QoS AWARE SERVICE ORIENTED ARCHITECTURE

Sagarika Adepu

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2013

APPROVED:

Krishna Kavi, Major Professor
Bill Buckles, Committee Member
JungHwan Oh, Committee Member
Barrett Bryant, Chair of the Department of
Computer Science and Engineering
Costas Tsatsoulis, Dean of the College of
Engineering
Mark Wardell, Dean of the Toulouse
Graduate School
Adepu, Sagarika. *QoS aware service oriented architecture*. Master of Science (Computer Science), August 2013, 45 pp. 8 figures, 30 numbered references.

Service-oriented architecture enables web services to operate in a loosely-coupled setting and provides an environment for dynamic discovery and use of services over a network using standards such as WSDL, SOAP, and UDDI. Web service has both functional and non-functional characteristics. This thesis work proposes to add QoS descriptions (non-functional properties) to WSDL and compose various services to form a business process. This composition of web services also considers QoS properties along with functional properties and the composed services can again be published as a new Web Service and can be part of any other composition using Composed WSDL.
# CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Service/WSDL/Service Composition</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Project Goals</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Proposed Solution</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Thesis Outline</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER 2. BACKGROUND / RELATED WORK</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Background</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>8</td>
</tr>
<tr>
<td>CHAPTER 3. TOOLS / ORCHESTRATION</td>
<td>16</td>
</tr>
<tr>
<td>3.1 WSDL</td>
<td>16</td>
</tr>
<tr>
<td>3.2 SOA / BizTalk</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Consuming and Orchestration of Web Services</td>
<td>18</td>
</tr>
<tr>
<td>3.3.1 Creating Web Services</td>
<td>19</td>
</tr>
<tr>
<td>3.3.2 Adding Web References</td>
<td>20</td>
</tr>
<tr>
<td>3.3.3 Generating Basic Schema</td>
<td>22</td>
</tr>
<tr>
<td>3.3.4 Property Promotion on Schema</td>
<td>23</td>
</tr>
<tr>
<td>3.3.5 Schema Mapping</td>
<td>24</td>
</tr>
<tr>
<td>3.3.6 Constructing Web Messages</td>
<td>24</td>
</tr>
<tr>
<td>3.3.7 Orchestration</td>
<td>25</td>
</tr>
<tr>
<td>CHAPTER 4. METHODOLOGY / IMPLEMENTATION</td>
<td>27</td>
</tr>
<tr>
<td>4.1 QOS Aware WSDL</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Services/Orchestration</td>
<td>30</td>
</tr>
<tr>
<td>CHAPTER 5. DISCUSSION AND CONCLUSION</td>
<td>40</td>
</tr>
<tr>
<td>APPENDIX: GLOSSARY</td>
<td>41</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>43</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 4.1. QoS-Aware WSDL schema for performance parameters ......................... 30
Figure 4.2. Service selection tool ................................................................................ 31
Figure 4.3. Orchestration interfacing ontology model.................................................. 33
Figure 4.4. Web services layered queuing network modeling ...................................... 34
Figure 4.5. Services in orchestration - flow diagram .................................................... 35
Figure 4.6. Components (Basic orchestration) used to build the business flow. (1). individual orchestration. (2) decide orchestration. (3) series orchestration ............. 36
Figure 4.7. Configuring input to the workflow. ............................................................... 38
Figure 4.8. Configuring receive inputs to the orchestration using schema. ................. 38
CHAPTER 1
INTRODUCTION

1.1 Service/WSDL/Service Composition

Service oriented architecture (SOA) offers a flexible methodology for the creation and management of software services. Software services are well-defined business functionalities situated in loosely-coupled and distributed computing settings such as Cloud and Web [1]. Web services are the building blocks with business logic and represent an application of service-oriented computing on the Web which are accessible with the help of Internet protocols. They are independent software components which provide a particular functionality. One can describe, discover and deploy them on the Web using standards such as WSDL, SOAP (which is a transport protocol for exchange of information) and UDDI (which is a registry and is used to store the service). In a service-oriented architecture, the service provider creates a WSDL (web service Description Language) service description and publishes it to one or more discovery registries (such as UDDI), so that service consumers can find the service using a wide variety of search criteria and then use the WSDL description to develop or configure a client that will interact with the service.

Web services Description Language (WSDL) is often used in combination with SOAP and XML Schema to provide web services over the Internet. WSDL is used to determine the functionality of the web service along with its communication protocols. Service providers can register services with Universal Description Directory and Integration (UDDI) or other such registry service using WSDL. Service repository can be
queried by customers to discover needed services. The discovery of a service is based on searching through categories and by matching the specification given in the WSDL.

Sometimes users would like to use QoS (Quality of service / non functional) characteristics such as reliability, performance, etc. in addition to functional characteristics of web services to select among several services providing the same functionality. One way to achieve the specified QoS goals is to select appropriate web services based on their QoS properties so that the composed system can satisfy the QoS requirements. There has been a lot of research work addressing the QoS issues in web services. QoS-aware service composition works, deal with the concrete service selection based on the end-to-end QoS requirements. The problem with QoS-aware service composition is that some selected service providers may have workload and performance fluctuations over time that can result in QoS violations.

The goal of our project is to develop a comprehensive solution and develop a QoS-aware service composition and execution framework to discover services based not only on their functionality but also based on nonfunctional (or quality of service) properties. Our goal includes service composition and specification of non-functional properties of composed services. These goals require the ability to specify non-functional (or QoS) properties with services, and the ability to compute non-functional measures of composed services.

1.2 Project Goals

Goal 1: WSDL extension for the description of QoS characteristics of a web service. In the selection of a web service, consider both functional and QoS properties
in order to fully satisfy the needs of a service consumer. Achieve a composed WSDL of the web service composition which describes the overall composed quality criteria.

At present, WSDL can only be used for specifying functionality of services. Non-functional properties, including several Quality of service (QoS) characteristics, are crucial to the success and wider adoption of web services. Customers would like to use QoS characteristics of web services for selecting from among (to select among) several alternate implementations. Each of the service providers declares similar functionalities for the same purpose – thus the customer expects more information about services. Typical QoS properties are security, reliability, and performance. WSDL should be extended in order to provide QoS related information with services. Once non-functional properties of services are specified, it will be possible to develop or extend tools for the discovery of web services based on both functional and non-functional properties. Additional tools can be designed for service aggregation, integration and composition based on QoS characteristics.

Goal 2: Exposing functionality as web services in a granular fashion is the first step towards providing the basic building blocks required for implementing solutions. A management layer is needed, something which can orchestrate the web services, controlling the flow and interaction between them and aggregating individual services into a larger composite solution.

Each service provides a specific and well defined functionality. Well defined interfaces permit for the discovery and invocation of services. Service repository can be queried by customers to discover needed services. The discovery of a service is based on searching through the categories and by matching the specification given in WSDL.
These services can be treated like an individual processes which can be orchestrated accordingly to form another business process. And finally, we should be able to expose the composition of web services as an independent web service.

1.3 Proposed Solution

Our approach offers QoS composition by composing the parameters of the individual services in the service composition. This approach, thus, supports a more generalized form of QoS-differentiated services, allowing QoS tradeoffs on a generalized set of QoS attributes, beyond performance and cost based attributes offered by the resource management. We have developed a QoS-aware service composition framework to achieve service selection and composition for satisfying QoS requirements of composite systems. Our research seeks to provide a comprehensive mechanism for QoS-assured service composition.

I have extended WSDL with quality information. This research work includes the process of introducing quality information of web services in its WSDL file and to utilize the information during service composition. We propose a framework for service composition along with the assistance of both ontological and performance modeling tools. QoS properties are modeled with the ontological engine that can be expanded in accordance with the service declaration. Properties that are subject to a chosen performance tool can also be noted in the ontology model for further semantic comparisons. In this thesis, we will focus on the service response time aspect of the service. We demonstrate the process of WSDL extension, along with its corresponding QoS ontology modeling, performance modeling, and service composition using an
example. This research work also includes developing a prototype framework and integrating the technologies discussed below to achieve QoS-aware service composition.

In this thesis, I have used the Visual Studio .Net Integration Development Environment (IDE) and C# language to create web services. A web service file will have an .ASMX file extension. Object-oriented programming skills are used to build the classes. Also WSDL is extended to describe QoS properties. This web service can be accessed through HTTP for any use.

I have used Microsoft BizTalk server to orchestrate the web services, controlling the flow and interaction between the web services and aggregating individual services into a larger composite solution [1][2]. BizTalk Server fully supports all the open standards upon which web services are built. Web services in our BizTalk Application were used in the following ways:

1. Consume and orchestrate existing web services.
2. Expose business processes (BizTalk orchestrations) as Independent services

1.4 Thesis Outline

In Chapter 2, necessary background information and related work is outlined. The work related to identifying Non functional parameters, extending WSDL with quality information is discussed. Previous research in the areas related to non functional properties and service oriented architecture is also illustrated briefly.

In Chapter 3, WSDL description and the role of WSDL in our framework are discussed. It also discusses the tools and technologies that are used to consume web
services, orchestrate them and expose them as independent processes. In this chapter, all the steps involved in each of the phases like creating a web service, adding a web reference, basic schema generation, how to promote these properties in schema, constructing web messages and orchestration are briefly explained.

In Chapter 4, approach and methods used to implement our framework are explained. The first part of the chapter explains the extension of WSDL to QOS properties with a sample WSDL that was extending to just one non functional parameter. The second part of the chapter explains our framework for demonstration purpose with a simple example.

In Chapter 5, the results obtained are presented.
CHAPTER 2
BACKGROUND / RELATED WORK

2.1 Background

The next emerging area in the domain related to SOA is the automated aggregation of services using approaches of the Semantic Web. Services build up service processes which represent the workflows of an enterprise. They can be added, deleted, and updated at runtime without interrupting the ongoing business. It is very often the case that only one service cannot fulfill a complex functionality request. As an alternative, to efficiently compose several services for the requested functionality becomes a necessity. In addition, the final solution of a composition is also required to satisfy some QoS conditions, for example, cost, response time or reliability. In many applications QoS is an important issue.

To manage the dependencies above, approaches based on the combination of syntactic and semantic service descriptions like WSDL, to decide whether a service provides a functionality being sought or not, are used. Common syntactic definitions like WSDL specify the order and types of service parameters and return values.

My research work is based on how to achieve the specified QoS goals such that the composed system can satisfy desired QoS requirements. QoS aware service composition works deal with service selection based on end-to-end QoS requirements. However it is easier to specify the overall QoS goals of the composite system, instead of decomposing the global goals into the desired QoS for the individual component services. We plan to develop a simple WSDL file for the composed system with the overall parameters of the composed system instead of properties of the discrete web
services. QoS aware service composition deal with the concrete service selection based on end-to-end requirements.

My research work includes a simple prototype of extended WSDL with just one non-functional QoS parameter that we want to associate with a web service without altering the original content rather than introducing additional languages on top of it is developed. Service providers can thus be endowed with methods and tools to obtain QoS predictions and answer questions like: “what is the expected service quality?” or “what is a satisfactory reconfiguration of the service parameters to guarantee an agreed QoS?” or also “what is the acceptable response time under different workload situations?”

Service composition techniques for satisfying end-to-end requirements have been widely studied. However, the algorithms for aggregating individual service QoS capabilities to derive the overall system QoS properties are preliminary. In this thesis, for easy understanding we have considered simple summation of the QoS property to derive the overall property of the composed system. However, our framework permits more complex composition techniques for evaluating the QoS properties of combined services.

2.2 Related Work

The description of non-functional properties related to SOA operational management has been described in [4]. In addition to adding some QoS criteria, semantic interpretation to the extensions have been realized in various frameworks [5][6][7]. An approach to describing service lifecycle information and QoS guarantees
offered by a service based on OWL-S can be found in [8]. Here, service profiles are appended with QoS Characteristics to generate a corresponding service description repository. The OWL-S based repository can automatically cover the traditional UDDI registry by mapping its elements. In [9], WSDL is extended to XWSDL where non-functional criteria are added in service definition. Following its predecessor X-UDDI [10], the web service registration and publication can be queried on the basis of these criteria. In [11], a unified semantic web services publication and discovery framework is proposed with a QoS Metrics extension to WSDL using PS-WSDL, USQL for service query, and UDDI mapping suites. In this thesis, we focus on a proof of concept for WSDL extension and its correspondent non-functional semantic model engineering, but not on the service registration. With our framework, it should be straightforward to apply well-defined UDDI extension tools such as mentioned in [12], or other registry tools.

In QoS-aware service composition, the composer needs to explore many candidate compositions to find one candidate composition that best satisfies the QoS requirements. If the composer does not contact the providers during the composition process to get the admission information, then the composition solution may be invalid. However, if for each composition candidate, the composer communicates with the providers to obtain the admission information, then the cost to evaluate many candidate compositions can be very high. The problem can be worse when some specific QoS attributes are considered. For example, consider security-aware service composition [21][22][23] QOS aware service composition works assume that there will be a large number of concrete services to choose from. But in some special application domains there might be a limited set of concrete services implementing the same functionality.
Some systems consider differentiated services, which offer different QoS levels for different levels of customers. These works mainly consider cost and performance tradeoffs and offer them through prioritized resource management. However, what the specific QoS levels and corresponding cost should be provided by the services have not been analyzed. An alternative approach that can offer extensive QoS tradeoff space is reconfigurable services. In many application domains, basic entities that can be used to assemble the system are designed to be configurable. For example, there are many applications that are designed to be reconfigurable for different QoS tradeoffs [29][30] and they can be wrapped into services to fit the SOA paradigm. QoS-reconfigurability can also be a service design goal for QoS-critical application domains. These reconfigurable services can increase the selection space at composition time and significantly help with end-to-end system QoS assurance.

There is other research work on QoS-aware service composition [24][25][26] but they only consider service selections. In some specialized application domains, there may not be too many alternative services implementing the same critical functionality. When the QOS requirements for a composite system are very stringent, it is likely that the composition process may fail. In [27] the composer renegotiates with the customer to adjust the QoS requirements. Even with careful negotiation and resource management, the system might encounter unpredictable events, such as failures, and cause violation of agreed upon QoS for the customers. Some research works consider dynamic composition or recomposition [27][28] to handle changes, which takes the same composition techniques but focuses on efficiency of the composition process.
QoS for traditional web applications has been studied within a layered architecture including transport, network, operating system, distributed system, and applications. However, such studies are not sufficient for studying service oriented systems. QoS of web services has, in the past five to six years, been investigated as a separate topic in a number of research projects. A range of research issues regarding QoS provisions on web services have been studied recently, which can be categorized as follows: 1. Methods for processing QoS to be used for web service selection and composition. 2. Models of web service QoS characteristics and related ontologies and semantic languages.

The majority of the development in the provision of quality of service support from web services is (1) to solve the issues of processing methods and (2) of models, ontologies, and semantic languages. Within the first category, service selection and composition are studied from the middleware perspective based on web service QoS. Researchers have also worked on static and runtime scheduling, runtime QoS collection, monitoring and binding/ re-binding, transaction support, privacy and trust support, and QoS negotiation and matching. These works form the basis of working toward a general solution to optimizing service oriented systems. A few research efforts have taken into consideration user satisfaction levels on web service usage and end-to-end or application QoS management. Within the second category, researchers have focused on defining models of QoS characteristics, and related ontologies and semantic languages, and constraints of service attributes.

Web services have eased the job of building systems to achieve a user's business objective by mapping required business activities to web services. With the
presence of automatically discoverable, network accessible, heterogeneous, and
dynamic web service, it is not the case anymore that the level of service is always fixed.
Service systems need extra control on QoS that enables them to recognize and react to
the environment. In [24], the problem of optimizing service system by application level
QoS management and focus on the system utility perspective of an application use
case, or a system end user in the layered architecture of the web service QoS stack is
discussed.

Little or no work has been done to provide a qualifyable end-to-end QoS to the
end user. In [27], authors have presented an optimization approach for the composition
of web services using dynamic service selection which allows specifying constraints on
quality requirements for the user both at local and global level, and to fulfill constraints
at runtime through adaptive reoptimization under variable QoS characteristics of web
services.

Certain research work implemented local approaches which select web services
one at the time by associating the running abstract activity to the best candidate service
which supports its execution. Local approaches can guarantee only local QoS
constraints, candidate web services are selected according to a desired characteristic,
example, the price of a single web service invocation is lower than a given threshold.
Certain research also proposed global approaches. The set of services that satisfy the
process constraints and user preferences for the whole application are identified before
executing the process. In this way, QoS constraints can predicate at a global level,
constraints posing restrictions over the whole composed service execution can be
introduced. In order to guarantee the fulfillment of global QoS constraints, second
generation optimization techniques consider the worst case execution scenario for the composed service. These above mentioned approaches could be very conservative and constitutes the main limitation of second generation techniques. Furthermore, global approaches introduce an increased complexity with respect to local solutions. The main issue for the fulfillment of global constraints is web service performance variability. Indeed, the QoS of a web service may evolve relatively frequently, either because of internal changes or because of workload fluctuations. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints web service selection and execution are interleaved: Optimization is performed when the business process is instantiated and its execution is started, and is iterated during the process execution performing reoptimization at runtime. Another drawback of such solutions is that, if the end-user introduces severe QoS constraints for the composed service execution, i.e., limited resources which set the problem close to unfeasibility conditions (e.g., limited budget or stringent execution time limit), no solutions can be identified and the composed service execution fails.

The goal of this thesis is to set the basis to overcome the limitations of the previous approaches to web services selection. The aim is to discover the web services of a flexible or reconfigurable process in such a way that the overall QoS perceived by the user is under QoS constraints. Severe constraints are very relevant whenever processes have to be performed with stringently limited resources. We introduce an
interface approach to the service selection problem identifying the optimal solution/services of the process.

To enable semantic description of service extensions, several ontological languages have been proposed. An overview of some of these languages can be found in [13]. They focus on the semantic modeling and mapping ontology applied to service descriptions. Our framework focuses on the engineering of ontology model and its references to the performance modeling tools. With the help of ontology mapping, different service description and advertisement standards should be easy to adapt in our framework.

Service composition methods and their languages can be broadly categorized into different types: Orchestration, Choreography, Coordination, and Assembly [14]. Composition methods use ontology to annotate QoS attributes that provide common ground for service synthesis, execution, and adaptation [15]. In our framework of QoS-aware service composition, services are selected based task (functional and non-functional) constraints. They can also be grouped into deterministic and non-deterministic depending on when these attributes were made known [14]. Various researches are hoping to gain optimal results by using detailed descriptions of QoS values of services during composition [16][17]. In [18], a quality-driven middleware serves as a composition manager that model multidimensional QoS attributes with utility functions, and optimizes them by local selection and global planning for different quality criteria.

In [19], requested and provided QoS properties are expressed as required specification documents and service specification documents respectively in the open
dynamic execution environment. The framework serves as a broker for service compositions that utilizes QoS model in its own ontological language. Service selection algorithms and metrics based on the ontology are utilized by the service broker. Its objective is to support ad-hoc service collaborations, while ours is to facilitate the description of QoS properties of existing and new composite services. The work is similar to ours with the emphasis on using ontology model as the tool to reason QoS attributes semantically just the same as ours. When monitoring the execution condition, the ontology model can facilitate the selection of correct set of QoS values according to the execution environment. The QoS-aware ontology modeling framework we propose can serve the same purpose.
3.1 WSDL

WSDL is the standard language suggested by World Wide Web Consortium (W3C) for service specification. It can be read as a conceptual model consisting of components with attached properties, which describes the service [3]. A WSDL specification contains abstract and concrete descriptions of the service. At abstract level, it describes the interface to the service: operations with message exchange patterns (MEP) and parameter types. At the concrete level, a binding specifies the transport type that the interface uses. An endpoint then associates a real network address with the binding, which forms the service. The service is invoked by supplying the declared signature to the interface through its endpoints.

Although the syntactic specifications provide information about the structure of input and output messages, and the functional descriptions of the service, WSDL does not address non-functional properties. To fully utilize web services, non-functional information, along with functionality, is needed in the service description. To augment any proposed extensions, backward compatibility and its extension level must be considered. Since WSDL description model addresses abstract and concrete components with services, the non-functional extensions to WSDL should be considered accordingly. It should be compatible with the original web services mechanism in that the addition may be considered optional. Web service engines and operations should be able to freely ignore the QoS information as they choose to operate in the conventional environment. For applications that adapt our framework, the
QoS-aware extensions are extracted easily. Extensions should be established at service level rather than at interface level, since the WSDL interfaces are bounded by the message exchange patterns and considered abstract models. At service level, an endpoint is where the abstract service binds to a concrete port type, where the overall service performance can be noted.

Since our motivation is not only the discovery of services meeting QoS requirements, but also composition of services leading to new services and ascertaining the QoS properties of composed services, we felt that available QoS extensions do not fully meet our needs. Hence we propose our own extensions to WSDL to specify QoS properties. To exemplify the utilization of these extensions, we propose a framework for service composition with the assistance of both ontological and performance modeling tools. QoS properties are modeled with the ontological engine that can be expanded in accordance with the service declaration. Properties that are subject to a chosen performance tool can also be noted in the ontology model for further semantic comparisons. In this thesis, we will focus on the performance aspect of the service; in particular service response time. As a proof of concept, we demonstrate the process of WSDL extension, along with its corresponding QoS ontology modeling, performance modeling, and service composition using an example.

3.2 SOA / BizTalk

Service-oriented architecture (SOA) is the current trend of application development, and web services are one of the key enabling technologies that use SOA. Web services, based on the widely used and industry-standard SOAP protocol, are
becoming increasingly ubiquitous. Exposing functionality as web services in such a
granular fashion is a great first step towards providing the basic building blocks required
to implement solutions, but more work is required. A management layer is needed—
something that can orchestrate the web services, controlling the flow and interaction
between them and aggregating individual services into a larger composite solution.
Microsoft BizTalk Server fills that role exceptionally well.

BizTalk Server fully supports all the open standards upon which web services are
built. With its core architecture based on XML and the .NET Framework, BizTalk Server
supports web services naturally. Web service support is tightly integrated into the
product.

A BizTalk solution can use web services in the following ways

1. Consume existing web services
2. Expose business processes (BizTalk orchestrations) as web services
3. Expose the messaging engine through web services (publishing schemas)
4. Invoke web services as part of pipeline processing, a custom adapter, or map
   execution

3.3 Consuming and Orchestration of Web Services

The use of web services is a key enabler of service-oriented architectures (SOA).
The tenets of SOA call for decomposing software solutions into a series of services that
are aggregated to form a higher-level service or a complete application. This approach
allows for greater reuse, because the services can be shared between applications and
can be combined in different ways to meet other business needs. However, developers
still need to write the middle layer that aggregates and choreographs the services, and
must also be concerned about compensating for unavailable services, handling returned faults, and so on.

A BizTalk orchestration (a graphical way to model and implement processes) simplifies this task by providing building blocks required to aggregate and manage web service invocations, thereby providing the higher abstract functionality required in a web service solution. Design patterns such as guaranteed delivery in an orchestration can be implemented which would allow us to take appropriate action if a web service is unavailable.

A BizTalk orchestration also provides a mechanism to group actions together in a transactional context, allowing appropriate compensation if the operations in a business transaction fail to complete.

At a high level, the typical steps required to consume a web service are:

1. Add a Web reference to the service that has to be consumed.
2. Add an orchestration.
3. Add a port to the orchestration, setting the port type to the Web port type that was created when the Web reference was added.
4. Create a new message of the same type as the web service request.
5. Create a new message of the same type as the web service response.
6. Send the request message.
7. Receive the response message.
8. Sign the assembly, deploy it, and test it.

3.3.1 Creating Web Services

Visual Studio .NET Integration Development Environment (IDE) can be used to create web services. By using the wizards in Visual Studio. NET we can create web
services. C# language has been used to create a web service. A web service file will have an .ASMX file extension. Object-oriented programming skills have been used to build the classes. This web service can be accessed through HTTP for any use. Once we have our web service ready, clients can use the services and extract the information from the XML documents that the web service produces. The clients could be a web page, console/windows applications, WML Script to interact with mobile phones, etc.

3.3.2 Adding Web References

BizTalk Server allows us to consume a web service by adding the web reference for the web service just the way it is done from any .NET Framework application that consumes a web service. When a Web reference to a BizTalk project is added, BizTalk Server queries the service's Web Services Description Language (WSDL) to determine what types, operations, and ports it exposes. The following artifacts are then automatically created and become available from the orchestration:

1. Web port type
2. Web message types (request and response)

In addition, the following items are added as part of the reference itself:

1. Reference map (contains links to the cached service and discovery file URIs)
2. WSDL (cached local copy of the web service description, method signatures, and types)
3. DISCO file (cached local copy of the web service’s discovery file, service and contract URIs)
4. ODX file (the BizTalk orchestration file that contains the actual types from the Web reference)
5. HTML file (a human-friendly version of the WSDL file, hidden in the IDE)
6. xsd file (optional, contains schemas for any complex types)

In the Microsoft Visual Studio Integrated Development Environment (IDE) these items appear under the Web reference, but in the project's folder in the file system a "Web References" folder has been added to the project. Each Web reference that is added to the project has a corresponding subfolder under the Web References folder. The odx file defines the types that are created after the WSDL is queried, and are subsequently available during the orchestration. If this file is double-clicked in Visual Studio, the file is not opened. Instead the corresponding HTML file will be seen. The HTML file gives a more refined presentation of the method signatures and data types expressed by the WSDL.

Entry and exit points from orchestrations are through ports, which are of a specific port type. The Web port type has a series of read-only properties that were set as part of the Add Web Reference operation. These properties identify the web service URI, methods, and parameters. In addition, a multipart message is created that reflects the request and, if appropriate, the response of the web service. This eliminates the need to manually create the schemas, and ensures conformity with the request and response types. A Web port can contain multiple operations, each of which represents a Web method call or response.

If the web service accepts complex types as parameters, or the return value is a complex type, then BizTalk Server needs a schema for those types. In this case, a xsd file is added to the Web References folder, and the schema file contains the definitions of the complex types as published by the WSDL file.

Before a Web port is added, Web reference has to be added to the BizTalk project. A Web reference is a description of a web service that is available to the
project. When a Web reference is added to project, BizTalk creates an orchestration Web port type, Web message types, Reference.map (map file), Reference.odx (orchestration file), <web service>.disco (discovery file), and <web service>.wsdl (Web Service Description Language file) to the project. If the Description Language (WSDL) file contains schema Web message types, BizTalk adds Reference.xsd to the project.

A Web reference includes:

1. A Universal Resource Locator (URL) for the web service.
2. A WSDL file that offers information about the service such as available methods, ports, and message types.
3. A reference map (Reference.map).

When a Web reference is added, all the Web methods for that web service must be compatible with BizTalk Server. Conditional attributes for specific Web methods in a web service cannot be specified.

3.3.3 Generating Basic Schema

BizTalk Server supports the following four types of schemas:

- XML schema: An XML schema defines the structure of a class of XML instance messages. Because this type of schema uses XML Schema definition (XSD) language to define the structure of an XML instance message, and this is the intended purpose of XSD, such schemas use XSD in a straightforward way.

- Flat file schema: A flat file schema defines the structure of a class of instance messages that use a flat file format, either delimited or positional or some combination thereof.
• **Envelope schema**: An envelope schema is a special type of XML schema. Envelope schemas are used to define the structure of XML envelopes, which are used to wrap one or more XML business documents into a single XML instance message.

• **Property schema**: A property schema is used with one of the two mechanisms that exist within BizTalk Server for what is known as property promotion. A property schema is a simple version of a BizTalk schema that plays a role in the process of copying promoted properties back and forth between the instance message and the message context.

### 3.3.4 Property Promotion on Schema

Promoted properties are message context properties that are flagged as promoted. Being promoted it allows the Message Engine to route messages based on their value, and being in the message context allows doing so without having to look at the message payload (which would be an expensive operation). They are available to the pipelines, adapters, message bus and orchestrations.

There are 2 ways to promote a message element:

1. **Quick promotion**: Quick promotion is the simplest way to create a promoted property. Visual Studio will create a property schema called PropertySchema.xsd and add in the message’s schema a reference to the generated property schema. Each property promoted this way will create a corresponding element in the property schema with the same name and type as defined in the message’s schema.

2. **Manual promotion**: To manually promote a property, a property schema must be created with the elements that will hold the promoted property values.
3.3.5 Schema Mapping

BizTalk Mapper can be used to define a transformation—a map—from one document to the other.

3.3.6 Constructing Web Messages

Web messages are the messages that are used when a web service is consumed (called). Web message is constructed from a Web message type. When a Web reference is added, BizTalk automatically creates Web message types, which BizTalk creates based on the Web methods from the added web service. Individual message parts can be created based on primitive .NET or schema types. Depending on the Web method parameter (input or output), BizTalk creates a Web message type from a primitive .NET type or a schema type. If the Web method parameter is a primitive .NET type, the message part uses a primitive .NET type. If the Web method parameter is a schema type, BizTalk adds the schema type to the BizTalk project as a schema in Reference.xsd. The schema is the basis for the message part. Reference.xsd can be found in the Web references folder. Alternatively, both primitive and schema .NET types can be created by calling a .NET class.

3.3.6.1 Constructing Web Service Requests and Responses

BizTalk Server is a message-oriented paradigm. This means that a web service can be called by constructing a request message and sending it to the Web port. If a Web method does not accept a parameter, still the same approach can be used, in that a multipart message that contains no parts is created.
3.3.6.2 Sending and Receiving

Sending a web service request and receiving a response is the same with web services as with any other port type. The messages in the send and receive need to be set to match the messages which were created for the request and response, and then we need to connect these shapes to the request and response operations of the Web port.

3.3.7 Orchestration

Once the schema and the references are ready, the main orchestration can be created. Select "Orchestration" from the "Add New Item..." once you right click on the project. Name the messages and change the “Message Type” property. Create messages accordingly for request and response.

Orchestration designer provides various tools and shapes that can be used to create the visual representation of underlying actions. They help us to efficiently design and implement orchestration. And also we can aggregate several web services to form a single orchestration to complete an entire business process.

Table 3.1 Tools available for orchestration in BizTalk

<table>
<thead>
<tr>
<th>SNO</th>
<th>Shape</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Receive shape</td>
<td>It helps us to receive a message from a port</td>
</tr>
<tr>
<td>2</td>
<td>Send shape</td>
<td>It helps us to send a message to a port</td>
</tr>
<tr>
<td>3</td>
<td>Construct Message</td>
<td>Helps us to construct a new message. We can specify the message variable and also make message assignments.</td>
</tr>
<tr>
<td>4</td>
<td>Decide</td>
<td>It helps us to conditionally branch or make a decision in the orchestration.</td>
</tr>
<tr>
<td>5</td>
<td>Delay</td>
<td>Delays can be built based on time for various actions that are inside the orchestration.</td>
</tr>
<tr>
<td>6</td>
<td>Expression</td>
<td>It helps us to use any expression in order to assign values to variables.</td>
</tr>
<tr>
<td>7</td>
<td>Message Assignment</td>
<td>Helps us to assign message value i.e. Assign values to one schema from another schema.</td>
</tr>
<tr>
<td>8</td>
<td>Loop</td>
<td>Helps us to loop the process or actions until a condition is met</td>
</tr>
<tr>
<td>9</td>
<td>Parallel Action</td>
<td>Helps us to perform two or more operations in parallel or simultaneously and also independent of each other</td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10</td>
<td>Port</td>
<td>Ports specify how the orchestrations sends messages and receives from the other business processes.</td>
</tr>
<tr>
<td>11</td>
<td>Start orchestration</td>
<td>It helps us to invoke another orchestration asynchronously.</td>
</tr>
<tr>
<td>12</td>
<td>Terminate</td>
<td>Helps us to immediately end the operation in the case of any occurrence of error.</td>
</tr>
<tr>
<td>13</td>
<td>Throw exception</td>
<td>Helps us to immediately throw an explicit exception in the case of any occurrence of error.</td>
</tr>
<tr>
<td>14</td>
<td>Transform</td>
<td>Helps us to transform data from one message to another.</td>
</tr>
</tbody>
</table>
4.1 QOS Aware WSDL

WSDL can only be used for specifying functionality of services. Non-functional properties, including several quality of service (QoS) characteristics, are crucial to the success and wider adoption of web services. Customers would like to use QoS characteristics of web services for selecting from among several alternate implementations. Each of the potential service providers declares similar functionalities for the same purpose – thus the customer expects more information about services. Typical among QoS properties are security, reliability, and performance [1]. WSDL should be extended in order to provide QoS related information with services. Once non-functional properties of services are specified, it will be possible to develop or extend tools for the discovery of web services based both on functionality and non-functional properties. Additional tools can be designed for service aggregation, integration and composition based on QoS characteristics.

As we proceed with the quality-aware extension to the specification of services, it will be necessary to define standard metrics for non-functional properties. Consider for example “response time” as a non-functional property, and consider the composition of two services with 3ms and 5ms response times. One cannot assume that the response time of the composed service is 8ms, since computation of service times are based on stochastic measures and it may become necessary to use appropriate models (e.g., queuing theory) for computing the response time of the composed service. The orchestration of services in a composed service plays an important role in modeling
QoS properties of the composite service. In the case of performance, additional complexity results from the current workload at a processing node: a lightly loaded node leads to faster response times. This may necessitate specification of performance properties at different levels of workloads (e.g., at low, average and heavy loads). These complexities can be managed using ontologies for the specification of non-functional properties.

WSDL2.0 Core standard provides element-based extensibility that can be used to specify technology-specific binding. We create an element in WSDL to represent QoS property specification. Then we use the element as the extension element to the endpoint. The service with the endpoint is therefore being annotated by the extended properties. For a QoS property extension element, we use complex type in the XML schema to accommodate the data structure of the QoS. As depicted in Figure 4.1, the QoS-aware extension schema exemplifies a non-functional property of performance. Within the performance criteria, response time is noted with its value, unit, and category. The extension can also be further referenced by importing latest XML schema version which can be updated on-the-fly as revising the QoS ontology model, thus conforming to the latest XML standards.

In [13], we have developed ontologies that can be used to select and compose web services. Ontology model provides more flexible organization and semantic interpretation of data with entities. The quality of service property of a Web service can be inferred by its performance attributes. Criterion of service selection can be formulated by the configuration of these attributes to indicate levels of service importance. Due to the dynamic nature provided by service oriented mechanism, even
the meaning of performance metrics should be adapted to fit the context of the service domain. To create an ontological model for Web services, leading to service composition, we have also demonstrated the process of establishing performance as non-functional property of Web services, such as response time, server utilization, and throughput. The model refers to the performance modeling used in the composition. In my thesis work, I will be referring to the same ontology model. The base ones serve as the mandatory performance attributes that all the Web services are required to specify as performance indicators. The model-related properties serve as the supplement to the application-specific modeling approaches, thus can store additional attributes for use by specific methods and tools. In our example, the base performance classification is represented by Quality-of-Services (QoS). The QoS subclasses ResponseTime, Throughput, and Utilization are base performance indicators, to quantify performance property. Model-related attributes include Workload and Statistics. Workload here is used as an attribute to evaluate the significance of base QoS properties. The attributes allow for recording the criteria under which the performance properties were derived and thus allow for adjustments when new running environments differ from these values.

For each of the base and model-related first-class elements, classification can also be refined into detailed subclasses. For instance, a response time can be ranked into subcategory such as Fast, Quick, Normal, Slow and Sluggish. Each of the rank can also be noted with its values that represent the class, based on specific context. As the new service composition emerges, the new service can easily be accommodated in the
ontological model, establish quantity and corresponding semantics, and is ready for further queries and reasoning.

```xml
<xsd:complexType name="criteriaService">
  <xsd:complexContent>
    <xsd:extension base="qwsdl:ExtensibleDocumented">
      <xsd:sequence>
        <xsd:element name="performance" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xs:NCName" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="performanceType">
  <xsd:sequence>
    <xsd:element name="ResponseTime" minOccurs="1" maxOccurs="unbounded" type="qwsdl:ResponseTimeType"/>
    <xsd:attribute name="Offered" type="boolean"/>
  </xsd:sequence>
  <xsd:complexType name="ResponseTimeType">
    <xsd:attribute name="value" type="float"/>
    <xsd:attribute name="unit" type="string" fixed="sec"/>
    <xsd:attribute name="category" type="qwsdl:CategoryType"/>
  </xsd:complexType>
</xsd:complexType>
```

Figure 4.1. QoS-Aware WSDL schema for performance parameters

4.2 Services/Orchestration

The purpose of our framework is to provide a platform that enables the use of appropriate tools for performance evaluation in web service composition. According to the approaches the process takes, developers can explore different tools that fit the nature of the composition. Appropriateness can also be explored by comparing various tools for their usability. To demonstrate the usability of the framework, we explain the use of a queuing model with services containing mandatory performance attributes.

I have developed a search tool that would help us to select services according to our criteria (both functional and non functional) using an Interface approach. This is a web service which can be hosted on any server and can be accessed through internet.
It would read all the WSDLs present in the repository and extract necessary functional and non-functional information using the tags that are used to define them within these WSDL files. This search engine would have the user interface to select the filters and search accordingly. Figure 4.2 is the screenshot of the web page of the tool.

![Service selection tool](image)

**Search Results**

<table>
<thead>
<tr>
<th>File Name</th>
<th>NonFunctionalProperty</th>
<th>NonFunctionalValue</th>
<th>NonFunctionalCategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>BizTalkProjectApplication_BizTalk_Operation1,Port_4.xml</td>
<td>ResponseTime</td>
<td>5</td>
<td>RT_Fast</td>
</tr>
<tr>
<td>BizTalkProjectApplication_BizTalk_Operation1,Port_4.xml</td>
<td>Utilization</td>
<td>80</td>
<td>UT_Thin</td>
</tr>
<tr>
<td>BizTalkProjectApplication_BizTalk_Operation1,Port_4.xml</td>
<td>Throughput</td>
<td>15</td>
<td>TH_High</td>
</tr>
<tr>
<td>MyService.xml</td>
<td>ResponseTime</td>
<td>5</td>
<td>RT_Fast</td>
</tr>
<tr>
<td>MyService.xml</td>
<td>Utilization</td>
<td>80</td>
<td>UT_Thin</td>
</tr>
<tr>
<td>MyService.xml</td>
<td>Throughput</td>
<td>15</td>
<td>TH_High</td>
</tr>
<tr>
<td>MyService1.xml</td>
<td>ResponseTime</td>
<td>5</td>
<td>RT_Fast</td>
</tr>
<tr>
<td>MyService1.xml</td>
<td>Utilization</td>
<td>80</td>
<td>UT_Thin</td>
</tr>
</tbody>
</table>

**Figure 4.2. Service selection tool**

While composing services, the flow among the component services can be described using workflow or business logic. Each of the services can be represented as service nodes, and the request flow can be modeled as waiting queues. In front of each service node, requests are waiting in a line for the service to process them in order. The composition model is formed with the integration of the coordinated services network. The performance outcome of the queuing network is the performance result of the newly composed service.
The orchestration of services in a composed service plays an important role in modeling QoS properties of the composite service. Appropriate models (e.g., queuing theory) are necessary for computing the QoS properties of the composed service. Our framework for service composition exemplifies the assistance of both ontological and performance modeling tools. QoS properties are modeled with the ontological engine that can be expanded in accordance with the service declaration. Properties that are subject to a chosen performance tool can also be noted in the ontology model for further semantic comparisons. I have implemented a demonstration of the process of WSDL extension, along with its integration with corresponding QoS ontology modeling, performance modeling, and service composition using an example.

Service composition decisions have to be made from considerations of both functional and non-functional requirements. To manage the semantics of both functional and non-functional aspects and facilitate the automatic selection of service components that meet the service level requirement, an ontology engine is proposed to efficiently and flexibly classify both functional and non-functional attributes [20]. The backend of the composition framework provides interfaces in utilizing ontology model and models for the evaluation of QoS properties for web service composition. In this thesis, I have implemented the interface between SOA application and Ontology model. The two modules are independent and any potentially compatible models and tools can be plugged in. The process of creating the ontological model, and use a queuing model for composition of performance related properties of services is clearly illustrated in [20]. Figure 4.3 is a screenshot which shows how the service composition is interacting with
Ontology model in order to derive the composed QoS properties of the service composition.

The simplest form of a web service composition involves two services, say WS a and WS b. The possible compositions of the two services can be sequential or parallel composition, say WS rs and WS rp. Borrowing the syntax from generic process algebra, the sequential composition can be represented as WS rs=WS a.WS b, and the parallel composition can be represented as WS rp=WS a||WS b. Assume WS_a and WS_b each represents an entity in different tasks/processes/services. Each task/process/service is assigned to run on its own processor on different hosts. The sequential and parallel composition examples are depicted in Figure 4.4.
In the case of similar services encountered same workload but running on different platforms, the selection process has to compare the performance indices such as response time or throughput.

The service composition in both sequential and parallel topology can be scaled by accommodating multiple services at once. Resources can be exclusively owned or shared among services. Service composition can be based on either serial, parallel or other composition of the services involved.

To export the ontological result that is acquired by the web service composition mechanism, we use service publishing interface for demonstration purposes. The interface also enables the abstraction that web service ontological engine (WSOE) and performance modeling engine provide. Figure 4.3 describes this process.

The WSOE provides for composition of services in the context of web services management. The utility of the composition services include basic service information maintenance and composition. Service management functions include insertion, update,
and deletion. WSOE Insertion creates a record in the performance ontology model with its name and associated performance properties. The performance properties in our running example are the response time of the service. Other non-functional or QoS properties can also be included within our framework. WSOE Update and WSOE deletion are used to update and remove the correspondent services. New service composition information created by the WSOE can be obtained by the WSOE Compose Seq or WSOE Compose Par. The former will take the list of web services in the order specified, and model them as a sequential network in the layered queuing model. The output will be the performance indexes for the composed service. For our simple example, the composition would return the predicted execution time. Likewise, WSOE Compose Par will take a list of web service in the argument, and model them as parallel network in the queuing model. The sequential and parallel compositions can be combined to obtain any general compositions of services.

\[
\text{RespTime\_Composition} = (\text{RespTime\_a} \mid \mid \text{RespTime\_b} \mid \mid \text{RespTime\_c}) \cdot \text{RespTime\_d}
\]

Figure 4.5. Services in orchestration - flow diagram
Components that will be used in our demonstration would include:

1. Loop orchestration (loops a service n number of times)
2. Decide orchestration (decides a service or selects a path based on a condition)
3. Series orchestration (orchestrates 2 services in execute in series)
4. Parallel orchestration (orchestrates 2 services to execute in parallel)
5. Individual service orchestration (orchestrates just one service and returns output)
6. API to read non functional properties from WSDL files

Each of these component orchestrations (Figure 4.6) will have entry point and exit point (entry port and exit port). They can be reused within a project in 2 ways.

1. Using the orchestration as it is in the flow by connecting the ports properly
2. By publishing this orchestration as a web service and using the web service reference within the orchestration.
Each of these components can be configured to receive inputs directly from a web service or physical xml file using schema. Schema will have to be mapped accordingly. An orchestration implementing a business process typically receives some documents (input) and sends others. Part of the information in the received documents is often transferred to the sent documents, perhaps transformed in some way. For example, an order fulfillment process might receive an order for some number of items, and then send back an acknowledgment indicating that the order was sent. Information from the order, such as the name and address of the purchaser, might be copied from fields in the received order to fields in the order acknowledgment. BizTalk Mapper can be used to define a transformation—a map—from one document to the other.

Within each of the components, each of the service’s non functional properties can be accessed from their respective WSDL files. The output (if exists, because may not exist in few scenarios, for example if the web service is void and does not return any output) will be constructed using the output schema which would also include composed non functional parameters for respective orchestrations. Schemas will be defined based on all the services that will be involved in the entire orchestration. Schema creation will have to consider the input messages that each service expects output messages that each service would send, messages related to Composition API, messages with respect to each orchestration, and messages with respect to any physical input or output files involved. All the properties /elements involved in the schema will have to be promoted before they can be used. Input and Output ports will have to be configured accordingly based on the flow. And the operations also will have to be configured accordingly either from web service operation or orchestration operation. Figure 4.7 describes
configuration of input message type to the workflow and Figure 4.8 describes configuration of schema of input message at each orchestration level.

Figure 4.7. Configuring input to the workflow.

Figure 4.8. Configuring receive inputs to the orchestration using schema.
In this thesis we demonstrate our framework with a simple example which involves 4 web services. Service 1, Service 2 and Service 3 offer same functionality but with different non functional properties (response time) [Figure 4.5]. Based on the user preference, one of the services is selected and then service 4 is executed in sequential with the selected service. So the total response time of the entire composition would be sum of response time of (Service 1 or Service 2 or Service 3) and Service 4. We have exported the ontological result that is acquired by the web service composition mechanism by using service publishing interface for demonstration purposes (ComposeSEQService - WSOE_Compose_Seq.asmx). The interface also enables the abstraction that web service ontological engine (WSOE).
CHAPTER 5
DISCUSSION AND CONCLUSION

In this thesis, I have described a framework for composing Web-services using both functional and QoS properties. Provision of quality of service support from web services is to solve the issues of processing methods; and of models, ontologies, and semantic languages. Service selection and composition are studied from the middleware perspective based on WS QoS. This thesis discusses about implementing runtime QoS collection, binding/ rebinding and transaction support. These works form the basis of working toward a general solution to optimizing service systems. This thesis also considered user satisfaction levels on web service usage and end-to-end or application QoS criteria satisfaction. It focuses on defining models of QoS characteristics and constraints of service attributes. I have first extended WSDL descriptions of Web-services so that non-functional or quality of service parameters can be associated with the service. I have also developed API for selecting Web-services based on both functional and non-functional properties. For the purpose of this thesis, I demonstrated how services can be composed either in series, parallel or any other flow, and how to use a queuing engine to derive the performance properties of the composed service. I have also described how to compose web services using QoS properties.
APPENDIX

GLOSSARY
HTML: Hyper Text Markup Language, HTML is a markup language for describing web pages.

HTTP: The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

IDE: An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

QOS: Quality of service. The parameters or properties which describe the quality or the non functional properties of the service.

SOA: Service oriented architecture is the Framework for software development in many application domains.

SOAP: SOAP protocol describes the communications protocols

UDDI: Universal Description, Definition and Integration (UDDI) is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet

URL: A uniform resource locator, abbreviated URL, also known as web address, is a specific character string that constitutes a reference to a resource.

WSDL: Web Service Description Language is the standard to describe web services

XML: XML stands for Extensible Markup Language. It is designed to transport and store data.
REFERENCES


