system. This paper also presents a novel Distributed Transactional and Security Support (TSS) Algorithm for enhancing transactional support for an interoperable SOA-based system. The TSS algorithm is an integration of three supporting algorithms, namely - the admission control algorithm based on response time target, secure-tracer algorithm, concurrent transactions scheduler (with asynchronous queuing) algorithm.

Keywords: Service-oriented systems, Interoperability, Measurement, Performance, QoS, scalable

Journal Reference Format:
Ochei, L.C., Ogunsakin, R., Wobidi, Echebiri (2020): Design Model for an Interoperable SOA-based: A Case Study of Procurement System Behavioural Informatics, Digital Humanities & Development Journal Vol 6. No. 1. Pp 137-156. SMART Research Group, International Centre for IT & Development (ICITD), Southern University Baton Rouge, LA, USA.

1. INTRODUCTION

An interoperable SOA-based system (or service-oriented system) is composed of more web services that are implemented in different platforms. These systems abound everywhere and are becoming very critical in business operations. A typical example of an interoperable system is the purchase order system (traditional e-commerce application), where for instance, a purchase order has to be submitted across multiple systems. Interoperable SOA-based systems have several challenges including performance, availability and security. For example, it is often difficult to connect concurrent business processes running on disparate platforms into a single transaction, coupled with the problem of running concurrent transactions with interleaving operations spanning across different applications and resources.

As a result of the relevance of interoperable systems and its challenges in ecommerce and other distributed services, there is a need to formulate a motivating problem, together with a design model and supporting algorithms that capture these challenges and design model to help system designers and developers, and architects build interoperable systems to improve transactional and security support.

The main contributions of the paper are:

i. Design model for interoperable SOA-based systems based on a case study of a procurement system.
ii. Description of a procurement system that requires transactions and security support, and then integrates a proposed architecture into the procurement System.
iii. Novel distributed transactional and security support (DTSS) Algorithm for enhancing transactional support for an interoperable SOA-based system.

The rest of the paper is organised as follows: Section 2 discusses the designing of the procurement system. Section 3 describes the development of the procurement system including the design objectives, application structure, description of the interoperable SOA-based system program specification, and system

2. DESIGNING THE PROCUREMENT SYSTEM

The Procurement System represents an interoperable SOA-based system and will also integrate the proposed model and algorithm. One of key reasons for designing the procurement system is to help us generate sample data to use for simulation. 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.

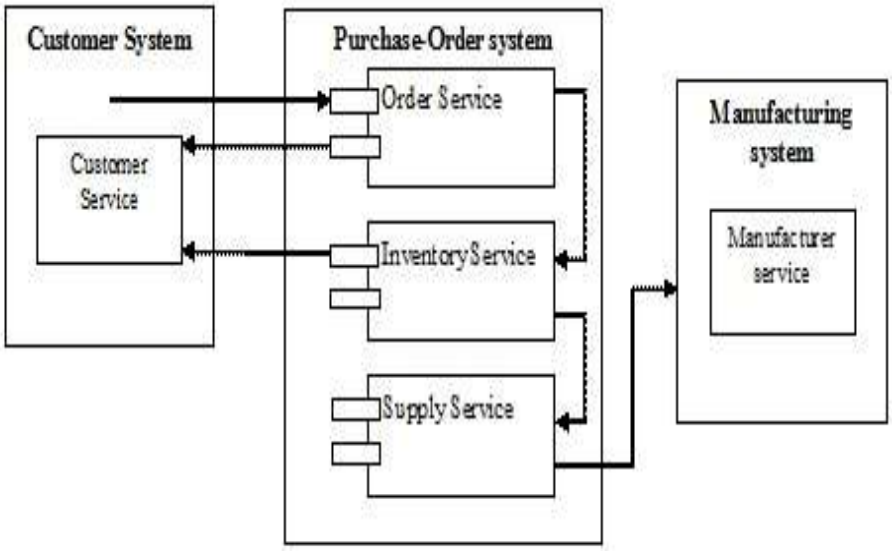

Figure 1: Detailed deployment Diagram for the Procurement System

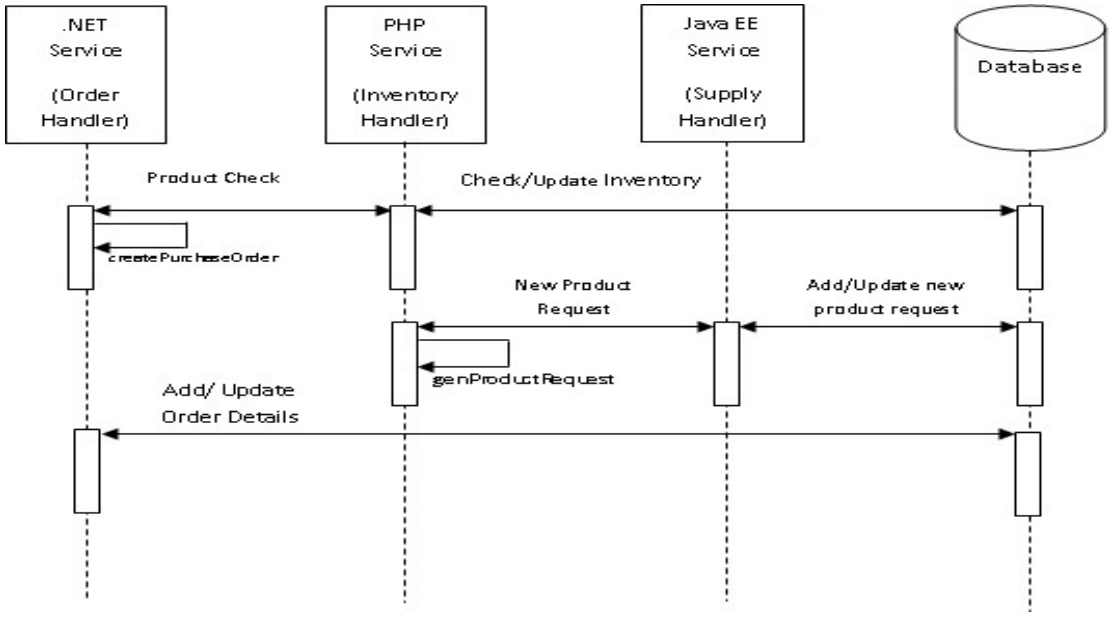

Figure 2: An expanded view of the Purchase-Order subsystem (sequence diagram for message flow between the three service handlers).

2.1 Features of the Procurement System

The distributed transaction scenario (or use case) which is adapted from Ochei et al (2021) is 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 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. This scenario 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 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.
vi. 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 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.
vii. There are databases that are shared by two or more components within the system.

2.2 Description of the Purchase Order Subsystem of the Procurement System

This study is proposing a protocol that will significantly improve transactional and security support for asynchronous applications in a distributed environment. This protocol (algorithm) will be integrated into a procurement system, which represents an interoperable SOA-based application, where a specific subsystem, the Purchase-Order system operates within its own organization. This system interacts with two other sub-systems, the Customer subsystem, and the Manufacturer subsystem. The details and the composition of these subsystems are being handled by the respective organizations, which may of course also be implemented using different platforms and resources.

The purchase order sub-system is composed of web services deployed in a distributed environment. In a nutshell, the procurement system is composed of three main web services - the order service, the inventory service and the supply service. The order service is implemented in .NET, the inventory service is implemented in PHP, and the supply service is implemented in Java EE. The implementations are not a demonstration of how to write a perfect web service application in Java EE, .NET or PHP platform. It is also not how you would write a secure application to handle a procurement system, but it provides a sample program which integrates our protocol. The model transaction involved in the purchase order sub-system of the procurement system is simple: Client application invokes a Web service to submit orders for processing. Thereafter the server application invokes a chain of web services to process the order. The metric application reports the client metrics used in the transaction.

3. DEVELOPING THE PROCUREMENT SYSTEM

The phases involved in the development of the procurement system are as follows:
(a) Development of dependable reusable components (or services). The services could be developed in different platforms (e.g., .NET, Java EE, and PHP).

This phase involves -

i. Identifying candidate services for implementation
ii. Defining the service and interface
iii. Implementing, testing and deploying the service.

(b) Development of the procurement application from these reusable components (or services).

3.1 Design Objective for the Procurement System

The main objective of the procurement system is to develop a sample program that is based on the motivating scenario in a way that supports the collection of collect metrics for performance analysis and evaluation. The data collected can then be used to develop a model to enhance transactional and security support for an interoperable SOA--based system. Some of the objectives of the design of the procurement system are:

i. Reproduce the motivating scenario so we can use simulation to study and obtain more information about the system.
ii. Design a system to use the client-oriented recording approach to collect client metrics for performance evaluation without degrading the performance of the system.
iii. The procurement follows the architecture of an enterprise transaction system.
iv. The procurement system is an interoperable SOA-based system.
v. It is also a distributed system that handles distributed transactions, and which is deployed in a distributed environment.
vi. In addition to reproducing the transaction model described in the motivating scenario, the design supports the simulation of concurrently running web service transactions where concurrency violations can be triggered. That is, the design will allow for concurrency violations to occur, and thereafter we integrate the proposed protocol to detect and handle the concurrency violations.

3.2 Application Structure of the Procurement System

The development of any application system begins by first sketching a rough diagram of your design based on the basic pages/modules/components that need to be created. The diagram shown below illustrates the flow of data and information across the procurement system.

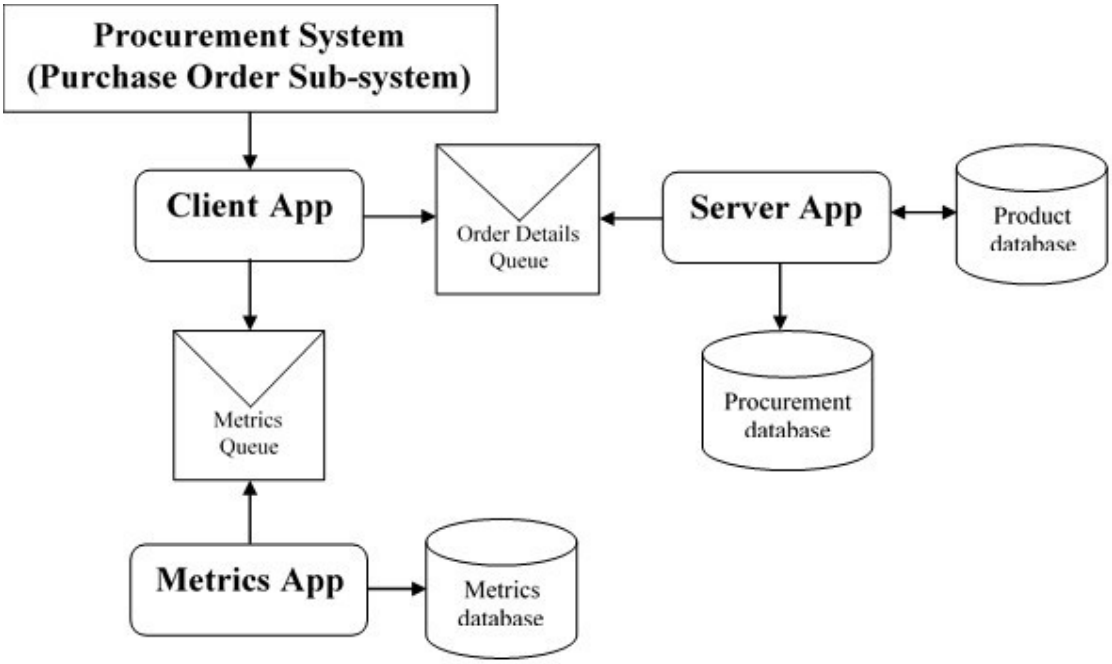

Figure 3: The Structure of the Purchase order subsystem

The above architecture maps well into the conventional three-tier client/server architecture. It is a distributed systems architecture, where the information processing is distributed over several computers rather than confined to a single machine.

The flow of data and information across the system is summarized below:

1. The client Application (represented by ClientApp.aspx) invokes a web service to submit an order for processing. In this implementation, the SOAP request apart from carrying the order details (in the SOAP body) also carries client metrics data in optional headers in the same SOAP messages that are used to complete the user operation.
2. The web service that processes the SOAP request places the order details in the order details queue.
3. The web service that processes the SOAP request strips off any metrics headers and place them on a queue.
4. The server application retrieves messages from the order details queue and invokes a chain of three web services to process the orders that were placed. Three web services are invoked- **Orderservice**, **InventryService** and **SupplyService**. In the next section, we will describe the basic web service methods.
5. The Metrics application de-queues client metrics data from the metrics queue and places them into the metrics database designed to allow general-purpose transaction analysis and reporting.

3.3 Description of the Interoperable SOA-based Application

There are three sub-application that work together to accomplish create orders, process order and gather metrics for analysis. In the section that follows we describe the transaction logic of each of these applications:

3.3.1 Description of the Applications

(a) Client Application

The sequence of steps used in performing the functions in the client application is outlined below:

1. Start
2. Create two queues if it does not already exist. They are – orderdetails, metricsdetails.
3. Declare simulation parameter such as scale, number of transactions, number of client etc.
4. Generate client thread
5. For each client thread generated
    - 5.1 Generate orderNO
    - 5.2 For each OrderNo generated
        - 5.2.1 Declare and set client parameters
        - 5.2.2 Make asynchronous calls to Order Web Service
        - 5.2.3 Get the result (transactionID) of the web service call
        - 5.2.4 Declare and set more client parameters
        - 5.2.5 Place client parameters in soap metrics queue
6. Stop

The flow of information between the client application and its components is shown in figure 4.

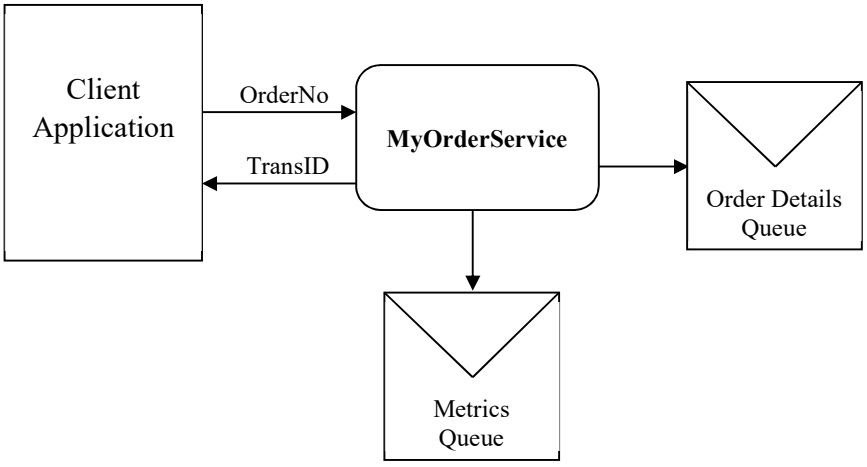

Figure 4: Flow of information between the client application and its components

(b) Server Application

The steps used in performing the functions in the server application are outlined below:

1. Open the message queue
2. Retrieves message from queue and extracts orderID
3. Invokes the OrderService asynchronously and passes the orderID as a parameter to it.
4. Once processing is completed, the OrderService web service method returns the Purchase number (PO) to it.

The flow of information between the server application and its components is shown in figure 4.

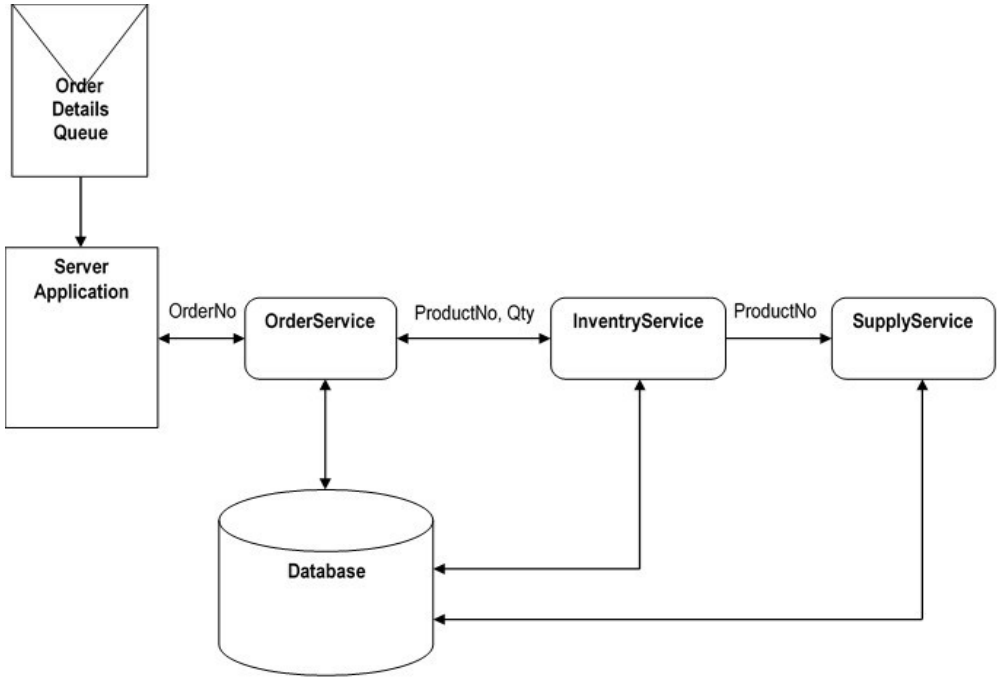

Figure 5: Flow of information between the server application and its components

(c) Metrics Application

The steps used in performing the functions in the metrics application are outlined below:

1. Open the metric database
2. Retrieve the metrics details from the metricsDetails queue
3. Insert the details into metricsTransaction table in the metrics database.
4. Report the number of messages retrieved.

The flow of information between the metrics application and its components is shown in figure 6.

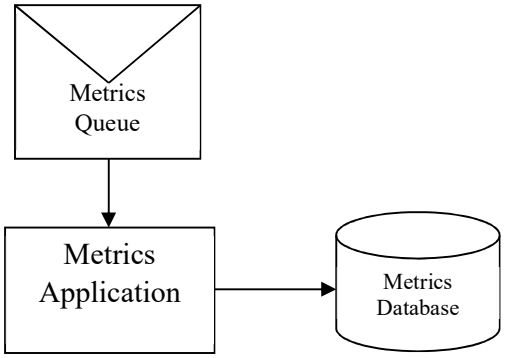

Figure 6: Flow of information between the Metrics application and its components

3.3.2: Description of Transaction Logic for Web Service Methods

There are several web services methods that would be created for consumption by the procurement system. The transaction logic for each of these web service methods are described below:

(a) OrderService web service method:

1. Open the procurement database
2. Select all records in the Orderdetailssetings table whose OrderID matches the one passed to it from the OrderService from ASPX page that represents the server application.
3. For each order details record, capture the product number (ProductNo) and the quantity demanded (QtyDemanded) by the customer.
4. Invoke the InventryService web service method pass the two parameters – productNo and QtyDemanded.
5. InventryService will return a Boolean value that indicates whether or not the quantity available for that ProductNo less than that QtyDemanded(that is, QtyAvailable-QtyDemanded >0)
6. if the Boolean value returned is True, then it generates a Purchase order number (PurchaseNo).
7. The results (PurchaseNo) are saved in the POResult table of the Procurement database.
8. Open the OrderTransaction table in the procurement database and update the serverEndTime to the current time and also the Transactionstatus to "Finished"
9. Returns a Boolean value to the web service method that invoked the inventryservice web service method.

(b) InventryService Web Service Method

1. Open the product table
2. Select the record that matches the ProductNo passed it.
3. Capture the current quantity available for that product
4. Compute the current quantity available
5. CurrentQty = QtyAvailable-QtyDemanded
6. If QtyAvailable-QtyDemanded < 0, then invoke the SupplyService, passing to it the ProductNo.
7. The InventryService returns a Boolean value to the OrderService indicating whether or not the product is available.

(c) SupplyService Web Service Method

1. Open product table
2. Generate a random number that represents a new product request.
3. Compute the quantity to reorder as difference between target level and current quantity available (ReorderQty = TargetLevel –QtyAvailable).
4. Update product quantity (in the product table) with this value(i.e., ReorderQty)
5. Insert the RequestNo in the ProductRequest table in the product database.
6. No value is returned to the calling web service method.

(d) MyOrders Web Service Method

1. Set and initialize variables
2. Save transaction details into orderTransaction table(in procurement database)
3. Generate a random number for the number of items to order for.
4. For each No. of items to order
    a. Generate the ProductNo
    b. Generate the Quantity of the product
5. Send order details to queue
6. Save order details to the database
7. Return transactionID back to the client application

3.4. Program and Module Specification

Modules are implemented in this system as application logic in the web pages of the web application. A module may contain several web pages each of which is used to perform a specific function. For example, the *ClientApplication* page contains the application logic to perform the function of simulating a population of clients submitting orders in which the OrderNO are generated randomly at a fixed interval.

Three main modules will be implemented in this system. These modules include:

(i) Client module – this module will be responsible for creating multiple threads to simulate many SOAP clients submitting order requests (generated randomly) for processing. This module may have other sub-modules that may be for example responsible for collecting client metrics and logging them into the metric database. This module is implemented using the Client application.

(ii) Server module – this module is response for obtaining order details from the Microsoft Message Queue (MSMQ), that is placeorderdetails queue, and logging them to the procurement database. This module is implemented using the Server application. This server application can also be created as a console application. The server application displays the result of the processing such as PO, transaction ID, Client ID etc.

(iii) Metric Module – this module is responsible for retrieving client metrics from one or more databases (data sources) and putting them in central repository and in the format that will allow for analysis and evaluation of the metrics. At the end, it displays summary of the client metrics data for each transaction. This module is implemented using the Metrics application. This client application, sever application and metric application can be created as a console application in any platform such as .NET platform.

3.5 System Specifications

This section describes the various specifications of the system such as database, input, output, and interface as well as program module specification.

3.5.1 Input Design

Input design addresses the design of the interface which would be used to collect information from the user for input into the computer. For example client application is built as a console application, and so data is collected directly by entering input parameters on the input screen using the keyboard. Given below is a sample interface for capturing simulation parameters.

Simulation Parameters

Enter Size of maximum OrderNO: **[cursor]**
Enter Number of Orders: **[cursor]**
Enter Milliseconds between Clients submissions: **[cursor]**
Enter Number of Clients: **[cursor]**
Enter Seconds between Client starts: **[cursor]**

Press enter key to start simulation:

Figure 7. Input interface for client application

3.5.2 Output Design

Outputs present information to system users. Outputs are the most visible component of a working information system and are the basis for the user's and management's final assessment of the system's value. Each of the three major applications simply displays on the output screen the transaction information that is related to the processing that takes place in the application. The output screen can be formatted in such a way that the information is displayed on a separate row (i.e., line by line). The Client Application Report could look like this:

Client Application Progress Report: Client 1
TransactionID: 0d196626-00c8-4fd3-965c-78b6ec0c29fb
ClientID: ff5bf6d7-d67f-42f6-b1bc-340fd57aca32
Client Start time: 8/8/2012 12:13:11 AM
Elapsed Time: 3153
Server Start time: 8/8/2012 12:13:15 AM
Transaction Status: **In progress**

Figure 8. Sample Client Application Report

The Server Application will display every order details it receives from the order details queue plus the transaction IDs for every transaction. The Server Application Report could look like this:

Server Application Report: Client 1, Transaction 1
ClientID: ff5bf6d7-d67f-42f6-b1bc-340fd57aca32
TransactionID: 0d196626-00c8-4fd3-965c-78b6ec0c29fb
OrderID: 880763
ProductNo: 5
Quantuty: 19
OrderDate: 8/8/2012 12:13:15 AM
Order Status: progress
Result: **True**

Figure 9. Sample Server Application Report

The Metrics Application will display every metrics report it receives after it logs it in the Metrics database. It shows the transaction and client IDs, the transaction start time, its elapsed time in milliseconds, and any other attribute related to the client transaction. Take note that the transaction status for metrics report is "completed" whereas for client report it is "in progress". A sample metrics report follows:

Metrics Application Report: Client 1
TransactionID: 0d196626-00c8-4fd3-965c-78b6ec0c29fb
ClientID: ff5bf6d7-d67f-42f6-b1bc-340fd57aca32
Client Start time: 8/8/2012 12:13:11 AM
Elapsed Time: 3153
Server Start time: 8/8/2012 12:13:15 AM
Transaction Status: **Completed**

Figure 10. Sample Metrics Report

3.5.3 Database Design

For this system, a relational database such as Microsoft SQL Server 2008, MySQL and Postgres is required to create and manage the data. Each database will consist of several tables that will directly relate to a specific process in that database. The databases and tables required for the systems are summarized below:
(i) Metric database contains the following tables - StartTransaction, EndTransaction, TransactionAttributes
(ii) Product database contains the following tables - Product table and ProductReqeust
(iii) Procurement database contains the following tables - OrderTransaction, OrderDetailsSetting
The table specifications can be seen in Appendix A

3.5.4 Server Work Queues

Servers usually contain a collection of resources that can process customer request. A viable way of distributing work to among these resources is to use queues. This study recommends using a Message Queue such as Microsoft Message Queue (MSMQ), a queuing resource manager available in most Windows operating systems. Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues.

Two message queues can be created programmatically for the system –
(i) placeorders to hold details of the order placed by customers
(ii) metricsdetails to hold the metrics details of the transaction

4. INTEGRATING THE PROPOSED MODEL INTO THE PROCUREMENT SYSTEM

From an implementation standpoint, the figure below (Figure 11) shows how our model is integrated into a web service distributed transaction scenario to improve transactional and security support for the system. In our case, this web service distributed transaction scenario simply represents the server application (that is, the one on the server-side) that processes the order request.

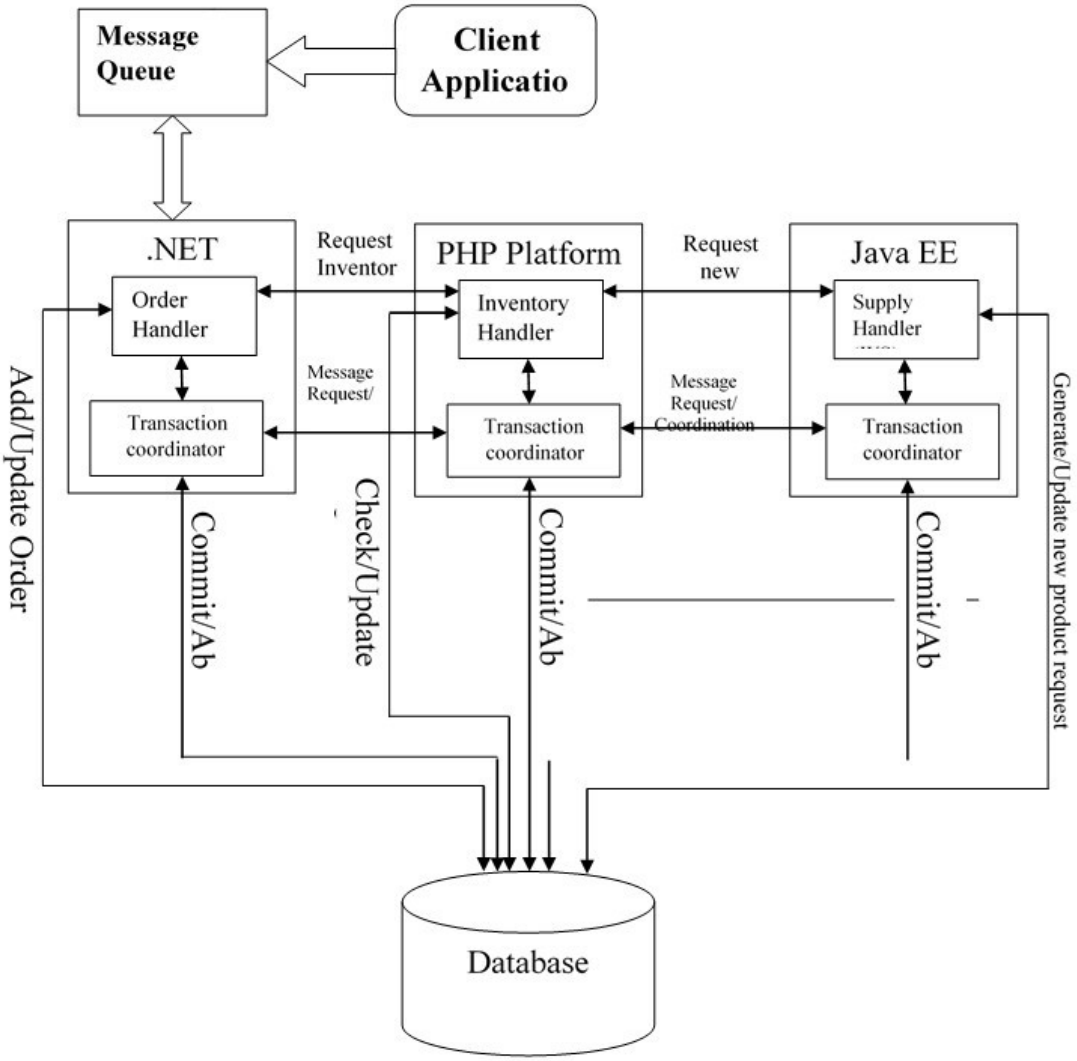

Figure 11. Deploying Transaction Coordinator component of our proposed framework on the server-side

The scenario illustrated above assumes that the messages are sent from a queue structure (such as message queue-MSMQ). Messages are sent to the queue from the client application coordinator which we have extended to allow for **asynchronous queuing model**. In this scenario, the Order handler web services calls the Inventory service and initiates a transaction. We assume that all participants of the transaction register with the distributed transaction coordinator (a component that implements our proposed model). The transaction coordinator coordinates all other transaction coordinators across the different platforms. The coordinator is a web service that could run either inside the server that runs the different handlers (for example, .NET, Java EE or PHP) or as a separate Web service on the network.

5. IMPROVING THE QUALITY OF SERVICE (QOS) FOR INTEROPERABLE SOA-BASES SYSTEMS

In the previous section, a high-level model of the simulated system is presented (Figure 12). In this section, we construct a model (performance model) to enhance transactional support for the system through improved quality of service. This focuses on response time and throughput, two important parameters that affect the quality of service. One of the ways to guarantee good quality of service (QoS) is to control and limit the number of requests handles concurrently by the system (Manasce, 2004). This is called admission control. The system load is directly related to the number of concurrently executing request in the system. That is, overload occurs when the system does not have enough resources (or capability) to handle the large magnitude of the concurrently incoming request. If a proper admission control policy is implemented, the number of request within the system is limited so that response time does not exceed a certain threshold. However, this is realized at the expense of rejecting requests. This means that we have to design a model in such a way that while accepted request experiences an acceptable level of service, the rejected ones do not have to suffer very large delays to be re-admitted. This also implies that we have to provide transactional and security support to the rejected requests.

One of the mechanisms that limit the number of request in a system is the limits on the multiprogramming level (MPL). Limiting the total number of request allowed in a system limits the amount of time spent by request at various queues and therefore guarantees that the response time will not grow unbounded. Our approach in this research is to first compute and set a limit, W, on the number of transactions allowed into the system. An arriving transaction that finds W transactions in the system is blocked.

Two options are considered to handle the blocked request:
(i) Delaying subsequent transactions from arriving at the system
(ii) Placing the request in a temporary queue waiting to re-enter the system

This approach is consistent with other research (Schroeder et al, 2006). The issue of how to adjust the MPL to achieve good system performance has been studied in the context of transactional workloads, both via extensive experimentation and queuing theoretic analysis. Schroeder et al (2006) reported that the two most critical factors in adjusting the MPL are the number of resources that the workload utilizes and the variability of the transactions' service demands. He also developed a feedback-based controller, augmented by queuing theoretic models for automatically adjusting the MPL.

In many cases, it may not be very easy to obtain and estimate the number of resources (that a workload utilizes) and the service times in a production environment (Manasce, 2004). Therefore, we attempt to construct a model for estimating the multiprogramming level of the system using the number of client threads, the number of request that each client thread makes and the length of the delay before starting each client and the length of the delay before each client makes the next transaction or request. These parameters are simple and easy to obtain from each simulation run. This model may be deployed (or invoked) from the client side. Most admission control algorithm assumes that the transaction takes place synchronously. That is, when a request is made, a response is expected before sending the next request. This assumption does not hold in asynchronous transactions, and this is what makes admission control very challenging.

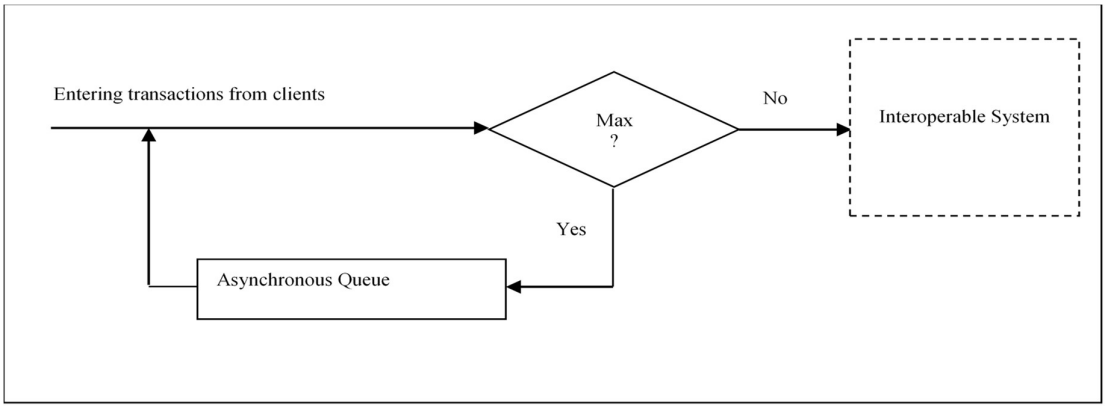

Figure 12. Improving Quality of Service with Admission Control

5.1 Admission Control Based on Response Time

Our approach is to design an efficient admission control that uses a combination of rate-limiting and automated adjustment based on response time. The major difference is that we are using a limit on response time to automatically control the invocation rate. This allows the system to close in on the optimal throughput value. Rate limiting will be implemented before invoking requests (either on the client-side or server-side) while automated adjustment based on latency target will be implemented after receiving responses. A class component is implemented to store the current parameters and to also coordinate the interaction between the two modules that implement rate limiting and automated latency adjustment. Two web services are involved on the client. The first web service submits a request based on the rate-limiting procedure and the second checks and reports the measured latency for a set period.

There are various approaches to rate limiting algorithms. The simplest form of rate-limiting is introducing delays between action executions. The problem with this approach is the difficulty in estimating how long the action will take to execute especially if the action is very complicated as in the case of multiple thread (Menasce, 2004). Another approach is to limit the number of times we execute the action within a period. Yet another approach is to use a rate-limiting algorithm like Leaky bucket (Schroeder et al, 2006). The last two approaches could allow the rate limit to be exceeded, allows for bursts up to two or more time the rate limit are designed for average rate limiting.

Within the context of this study, the following is required of an efficient rate-limiting algorithm:

1. Limit the number of web service invocations within any period of length in such a way that no burst would be allowed.
2. The algorithm would perform a close to the rate limit as possible without exceeding it.
3. Enforce the rate limit whether the action is performed in a single thread or by multiple threads in parallel.
4. Enforce the rate limit whether the action is performed synchronously or asynchronously.

5.2 Description of the Enhanced Algorithm

The proposed algorithm will simply be a distributed algorithm. A distributed algorithm is designed to run on separate processors where each processor is performing a different part of the overall algorithm at the same time and then submitting the results. Distributed algorithms are typically executed concurrently, with separate parts of the algorithm being run simultaneously on independent processors and having limited information about what the other parts of the algorithm are doing. The atomic commit problem is one of the problems that can be solved using a distributed algorithm.

An example of a distributed algorithm for solving the atomic commit problem is the two-phase commit protocol (Manikandan et al, 2012; Samaras, 1995). Other problems that can be solved using distributed algorithm are leader election, consensus, distributed search, spanning tree generation, mutual exclusion, and resource allocation.

5.3 High-Level Description of Proposed Algorithms

This study presents four algorithms that can be implemented separately to improve transactional and security support for interoperable SOA-based systems. In this section, we provide the pseudocode of these algorithms and models. We present a series of algorithms for implementing our transactional model.

Algorithm 1. Distributed Transactional and Security Support (TSS) Algorithm

Input:
A set of n transactions(t_1, t_2, …, t_n)
Output:
A Boolean variable indicating whether or not transaction has committed.
Start:

```
1   set commit to false
2   while(transaction has not committed)
3   {
4           //call procedures to perform tasks
5           call error procedure
6           call coordinator procedure
7           call scheduler procedure
8           call secure-tracer procedure
9           attempt to commit
10          if(commit is possible)
11          {
12                  report success
13                  set commit to true
14          }
15          else
16          {
17                  report failure
18          }
19  }
```

Algorithm 2. Admission Control algorithm based on response time target

Input:
A set of n clients(c1,c2, …, cn),a set of m transactions(t1, t2, …, tn), targetresponsetime, targetthroughput, setperiod, servicedemand, invrate
Output:
avg_responsetime, min_responsetime, optimal_throughput
Start:

```
1   set  done to false,
2   set target_throughput to targetthroughput
3   target_responsetime to targetresponsetime
4   set min_responsetime to targetresponsetime
5   set measured_responsetime to 0
6   set sumresponsetime to 0
7   set set_period to setperiod
8   set txncount to 0
9   set service_demand to servicedemand
10  set inv_rate to invrate
11  while(!done)
12  {
13     open queue
14     for each transaction in the queue
15     {
16        set starttime to currenttime
17        compute initial stoptime
18        increament txncount
19        compute measured_responsetime
20        if (measured_responsetime is less than min_responsetime)
21        {
22           set min_responsetime to measured_responsetime
23        }
24        compute sumresponsetime
25        if (currenttime is greater of equal to stoptime)
26        {
27                  compute avg_throughput
28                 compute avg_responsetime
39                  reset txncount and set sumresponsetime to 0
30                  set next stopttime to compute response time  and throughput
31        }
32        if (avg_responsetime is greater than target_responsetime)
33        {
34                  Decrement inv_rate
35                  Decrement throughput rate
36        }
37        else
38        {
39                  Increcrement inv_rate
40                  Increment throughput rate
41        }
42     }
   }
```

Algorithm 2. Secure-tracer algorithm

Input:
A set of n transactions($t_1, t_2, \ldots, t_n$)
Output:
A set of m suspicious transactions($t_1, t_2, \ldots, t_m$) such that $m \leq n$
Start:

```
1   Connect to central log
2   foreach(transaction in central log)
3   {
4       Sender send message to recipient
5       Sender sends message to central log
6       Create a message correlation identifier(id, timestamp)
7       Recipient sends reply to sender
8       Recipient sends message to central log
9       Locate correlation ID
10      Apply security processing logic
11      if(message is suspicious)
12      {
13          Put suspicious messages in alert queue
14      }
15  }
16  Report suspicious messages for attention
```

Algorithm 3. Concurrent transactions scheduler (asynchronous queuing) Algorithm

Input:
A set of n transactions($t_1, t_2, \ldots, t_n$)
Output:
A Boolean variable indicating completion
Start:

```
1   Coordinator(client) sends message to message queue
2   Scheduler(server) connects to message queue
3   foreach(transaction in message queue)
4   {
5       Make asynchronous call
6       Process message
7       Receive asynchronous response
8   }
9   Report completion of message processing
```

6. CONCLUSION AND FUTURE WORK

This paper has presented a design model for implementing an interoperable SOA-based system in which there is a need for transactional and security support. A case study of a procurement system was used to illustrate the design model. Thereafter the web service transaction model was integrated into the application structure of a procurement system. A description of the procurement system, as well as its architecture, has been provided. This was followed by an illustration of how the Transaction Coordinator component of our proposed architectural framework (an extension of a standard web service transaction framework) maps to the architecture of the procurement systems to provide transactional and security support for the procurement system.

The Distributed Transactional and Security Support (TSS) Algorithm have also been provided to enhance the implementation of the design model to be implemented to provide transactional and security support for the interoperable system. In future we plan to incorporate the algorithm into an interoperable system to evaluate its performance regarding transactional and security support.

REFERENCES

1. Alrifai M., Dolog P., Nejdi W. (2006). Transactions Concurrency Control in Web ServiceEnvironment. Web Services European Conference on, pp. 109-118, Fourth IEEE European Conference on Web Services (ECOWS'06), 2006.
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. Samaras, G., Britton, K., Citron, A., & Mohan, C. (1995). Two-phase commit optimizations in a commercial distributed environment. Distributed and Parallel Databases, 3(4), 325-360.
4. Manikandan, V., Ravichandran, R., & Francis, F. S. (2012). An Efficient Non-Blocking Two Phase Commit Protocol for Distributed Transactions. International Journal of Modern Engineering Research, 2(3), 788-791.
5. Menasce, D., Almeida V., Dowdy, L. (2004). Performance By Design: Computer CapacityPlanning by Example. Pearson Education, Inc. New Jersey, USA.
6. 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 125-144. Available online at www.isteams.net.cisdijournal
7. Schroeder, Bianca; Harchol-Balter, Mor; Wierman, Adam; Iyengar, Arun; and Nahum, Erich(2006): How to Determine a Good Multi-Programming Level for External Scheduling. Computer Science Department. Paper 878. Retreived on September 19 from http://repository.cmu.edu/compsci/878
8. Connolly, B. (2021). *Capturing and Analyzing Client Transaction Metrics for .NET-Based Web Services*. Microsoft. Retrieved from https://docs.microsoft.com/en-us/archive/msdn-magazine/2004/july/capture-and-analyze-client-transactions-with-net-web-services

Appendices

Appendix A - Table Specification

1. Product database

Table 1. Product table

SN	Field Name	Data Type	Field Size	Sample Data
1	ProductNo	Number(PK)	Long Integer	1
2	ProductName	Text	255(default)	A
3	ProductPrice	Number	Long Integer	34
4	ProductQty	Number	Long Integer	10
5	ReorderLevel	Number	Long Integer	5
6	TargetLevel	Number	Long Integer	10

(ii) ProductReqeust Table – stores the details for generating new product request

Table 4.2 – Product request table

SN	Field Name	Data Type	Field Size	Sample Data
1	ID	Autonumber(PK)	Long Integer	2
2	ProductID	Text	255(default)	21
3	RequestID	Number	Long Integer	1263

2. Procurement database

(i) OrderTransaction Table – stores transaction details on the server side.

Table 4.3. OrderTransaction Table

SN	Field Name	Data Type	Field Size	Sample Data
1	TransaNo	AutoNumber	Long Integer	1
2	TransactionID	Text	255(default)	74d5a6ec-185e-476a-8917-fb341aee3ef0
3	OrderID	Number	Long Integer	88309
4	ServerStartTime	Date/Time	General date	8/23/2012 2:42:15 PM
5	ServerEndTime	Date/Time	General date	8/23/2012 2:48:15 PM
6	TransactionStatus	Text	50	In progress

(ii) **OrderDetailsSetting Table** – stores details of orders placed by customers

Table 4.4. OrderDetailsSetting Table

SN	Field Name	Data Type	Field Size	Sample Data
1	ID	AutoNumber	Long Integer (indexed)	1
2	TransactionID	Text	255(default)	74d5a6ec-185e-476a-8917-fb341aee3ef0
3	OrderID	Number	Long Integer	88309
4	ProductID	Number	Long Integer	88309
5	Quantity	Number	Long Integer	88309
6	OrderDate	Date/Time	General date	8/23/2012 2:42:15 PM
7	OrderStatus	Text	255(default)	In progress

3. Metrics Database

(i) StartTransaction Table – stores client details that are generated just before a web service call is made.

Table 4.5. StartTransaction Table

SN	Field Name	Data Type	Field Size	Sample Data
1	ID	AutoNumber	Long Integer	1
2	ClientID	Text	16	d1046289-1e58-46f6-aec9-322f33e62bfc
3	ClientStartTime	Date/Time	General date	8/23/2012 2:42:15 PM
4	TransactionType	Text	255(default)	Place order

(ii) EndTransaction Table - stores client details that are generated after receiving response of a client's asynchronous request is received.

Table 4.6. EndTransaction Table

SN	Field Name	Data Type	Field Size	Sample Data
1	ID	AutoNumber	Long Integer	1
2	ClientID	Text	10	d1046289-1e58-46f6-aec9-322f33e62bfc
3	ClientEndTime	Date/Time	General date	8/23/2012 2:42:15 PM
4	TransactionStatus	Text	50	Completed

(iii) TransactionAttributes Table – stores attributes of the simulated transaction.

Table 4.7. TransactionAttibutes Table

SN	Field Name	Data Type	Field Size	Sample Data
1	ID	AutoNumber	Long Integer	1
2	TransactionAttribute	Text	255(default)	Number of clients
3	Attributevalue	Text	255(default)	5