

QServ: Integrating Testing and Auditing into QoS Management of Web Services

Carolyn M. Ortega
Computer Science Department
City University of New York Graduate Center, USA
ortegac@mail.montclair.edu

Abdullah Uz Tansel
Computer Science Department
City University of New York Graduate Center, USA
tansel@baruch.cuny.edu

ABSTRACT

In a web service environment, service requesters are able to locate functionally equivalent services dynamically making quality of service (QoS) the differentiating factor amongst the web services. Service providers need to formulate QoS aware services in order to remain competitive and to achieve the highest possible profit from their offerings. There are several quality attributes to consider in any operating environment and we've grouped these requirements into 5 major categories: Service Dependability, Architectural Flexibility, Operational Capability, Risk Exposure and Financial Accountability. In the web services environment the realization of the attributes in these quality categories has increased in complexity due to the distributed and dynamic nature of the environment. While much of the research, standards and specifications address these issues, to the knowledge of the authors, an end to end solution for managing the quality attributes in a web service environment that include both testing and auditing has not been proposed. This paper will describe some of the current research that has been conducted to address the various aspects of quality as well as introduce the design for an end-to-end solution that will include testing and auditing.

INTRODUCTION

It is critical that the outcomes of a service in SOA or a web services environment are consistent, reliable, valid and expected to ensure continued use and success of the service as well as for compliance with various regulations which are imposed on the business. From an auditing perspective, the responsibility of compliance to regulations like HIPPA, SOX, FISMA, IPEDS and GAISP [Robinson, 2005] fall in the hands of the Information Technology (IT) department. From a testing perspective, we must be certain that our services behave as expected functionally and performance wise especially when the services are discovered dynamically. Testing and validating our services are strongly desirable to ensure reliability of our business services. In many of our IT environments we are challenged to meet the requirements of the regulations put before us. That challenge has now been multiplied exponentially with the introduction of Service Oriented Architecture (SOA) and Web Services into application development.

There are many advantages that can be realized through Web Services but some of these advantages also raise concerns from a quality control perspective. We must be able to monitor how our environments are changing and demonstrate that as these rapid changes take place that we are in control, in compliance and do not compromise on quality. The goal of this paper is to define methods for various aspects of quality control like QoS management, testability and auditability in a SOA and Web Service enabled environment to achieve compliance.

The paper is structured as follows: First we will discuss quality attributes in a SOA/Web Services environment and provide definitions and categorizations of these qualities. Second, we will discuss current approaches for specifying and managing Quality of Service (QoS) attributes in a SOA/Web Services environment. Third, we will review some of the research in testing methods for web services. In the fourth section of the paper, we will review current research methods in auditing of web services and in the final section of the paper we propose a design for an end to

end QoS management system, QServ that will include service selection, service composition, quality management, testing and auditing.

QUALITY CONTROL ATTRIBUTES

O'Brien (2005) has identified 13 quality attributes to consider in any operating environment. We have grouped these requirements into 4 major categories: Service Dependability, Architectural Flexibility, Operational Capability and Risk Exposure. In addition, we have identified a fifth category; Financial Accountability. The objective would be to maximize service dependability, architectural flexibility and operational capability and to minimize risk exposure in the web service environment. Financial accountability would contain attributes that may be affected by the service provider's ability to achieve success or failure in the other categories. The service requester could then make service requests specifying the desired quality that must be achieved during web service execution using these four categories.

Reliability, scalability, availability and performance are all quality attributes that contribute to the overall service dependability of the web service environment. Reliability refers to the integrity and dependability of a service over time. It includes message reliability as well as service provider reliability. Availability is defined as the amount of time that a system is up and functioning as a percentage of the total time it should be operating. Scalability is the ability for all services to function without degradation as the user's needs increase. Performance is quantified based on response time, throughput and overall timeliness. The combination of these 4 attributes can provide us with a Service Dependability metric. The metric can be calculated by measuring various aspects of the execution of the actual service or it can be derived through customer feedback.

Extensibility, modifiability, adaptability and usability are all qualities that describe the Architectural Flexibility of the web service environment. Extensibility is the ability to extend services or parts of the system. Modifiability is defined as the ability to make changes to a system quickly and cost effectively. Adaptability is the ability to adjust system services based on changing requirements and business needs. Usability describes the efficiency, clarity, ease of use and elegance of the design of the user interface systems. Two of the major goals of a web service environment are the loose coupling of services and the reusability of services. These attributes are very much related to this goal.

Operability, Deployability and interoperability are all quality attributes of a service oriented environment that describes the Operational capability of the environment. Operability and Deployability illustrate the ability to create and maintain a heterogeneous collection of hardware, middleware and software from multiple vendors that make up complex operating environments. Interoperability promotes the communication and exchange of information between multiple entities on potentially disparate hardware platforms, operating systems and programming languages.

Security, testability and auditability are all qualities that relate to the overall risk exposure of a service oriented environment. Security is related to the access control, non-repudiation, confidentiality, authenticity and integrity that can be maintained or achieved by a service. Testability describes the extent to which a system or services makes possible the ability to develop test criteria and to be able to determine that those criteria have been met. Auditability is the ability of a service to maintain transaction records to support specified financial or legal audits. Providing a metric on operational capability, architectural flexibility and risk exposure, whether calculated or derived, could be informative to service requesters that have these types of requirements for their business processes.

Financial accountability differentiates web services in terms of their value proposition and consists of attributes such as service charge, compensation rate and penalty rate. The service charge is the cost for the client to use the service. The compensation rate is the percentage of the original execution price that will be refunded when the service provider cannot honor the committed service. The penalty rate is defined as the percentage of the original price service users need to pay the provider when they want to cancel the committed service. The compensation rate may be directly affected by the attributes in the other 4 quality control categories (Yeom et al., 2006).

SPECIFICATION AND MANAGEMENT OF QOS FOR WEB SERVICES

Two types of QoS approaches have emerged in current research to address QoS issues in web services. The first approach proposes a new infrastructure for specifying QoS requirements within Web services and the second approach involves extending functionality of the UDDI either within or outside of the UDDI by introducing an additional server or broker (Tian et al., 2001).

In the first approach, researchers have proposed extending the definition of the UDDI to include Service Level Agreements (SLA) offered by the service providers. Tian et al., (2001) has summarized three offerings that use this approach. The first offering is WSLA which was developed at IBM and provides individually negotiated and customized service level agreements for each customer (Keller, 2002). The design goal of WSLA is to develop a formal and flexible XML based language for SLA definition for each client. The second offering is WSOL which was developed at Carleton University and it provides various levels of service using the same functional specification which differ in QoS level, monetary penalties, management responsibilities and 3rd party accounting (Tosic et al., 2003). The third offering is SLAng which is an XML based language for defining service level agreements with support for inter-organizational service provisioning of resources like storage, network, middleware and applications as well as non-functional parameters of an SLA. SLAng contains vertical and horizontal SLAs which consist of layers like an application layer, hosting layer, persistence layer and a communication layer. A horizontal layer is a contract based on services (a component and web service provider) and may consist of a service peer, container peer and networking peer. SLAng is not just for web services and does not support dynamic lookup of new services or an update of non-functional service properties at runtime.

In the second approach, researchers have proposed introducing new infrastructure into the environment to manage, monitor or broker the quality of service of the web services [Tian et al, 2001]. An example of this approach is UX, which is a server that collects QoS information about providers and can be queried by clients. Service providers develop reputations which are then shared amongst customers. The UX server is extended with an inquiry interface that conforms to the standard UDDI one. Another example is UDDIe. UDDIe proposes to extend the UDDI within the UDDI and provide services using a leased based approach. Service providers specify QoS properties such as bandwidth, CPU, and memory requirements. Services can be made available for a period of time or on a lease based fee. The lease based option includes 3 leasing types: finite, infinite and future. The client sends their request to the QoS broker of the G-QoS framework (grid service discovery) system. The broker is not a part of the UDDI. The broker makes the selection of appropriate service based on a weighted average concept.

Another infrastructure for managing QoS that has been proposed in current research is QBroker (Yu, 2006). QBroker is an end to end QoS Management system that provides service tracking, service planning, service selection, QoS value adaptation and service re-optimization. The service tracking module identifies available services as well as collects their QoS information. The service planning module finds all possible combinations for building a composite service. The service selection module determines the best service for each component of the composite service process. The QoS value adaptation module is used to update the QoS values of each service as new values are reported. There are two versions of QBroker: Offline tool (vBroker) which acts as an offline system design, planning and analysis tool, and the Online service, Qservice that provides online decisions for service requesters to construct optimal business processes under the end to end QoS constraints.

We would like to propose an end to end QoS model that includes a combination of the methods described above, in addition to the testing and auditing of services. First we will examine the different methodologies developed for testing in a dynamic web services environment. We believe that a dynamic QoS Management model should address testing which contributes to the overall quality of the services and in some cases is a necessary condition that must be satisfied before execution of the services.

TESTING ISSUES IN WEB SERVICES

The dynamic discovery and invocation capability of web services provide little opportunity for testing beforehand and complicates the issue of testing in general. A service subscriber is limited to black box testing of a service through information that is provided through the WSDL file of the service. In addition, a single business process may involve multiple web services or services that call other services which further complicates the testing dilemma. Methods for testing web services should cover test scenarios that include and test the discovery of the web service, the data formats exchanged between the services, the request/response mechanisms, the invocation sequences, hierarchical functional dependencies amongst web services and input/output dependency between web services.

Current research has proposed that additional information be provided in the WSDL to perform additional testing (Tsai et al., 2002).

In Offut and Xu (2004) a method for testing request/response interactions in Web services is described using Data perturbation. Web services interaction may be multilateral or peer-to-peer in nature. Data perturbation provides request/response testing for peer-to-peer web services. The tests are based on XML messages that can be represented using a regular tree derivation and the process consists of modifying request messages, resending the modified messages and analyzing the response message for the correct behavior. There are two types of data perturbation testing: Data Value Perturbation (DVP) and RPC Communication Perturbation (RCP). In DVP the values in the SOAP messages are modified according to the boundary value rules defined on the types of values in the message. The messages are sent using these boundaries and the responses are analyzed. In RCP testing the testing consist of remote procedure calls through messages. Mutation analysis is used to generate various message values which are passed to the remote procedure call. The remote procedure call acts on the messages that are passed through its normal functioning and returns a message as the value. The mutated messages sent to the remote procedure call and the messages sent back from the remote procedure call are used to analyze the behavior of the service.

Another method that has been proposed for facilitating testing of web services requires that additional information be specified in the WSDL file of the services so that the client may develop more meaningful tests. In [Tsai et al, 2002], the author describes the type of information that would be desired in the WSDL file that could enable the client to perform Input-Output Dependency testing, Invocation sequences of services testing and hierarchical service dependency testing. Providing this information would be considered an extension to the current WSDL specification. Currently, the only type of testing that can be performed is black box testing of services since no access to the code is provided. Current specifications for WSDL only provide the number of input and outputs, the variable types of each input and output, the order of inputs that are given and the outputs returned in addition to how a web service should be invoked. Information such as the dependence between input and output of the web services and invocation sequence cannot be obtained from the WSDL file.

Specifying Input-Output dependency information in the WSDL file would help address issues in regression testing and data flow testing. To address Input-output dependency, the WSDL schema would be extended to include a complex type, `WSInputOutputDependenceType` which includes at least one element of type `WSIOPairType`. The `WSIOPairType` is also a complex type which has two sub-elements, input and output of `WSIOModeType`. The information would be automatically generated by dependence analysis within a web service. The association of input and output between web services will help to fine-tune the testing process by clearly identifying necessary tests for regression testing and clarifying what expected results should be (Tsai et al., 2002).

Invocation Sequence testing would identify and trace the calling relationships between services. A web service reference type which has a web service name and the link to its' WSDL document would be included in the WSDL document. The complex type `WSInvocationDependenceType` would have two sub-elements; `WSICallers` to represent the callers and `WSICallees` to represent the callees. Both of these sub-elements would be of type `WSICalleeTypes` and `WSICallerTypes` respectively which would have a link to the WSDL files being called as a reference. By tracing this information among participating web services it is possible to generate the complete calling sequence of the web service. This is useful in path testing and data flow testing.

Providing Hierarchical Functional Descriptions within the WSDL file would provide additional information beyond structural types like dependency and calling sequences. The WSDL would be extended to include sub-elements; `WSFParents` and `WSFChildren` which are of `WSFParentType` and `WSFChildrenType` respectively. For both of these types, the information contained would be a link to their corresponding WSDL files that would provide the hierarchical structure between services. By using this functional description, it becomes possible to provide analysis such as functional dependency testing and enhanced regression testing.

Sequence specifications capture calling sequences, current behaviors and timing aspects for real-time systems. For example providing the sequence `OpenAccount`, `Deposit`, (`Deposit` or `Withdraw`), `CloseAccount` could be useful in deriving appropriate test scenarios and reduce effort and costs to perform and automate these tasks in their business process. Including sequence information in the WSDL files would provide service subscribers with the ability to set up these types of test scenarios (Tsai et al., 2002).

AUDITING IN A SOA/WEB SERVICES ENVIRONMENT

The question of how we monitor for compliance is an important one. There are two methods of change that we believe should be monitored. The first method is to monitor the dynamic composition of the environment. The second method is to audit service process flows that have been executed. We believe that there is a direct correlation between the extent to which testing has been performed on services prior to their use in a dynamic production environment and the degree of monitoring or auditing that is required to limit the risk exposure. This relationship will be explored in detail in future research.

The first method for ensuring that the SOA architecture meets the necessary audit and compliance rules is described in (Deubler et al., 2004). This method requires that verifications of the process flows and services be performed prior to their execution as services in production. Areas of vulnerability and audit potential process flows can be detected during this phase and then tested. The aim is to ensure that the necessary security controls are implemented during the development and testing process. This effort should decrease the need for, or simplify the audit process. We’ve extended this concept to include elements and functionality that will address various testing scenarios like input/output dependency and regression testing.

Services can be viewed as system sub-functions which represent a pattern of interactions or a process flow in an application. The behavior of a service can be specified informally by associating its inputs (In Var –stands for variable) and outputs (Out Var) as well as by its state which is determined by local variables (Local Var) when known. The service behavior is defined for valid sequences that make up process flows along with the services associated messages. Since services are orchestrated to communicate with one another via connecting ports (source and destination), we are also able to build up execution scenarios which can then be used for the basis of a behavioral verification of security or audit properties. A partial view of the services schema in Table 1: Service Table provides a sample of process execution. Additional views and schema that handle testing detail will be defined in future research to support the entire process of pre-execution verification and testing.

Table 1: Service Table.

| Source | Destination | In Var | Local Var | Out Var | Proc ID | Requestor | TimeStamp |
|--------|-------------|--------|----------------|---------|---------|-----------|-----------|
| 11 | 23 | 20 | 20, 21, 22, 23 | 23 | 1 | 506 | 10:23:32 |
| 23 | 56 | 23 | 23, 11 | 11 | 1 | 506 | 10:23:52 |
| 56 | 7 | 11 | 11,23 | 23 | 1 | 506 | 10:24:11 |
| 7 | 87 | | | Print | 1 | 506 | 10:24:44 |
| 11 | 63 | 112 | 112, 111, 15 | 15 | 2 | 711 | 10:40:00 |
| 63 | 23 | 15 | 15, 45, 48, 93 | 93 | 2 | 711 | 10:41:32 |
| 23 | 57 | 93 | 93, 2 | | 2 | 711 | 10:42:30 |

Deubler et. al., (2004) describes a development process for service based systems which integrates the security and necessary checks as a part of the process. The process consists of four service-specific phases: service identification, use case modeling, service modeling and component design.

In the service identification phase, services are specified to be actors in activity diagrams (Deubler et. al., 2004). The use case modeling phase identifies the flows of events of the service functions and is the initial phase where the security and audit requirements are denoted. The services are then mapped to requirements in the formal models of the execution scenarios.

In the service modeling phase, behavior of the services are formally specified and the two main security concerns, authentication and authorization are analyzed. In service based systems it is possible that information for access control decisions can be distributed over more than one service. In this scenario, access control enforcement becomes more complicated as multiple services must be taken into consideration when validating security. An authentication and authorization predicate are used to specify parts of the system's security policy (Deubler et. al., 2004). The last phase is called component design and in this phase services are mapped to system components.

Another area for identifying security related services are those which may perform critical operations like reading from or writing to a database. We can identify the process flows which access data and provide the associated auditing rules to ensure that data is only disclosed when appropriate.

There are tools that can be used for modeling and verification of service based systems like AutoFocus (Deubler et. al., 2004). AutoFocus is a case tool for graphically specifying distributed systems. Systems are specified using static and dynamic views which are conceptually similar to those offered in a UML profile for component based communicating systems. Through threat analysis and model checking it is possible to identify the security and audit problems that may be present in an orchestration of services before they are made available to the public as a business process or service. Regardless of the tools that are used for modeling and verification, the concepts presented in this paper would still apply.

In the second method, Hacigumus (2006) describes how service process flows that have been executed can be inspected based on their configuration as well as by the data that was disclosed. The interactions among the service elements are defined as the process flows, which are essentially the execution of the higher level business processes. The history of the processes which ran over time is captured in a log which consists of process flows which were executed. By examining these process flows, suspicious process patterns can be identified. This is referred to as a structural audit. In addition, the ability to define process flows that could potentially disclose confidential or sensitive information is identified. The executed process flows are reviewed to determine if any of these processes have been executed in the system. This type of audit is referred to as a data disclosure audit. In order to determine which processes could have posed a security breach, two different types of logs are maintained to facilitate the auditing process: process logs and data access logs.

A user service request can be modeled as a sequence of web service element calls in the system. The web service information that is stored in these logs are <Source, Destination, Variables, ProcessID, Requestor, Timestamp> which is another view from the Table 1: Service Table; where the Source is a service element that initiates the service call, the Destination is a call from the source to another service element, the Variables hold any input parameters or references for the service call and the Process ID provides the capability of grouping the service calls as a unit of execution. Once the raw audit logs are captured they can be searched to construct and execute audit trails.

In the examination of process flows that have already been executed, the aim is to identify the process flows that violated the systems security and privacy policies. The first step to achieve this is to eliminate the process flows that are outside of the scope of the audit from the audit logs for a given process flow. For example, if the audit is looking for processes executed by a particular user, eliminate the processes that do not meet those criteria.

Table 2: Service Table for Specific Requestor shows the services that were executed by requestor 506 only. We can further eliminate processes based on the timestamp value and so on. The remaining process tuples are the ones that should be examined as a part of the audit.

Table 2: Service Table for Specific Requestor.

| Source | Destination | In Var | Local Var | Out Var | Proc ID | Requestor | TimeStamp |
|--------|-------------|--------|----------------|---------|---------|-----------|-----------|
| 11 | 23 | 20 | 20, 21, 22, 23 | 23 | 1 | 506 | 10:23:32 |
| 23 | 56 | 23 | 23, 11 | 11 | 1 | 506 | 10:23:52 |
| 56 | 7 | 11 | 11,23 | 23 | 1 | 506 | 10:24:11 |
| 7 | 87 | | | Print | 1 | 506 | 10:24:44 |

For the structural audit, the logs are searched for matches between the process flows that were identified by the audit as suspicious and the remaining potential process flows. For the data disclosure audit we first identify what is referred to as data operation nodes. Data operation nodes access the information sources in the system’s data processing elements that search for specific information. For example, a data processing element may execute an operation which searches for records that satisfy a given condition. Once the data processing elements have been identified, a check is performed to see if any of these elements were executed (Hacigumus, 2006).

THE PROPOSED QOS MANAGEMENT MODEL INCORPORATING TESTING AND AUDITING

Implementing testing and auditing of web services as part of an end to end QoS management system in a dynamic environment has not been considered to the best of our knowledge in current research in the area of quality management. We believe that incorporating these quality attributes into the QoS management paradigm will provide greater reliability and availability of the orchestrated web service.

It is our intent to design a QoS Management system, QServ that will manage SLA requirements like service dependability and risk exposure, as well as handle testing and auditing in a dynamic web service environment. In order to create an end-to-end QoS Management system that includes testing and auditing, there are several tasks that must be handled by the system. These tasks are represented in the Process Composition Manager, Process Test Manager, Process Request Manager and the Process Audit Manager.

The Process Composition Manager must be able:

To receive specifications of services from developers for process composition.

To identify multiple possibilities of services for process composition. These services should be functionally equivalent, meet minimum QoS service requirements and be interchangeable in the process plan. The services are discovered through communication with the UDDI and other QServ managers. A link to the WSDL document of the discovered services are stored in the service repository and the services which make up the process plans which are stored in the process plan repository.

To coordinate testing of services with the Process Test manager. This coordination also involves updating the service repository and process plan repository with relevant and up to date testing scores for the services and processes. It should be noted that under conditions of low risk exposure, services may be discovered dynamically and put into production prior to testing taking place. However, these services that have not been tested cannot be promoted to lower levels of risk exposure until all designated phases of testing have been performed.

To coordinate updating the service repository and process plan repository with relevant metrics relating to QoS and audit monitoring. Services which violate audit policies should either be removed from the service and process plan repository or updated to reflect a higher risk exposure.

The Process Test Manager must be able:

To perform various test at the service level like testing data formats exchanged between the services, the request/response mechanisms, the invocation sequences, hierarchical functional dependencies between services and input/output dependency between services as well as performance testing.

To perform a complete test of the process plan.

To maintain test scores which may also be updated through actual production execution and auditing

The Process Request Manager must be able:

To accept QoS process request from the service requesters which may have specific QoS service requirements at the individual service level or at the overall QoS requirements level for the process.

To select services based on their QoS metrics and testing scores obtained through the process plan repository

To execute the process plan and store the execution data in the executed process flows repository.

To update the process plan repository with QoS data based on real execution information derived from the executed process flow repository.

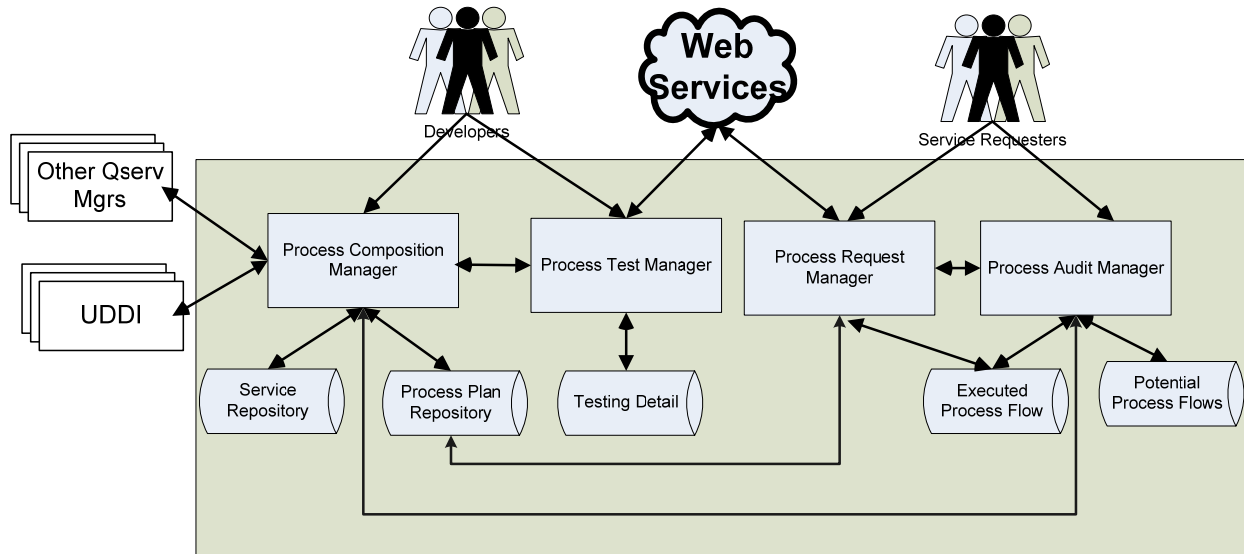
To accept QoS feedback from the service requesters about the process plan and to update the process plan repository appropriately

The Process Audit Manager must be able:

To audit service interaction in an environment dynamically and on demand. Structural audits or dynamic monitoring can be invoked automatically through programmed user audit requirements. If services with low test scores or process plans with high risk exposure are used, the audit of the process transaction may be triggered automatically. Additionally, the service requester may directly ask for auditing to be performed of the process transactions.

To coordinate with the process composition manager in the development of the potential process flows that could violate system security. These potential process transactions would be stored in the potential process flow repository.

Figure1: QServ Architecture.



Our initial design for the QServ management system is depicted in Figure1: QServ Architecture. QServ would manage QoS metrics for business processes, provide testing of the services that make up the business processes as well as provide audit tracking of the services. The processes would be represented by an orchestration of web services where each service would have associated with it, its own QoS metrics and the overall QoS requirement for the business process would be derived from the aggregation of QoS requirements from each service using an appropriate algorithm. This overall QoS requirement would be used when requesting services to indicate the level of quality required for the business process. The services would be optimally selected to meet the overall QoS requirements using an appropriate algorithm. The process of selecting web services that exhibits a general flow structure (i.e. composite services containing sequential, parallel, conditional or loop structures) and have more than one QoS requirement is a combinatorial problem that can be characterized by the 0-1 Integer problem. Yu (2006) has developed two algorithms; WS_IP and WFlow to solve the QoS service selection problem.

As a part of the QoS requirement, the service requester would be able to specify a minimum testing score requirement in addition to other QoS attributes like availability and performance. The testing score requirement would dictate the degree to which dynamic composition could be used in the web service selection as well as indicate how much testing should have been performed with these services prior to use in the business process. If the service allows dynamic service selection with low testing scores yet has a security requirement, then dynamic auditing may be required to ensure that any new functionally equivalent and acceptable services that are introduced into the business process at service selection time are behaving appropriately. The use of dynamically located, untested services would still be an option; however, for some business processes the need for auditing becomes more critical in this case. In the event that a service exhibits suspicious behavior which violates a predefined audit scenario, then the appropriate action will be executed and the service repositories would be updated to reflect this new information.

CONCLUSION

We observed that there is a correlation between the amounts of pre-qualifying testing that have been performed in the environment and the degree of dynamic auditing or monitoring that is necessary during execution. This observation, the investigation of the correlation between testing and auditing of web services, the design of an end to end QoS Management system and the testing methods and audit methods required will be the basis of our future research.

REFERENCES

Deubler, M., Grunbauer, J., Jurjens, J., & Wimmel, G. (2004). Sound Development of Secure Service-based Systems, *ACM*, 115-124.

- Hacigumus, H. (2006). Middleware Support for Auditing Service Process Flows, *ACM MW4SOC 2006*, November 27-December 1, 2006, Melbourne, Australia, 24-29.
- Keller, A. & Ludwig, H. (IBM) (2002). The WSLA Framework: Specifying and Monitoring of Service Level Agreements for Web Services, *IBM Research Report RC22456*.
- O'Brien, L., Bass, Len, M. & Paulo (2005) *Quality Attributes and Service Oriented Architectures*, Carnegie Mellon University, 1-38.
- Offut, J. & Xu, W. (2004). Generating Test Cases for Web Services Using Data Perturbation, *TAV-WEB Proceedings of the ACM SIGSOFT*, September 2004, 29 (5), 1-10.
- Pisinger, D. (1995). *Algorithms for Knapsack Problems*, PhD Thesis, University of Copenhagen, Copenhagen, Denmark.
- Robinson, T. (2005). Data Security in the Age of Compliance, *ACM Data Security*, 25 -30.
- Tian, M., Gramm, A., Ritter, H., Schiller, J. & Winter, R., (2001). A Survey of current Approaches towards Specification and Management of Quality of Service for Web Services, Freie Universitat Berlin, Institut fur Informatik, 1 – 13.
- Tosic, V., Pagurek, B., Patel, K., Esfandiari, B. & Ma, W., (2003). Management Applications of the Web Service Offerings Language (WSOL), *The 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Velden, Austria.
- Tsai, W. T., Paul, R., Wang, Y., Fan, C. & Wang, D., (2002). Extending WSDL to Facilitate Web Services Testing, *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, 1-2.
- Yeom, G., Yun, T., & Min, D. (2006). A QoS Model and Testing Mechanism for Quality Driven Web Services Selection, *Proceedings of the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, IEEE, 1-6.
- Yu, T., (2006). *Quality of Services in Web Services: Model, Architecture and Algorithms*, Dissertation, University of California.