A SERVICE ORIENTED ARCHITECTURE USING WEB SERVICES

Sonny Anousone Phrommany

Follow this and additional works at: https://scholar.dsu.edu/theses
A SERVICE ORIENTED ARCHITECTURE USING WEB SERVICES

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Information Systems

March 30, 2006

By

Sonny Anousone Phommany

Project Committee:

Dr. William Figg
Dr. Omar F. El-Gayar
Dr. Ronghua Shan
We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Master of Science in Information Systems.

Project Committee

Faculty supervisor: ___________________________  Date: 8-16-2006

Committee member: ___________________________  Date: 8/16/06

Committee member: ___________________________  Date: __________
ACKNOWLEDGMENT

Systems analysis, design, and development demonstrated in this document are partially inspired by Wells Fargo & Company best practices and standards and fully inspired by industry standards (Microsoft, Java Community, Apache group, WSI, W3C, UML, OMG, etc) and Sonny Anousone Phrommany’s experiences in designing and developing systems for both the public and private sectors. XML Schema (types, elements, definitions, organization, etc) best practices and naming standards inspired by WFXML2.1 standards, but not necessarily in whole, thus, not exposing any Wells Fargo & Company private intellectual property.
ABSTRACT

This project was undertaken to provide a better understanding of what a Service Oriented Architecture (SOA) using Web Service is; the various artifacts involved; recommended tools needed in order to build to an SOA; and a demonstration of how a .NET system can easily interact with a Java system using Web Service technologies.

As a result of this project a fully functional ASP .NET client system will interact with a Java Customer Profile service. Two heterogeneous environments that once would have cost companies significant amount of time and dollar to integrate, now are easily integrated using Web Service technology.
DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

[Signature]  
Sonny Anousone Phrommany
# TABLE OF CONTENTS

ACKNOWLEDGMENT.................................................................................................................. III

ABSTRACT ..................................................................................................................................... IV

DECLARATION............................................................................................................................. V

TABLE OF CONTENTS .............................................................................................................. VI

LIST OF TABLES ....................................................................................................................... VIII

LIST OF FIGURES ..................................................................................................................... IX

INTRODUCTION ........................................................................................................................ 1

  STATEMENT OF PROBLEM ..................................................................................................... 1
  OBJECTIVES / DELIVERABLES ................................................................................................. 2

LITERATURE REVIEW ............................................................................................................. 4

  SERVICE ORIENTED ARCHITECTURE ................................................................................. 4
  WEB SERVICE ....................................................................................................................... 6
  XML ..................................................................................................................................... 8

SYSTEM DESIGN ..................................................................................................................... 18

  OVERVIEW .......................................................................................................................... 18
  USE CASE MODEL .............................................................................................................. 18
    1. Overview ....................................................................................................................... 18
    2. Use Case - Get Customer .Net Client ............................................................................ 19
    3. Use Case – Customer Profile Java Service ................................................................... 21
  LOGICAL MODEL ................................................................................................................. 22
    1. Overview ....................................................................................................................... 22
    2. Analysis Model .............................................................................................................. 24
    3. Design Model .............................................................................................................. 25
      Layered Architectural Approach ..................................................................................... 26
      Design Component Types ............................................................................................... 28
      Service Oriented Architecture ....................................................................................... 34
  PHYSICAL MODEL ............................................................................................................... 35
    1. Overview ....................................................................................................................... 35
    2. Web Service Artifacts ................................................................................................. 35
LIST OF TABLES

Table 1: Design, Layer, and Analysis Traceability Matrix for .NET Client .......... 30
Table 2: Design, Layer, and Analysis Traceability Matrix for Java Service .......... 33
Table 3: Web Service Artifacts ........................................................................... 36
Table 4: Node: Layer to Namespace: Assembly Traceability Matrix for .NET .... 41
Table 5: Node: Layer to Package: Jar Traceability Matrix for Java ................. 44
Table 6: Get Customer View Sequence Model Steps ........................................... 45
Table 7: Display Customer Information Sequence Model Steps ....................... 46
Table 8: Display SOAP Information Sequence Model Steps ........................... 47
Table 9: Display Fault Information Sequence Model Steps .............................. 48
Table 10: CD Folder Structure Description ....................................................... 61
LIST OF FIGURES

Figure 1: Entertainment System ................................................................. 6
Figure 2: SOA High Level................................................................. 7
Figure 3: SOA Roles................................................................. 7
Figure 4: XML ........................................................................ 9
Figure 5: XML Schema ................................................................. 13
Figure 6: XML that conforms to the XML Schema .................................... 13
Figure 7: Message Schema ............................................................. 14
Figure 8: WSDL ........................................................................ 16
Figure 9: SOAP Request ................................................................. 16
Figure 10: SOAP Response ............................................................... 17
Figure 11: Schema to HTTP Overview .............................................. 17
Figure 12: System Design Phases ....................................................... 18
Figure 13: Get Customer .Net Client - Use Case Diagram ..................... 19
Figure 14: Get Customer .Net Client – Activity Diagram ...................... 20
Figure 15: Customer Profile Java Service - Use Case Diagram .............. 21
Figure 16: Customer Profile Java Service – Activity Diagram ............... 22
Figure 17: Model View Controller ..................................................... 24
Figure 18: Get Customer .Net Client - Analysis Model ......................... 25
Figure 19: Customer Profile Java Service - Analysis Model .................. 25
Figure 20: Get Customer .Net Client – Layers ..................................... 26
Figure 21: Customer Profile Java Service – Layers ................................ 27
Figure 22: Get Customer .Net Client UI, BL, DA, and DS Layer - Design Model .... 29
Figure 23: Get Customer .Net Client FW and BE - Design Model ................... 30
Figure 24: Customer Profile Java Service I, BL, DA, and DS Layer - Design Model 32
Figure 25: Customer Profile Java Service FW and BE Layer - Design Model .......... 33
Figure 26: Customer Profile Java Service – Service Interface .................. 35
INTRODUCTION

This project will deliver a more in-depth explanation of Service Oriented Architecture (SOA) using Web Service as outlined in appendix A “High Level Outline.” From there, I will show examples and all the artifacts (XML, Schema, SOAP, WSDL, and service implementation code) that are needed in order to develop an SOA using Web Service at any particular company. The tools that I will use to implement these services will be Enterprise Architect, XML Spy, Visual Studio .Net, IIS, Eclipse, and Tomcat. After going over the artifacts, tools and implementation procedures, I will give a demo showing a Get Customer Information System consuming a Customer Profile Service System. This will demonstrate how SOA using Web Service provides interoperability between heterogeneous systems, i.e., .Net and Java.

Statement of problem

Enterprises, such as Wells Fargo, operate in a large number of custom built applications. All of the applications systems use a wide variety of APIs and integration approaches. Moreover, the systems are deployed on a wide variety of platforms (UNIX, Windows, Linux, AIX, etc.) and use a wide variety of languages (Java, C++, C#, COBOL, etc) and technologies. All of these monolithic systems are tightly coupled, i.e., not easily allowing for integration or reusability of functionalities throughout the enterprise without a significant cost. That is where SOA using Web Services comes into play. It provides a standard way to integrate applications independent of hardware, operating systems, and development language. This project will expand on the Web Service focus area to SOA and demonstrate its usage at an enterprise level.
Objectives / Deliverables

For this project I will act as a consultant providing SOA information to management (the audience) and demo a .Net client application retrieving customer information from a Java Customer Profile Web Service. The main goal of this project is to demo the application to show interoperability between heterogeneous systems using SOA with Web Service.

After completion of the project the following objectives deliverables will be met:

- A power point explaining the different artifacts that make up an SOA using Web Service as outlined in the “High Level Outline,” see appendix A.

- A .NET Web Page with a text box to enter customer id and or social security number and a button that will invoke the Java Customer Profile Web service and display customer information on the page. The customer information displayed will be Name Prefix, First Name, Middle Name, Last Name, Name Suffix, Social Security Number, Number of Dependents, Marital Status, Home residence effective date, Birth Date, Employer Name, Employment Start Date, Job Title, Address, City, State, Zip, Phone Number. A design document for the client application will also be provided to show how the code was implemented to invoke the service. The design document will include Use Case(s), Mock-up screens, Sequence Diagrams, Logical Diagrams, and Class Diagrams. In addition, all the .NET Code will be provided.

- A Java Customer Profile Web Service that will retrieve customer data from a MySQL database and send back to client applications. XML Schemas will be created to define my Customer Profile data model and request response payload. A WSDL document will also be created to describe my Web Service. A design document for the Customer Profile
service will also be provided. The design document will contain Use Cases, Class
Diagrams, Sequence Diagrams, and Logical Diagrams. In addition, the Java code to
create the service will also be provided.

- A MySQL database will be implemented to contain customer information. A design
document with a logical data model and physical model will be included. Scripts used to
populate or retrieve data will also be provided.
LITERATURE REVIEW

Service Oriented Architecture

What is Service-Oriented Architecture (SOA)? SOA defines how two computing entities interact in such a way as to enable one entity to perform a unit of work on behalf of another entity. The unit of work being done is called a service. A service is a self-contained piece of software, with a defined role, that does a specific business function that can be used or “consumed” by many applications regardless of the calling application’s state or platform (Tech Target Online). Another definition, an SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. Thus, the use of services allow for loose coupling of software systems (He, 2003). Loosely coupled as defined by John Hagel (2002), states that, “loosely coupled is an attribute of software systems, referring to an approach to designing interfaces across modules to reduce the interdependencies across modules or components – in particular, reducing the risk that changes within one module will create unanticipated changes within other modules. This approach specifically seeks to increase flexibility in adding modules, replacing modules and changing operations within individual modules.” Most systems today are not loosely coupled. They were built “tightly coupled” and therefore integration and reuse of functionalities of the software is harder to implement and comes at a greater cost to the business.

An analogy of a SOA would be a home theater system, Figure 1. In a home theater system, you have various components, such as, a DVD player, VCR player, receiver, CD player, television, etc. These are all self contained pieces of hardware. That is, they don’t
need each other to function. A CD player can play CDs without the help of a VCR and vice versa. Furthermore, a television does not have to be on for a VCR to able to record a show. Each one of these components provides separate distinct services. These independent home theater components are similar to a service in a SOA. They are loosely coupled! An example of a real world SOA service would be a Customer Profile service. Its defined role and or its functionality would be to retrieve customer data from a database or a system of record (SOR) of some sort and return data to the calling/consuming client application. Its distinct service is to provide customer data. This is analogous to a CD player providing CD playing service. Further, in a home theater system, the receiver and the speakers do not have to be physically located in the same room. The components, whether close or far, are connected via cables of some sort. This is the same for the service and the application in a SOA. They usually do not physically sit on the same server but connected via a Web service using HTTP and SOAP. This allows them to be shared and reused across an enterprise.
**Web Service**

A Web service is a new technology that you can use to create platform-independent applications. You can develop a Web service by using languages and platforms that adhere to a standard set of technologies (Arora & Kishore, 2002). The connection in a SOA is typically done using a Web service. Also if the service executes business logic then it is known as a business service, i.e., a Web service that performs business logic. This is the current, highly complimentary approach to SOA due to adhering to standards. This is similar to having the option of connecting a VCR to a television using a RCA, coaxial, s-video, or optical cable, etc. Some home theater components even have proprietary ports for proprietary connections. They usually do not last in the industry due to lack of standards. Other approaches to SOA would be CORBA, DCOM, and EJB to name a few. All have failed to a certain extent due to
lack of open standards, loose coupling, and or having proprietary data formats. A web service on the other hand uses XML, SOAP, WSDL, UDDI, HTTP (all governed by open standards) and a programming language of choice to implement, which makes a web service the choice for a SOA. Figure 2 shows a simple SOA. The service request response would be the connection/web service using HTTP/SOAP. Note that a service provider can also play the role of a service consumer and vice versa.

![Figure 2: SOA High Level](image)

SOA strategy is structured around four key roles, Figure 3: producers, providers, consumers, and management of services. The producers’ responsibility is to design and build the services and to offer and manage services. The consumers, on the other hand, will use the services and assemble them to meet the needs of the business. Finally, management oversees the overall process, ensuring the correct services are built and available, and are being appropriately used to meet the business needs.

![Figure 3: SOA Roles](image)
Why is SOA important to an enterprise? An enterprise landscape usually operates in a large number of custom built applications. All of the application systems use a wide variety of APIs and integration approaches. Moreover, the systems are deployed on a wide variety of platforms (UNIX, Windows, Linux, AIX, etc.) and use a wide variety of languages (Java, C++, C#, COBOL, etc) and technologies. All of these monolithic applications are tightly coupled, i.e., not easily allowing for integration or reusability of functionalities throughout the enterprise without a significant cost to the business. That is, where SOA with web services come in to play. It provides a standard way to integrate applications independent of hardware, operating systems, and development language. These modular services when “glued” together by an application becomes a workflow for a particular business process. A module used in one application can be re-used in another, thus, saving time and development efforts for the business.

XML

As mentioned before a Web service is the complimentary approach to SOA. It is created using technologies such as XML Schema, WSDL, SOAP, and HTTP. Out of the five components, its key player would be XML. So what exactly is XML? From Arora and Kishore (2002), “the Extensible Markup Language (XML) is a new markup language that is used to store data in a structured format. A markup language provides you with a set of tags that you can use to describe and format the text in a document.” Furthermore, it is a tag based language much like HTML. HTML was invented for the purpose of describing what a page should look like on a browser. Whereas, XML was invented to describe data that is being looked at independent of how it is displayed. One of the reasons for failures with other approaches to SOA, such as DCOM, was due to not passing messages using XML but rather a
binary and or often proprietary format. XML was created to get rid of such data ambiguity.

In Figure 4, looking at the two ways of describing data, in the “Other Format” version, it is hard to make out what the data stands for. In the “XML” version, even a person without XML knowledge would be able to make out what the document is describing. It is self-describing.

Other Format:
2021978

XML:
<?xml version="1.0"?>
<SonnyPhrommany>
<DateOfBirth>
<Month>2</Month>
<Day>20</Day>
<Year>1978</Year>
</DateOfBirth>
</SonnyPhrommany>

Figure 4: XML

So what makes up an XML document? An XML document is composed of the markup and the content. The markup is represented by the “<” left bracket. For example, <SonnyPhrommany> would be a markup tag. In a XML, there are six types of markups: element, attribute, comments, processing instructions, entity references, and CDATA sections. First, element tags make up a majority of the XML document. In Figure 4, <DateOfBirth>, <Day>, <Year>, <Month>, and <SonnyPhrommany> are all elements. Element tag begins with “<” and ends with “>” and must have a start tag like <Day> and end tag </Day>. The names of element tags are case sensitive and must start with a letter or an underscore. Elements may contain, text, other elements or be empty. If the element is empty, then it can
be defined like `<Day/>` or `<Day></Day>`. Second component is the attribute tag. This allows specification of key value pair for a particular start tag element. This gives a more refined description of the element. For example, `<SonnyPhrommanny height="69" weight="182"/>`, height="69" and weight="182" are attributes. Third, comments are ways of allowing a more general description of the data and will be ignored by the XML parser. Comments look like this, `<!-- Figure 4 is an XML Document to describe Sonny Phrommanny's Date of Birth -->`. Fourth, processing instructions start with “&lt;?” and end with “?&gt;”. They are information or instructions for the application and ignored by the XML parser. Fifth, entity references enable the use of otherwise reserved characters in the content. For example, the “&lt;” is reserved in XML. In order to use it in the content, say for a less than symbol, it has to be specified like such “&lt;”, `<Formula>x &lt; y</Formula>`, and not like, `<Formula>x < y</Formula>`. The latter will error out. Last is the CDATA section. Its purpose is to escape blocks of characters that otherwise might be interpreted as markup. For example, `<![CDATA[<HTML><BODY>This is my HTML page.</BODY></HTML>]]>`, everything between the “&lt;!” and “!” will be ignored by the XML parser and will be taken literally. If all the rules above are applied to a particular XML document, then the XML would be considered a well-formed XML document. But a well-formed document does not make it a valid XML document. A valid XML document means that the document is valid against an XML Schema or a DTD. In which will be elaborated in further sections.

Another advantage of using XML is that it is platform and language independent, and machine and human readable. Furthermore, XML is a standard! Someone half way around the world using some machine x will be able to process and read the XML data as long as they know the specification. The World Wide Web Consortium (W³C) is the primary
standards organization to uphold the XML specification. Its latest recommendation for XML is at version 1.1, can be found at http://www.w3.org/TR/2004/REC-xml11-20040204/.

The XML document in figure 4 is a very simple well-formed XML. So what if you have a more complicated XML document with multiple elements that are named the same? For example, two elements generically named <name> in one XML document. And in the XML, one element has a value of a first and middle name, <name>Sonny Anousone</name>; while the other has just the first name, <name>Sonny</name>. How will the receiving application resolve and know the meaning of each <name> element and how will it know how to use within a particular context? This is where namespaces and XML schema comes into play. Namespaces are ways to qualify each element in an XML document to make them unique and are defined within an XML Schema. An XML Schema is an XML document that describes and defines rules that are valid for a particular XML instance.

From W3Schools.com, an XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes (Refsnes Data)

Figure 5 is a simple XML Schema that defines the XML in figure 6. The XML, in figure 6, is considered well-formed, and valid in regards to the XML Schema that defined it. This is not true for the XML in figure 4, which is only well-formed. It's not valid because it doesn't reference an XML Schema. So essentially there is nothing to be valid against. Figure 5, line 2, defines the root element of the schema. All schemas are required to have a root
element named schema. The root element of `<xs:schema>` is namespace qualified to
`xmlns:xs='http://www.w3.org/2001/XMLSchema'`. This is known due to the prefix of
`xs` in the element which is mapped to the `xmlns:xs` in the namespace. This particular
namespace allows the use of the various predefined elements and data types by the W3C.
These data types defined by the W3C allow for easier mapping to database and programming
language primitive types, but not specific to any one technology type. It further reduces
ambiguity by defining data types for an element in an XML document. The sender and
receiver of the XML will have a clearer understanding of the data being exchanged. The
target namespace of “http://www.sonnyp.com/entity/sonnyphrommany/dateofbirth” is
also defined for this schema. The URL is a unique string. This means that an XML
document will be namespace qualified as shown in figure 6, line 3; thus, making the XML
instance unique. This resolves possible element collision within the XML. In the schema
document, lines 4-6, are simple types. They are simple types because they do not contain
child elements within and contain no attributes. These simple types are all of data type
`xs:string`. Possible data types are “xs:short”, “xs:date”, “xs:float”, etc. Remember, they
are defined by the W3C. Lines 7-13, define a complex type of “DateOfBirth_Type” that
contains child elements of simple types: month, day, and year. The naming convention of
“_Type” is used. The `<xs:sequence>` tag denotes that the simple types defined has to be in
that particular order in an XML instance or else it is not valid. Line 17 defines a global
element that has a name of “dateOfBirth” which is of the type “DateOfBirth_Type”.

An important characteristic of an XML Schema is that it allows for importing of other schemas. In other words, it is extensible. You can essentially bring in data that was defined in another schema into your own schema. This ability will help define a common reusable business vocabulary to define all your business data for a particular company. For large corporations, this provides a common vocabulary when exchanging messages between disparate systems. For example, a line of business (LOB) x within a corporation should have the same meaning of what a date of birth is from LOB y. This date of birth construct should be defined in one place and reused across the enterprise. In other words, a particular company

Figure 5: XML Schema

Figure 6: XML that conforms to the XML Schema
should have the same meaning of date of birth in all their LOBs data vocabulary. In rare cases, date of birth might be different in one LOB vs. another LOB. That is where namespaces will help define the context for how the date of birth is used.

In practice, a LOB can define its own schema(s) and inherit / import the data type from a “common” enterprise schema repository. The LOB is not restricted to using the types defined just from the “common” repository. If the data type is truly different in meaning than what is defined in the “common” enterprise repository, then it should be defined in its own LOB schema with a unique namespace to qualify it.

The messages in a SOA using Web services are typically defined by a message schema (Figure 7). Message schemas will define how the request and response message payload will look like. In other words, it defines input output parameter for a particular operation for the, e.g., GetCustomer and GetCustomerResponse. The message schemas usually import types from common schemas as shown in Figure 7, line 3; importing the common reusable schema from Figure 5. In practice, message schemas will import from industry, enterprise, and LOB schemas.

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xs:element name="id" type="xs:string"/>
  <xs:complexType name="GetDateOfBirthRequest_Type">
    <xs:sequence>
      <xs:element ref="id"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="GetDateOfBirthResponse_Type">
    <xs:sequence>
      <xs:element ref="id:SonnyPharmacy"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

*Figure 7: Message Schema*
A Web service is described by a web service document language, also known as a WSDL. This standardized XML based document details out what the service will provide. It should be all that is needed in order to use the Web service. A WSDL document is broken into types, messages, port types, bindings, and service sections, see Figure 8. The first section is types. This section will define the messages and data types being used in the Web service. This is usually described in an external message XML schema document and imported into the WSDL. The next section is messages. This will reference the request response messages in the XML schema. In other words, it defines the required input output parameters for a particular operation. Port types describe the operations that are available for the Web service and its related messages. Bindings define the message format and communication method for the Web service. The message format is SOAP and the communication method is HTTP. Lastly, the service section defines the name of the service and its location.
SOAP is a standardized XML based document used to define transmitting messages. Its parts consist of a soap header (optional), soap body, and an envelope to wrap both elements. In figure 9 and 10, only a soap body is shown. If a soap header were to be there it would sit at the same level as the soap body. The soap body wraps the request message and response message as defined in the XML schema.

Figure 8: WSDL

Figure 9: SOAP Request
Figure 10: SOAP Response

Figure 11, shows a high overview of how it is all tied together. It is split into two categories, Wire stack and Description Stack. The Wire Stack shows what gets transmitted and protocol used on the network. The Description Stack shows what defines the Web service. Generally, LOB schemas import from enterprise schemas, which will, in turn, import industry defined schemas. The message schemas which are usually LOB specific; will use all three types of schemas to define its messages. From there, the message schemas are used in the WSDL document to define the service and the use of SOAP to bind the message in that particular format in transition, in our case, using HTTP. HTTP is a transmission protocol used to transfer information via the Internet.

Figure 11: Schema to HTTP Overview
SYSTEM DESIGN

Overview

The MSIS demo system is composed of two systems, Get Customer .NET client and Customer Profile Java service. The systems were designed in phases as depicted in Figure 12. Each phase within the system design feeds into the other phase(s) and eventually to construction of the systems. The different phases will be elaborated in their perspective sections.

![System Design Phases Diagram]

*Figure 12: System Design Phases*

Use Case Model

1. Overview

The Use Case Model will show the MSIS SOA demo requirements or “objectives” at a high level from an architectural perspective. It is not meant to cover all possible scenarios in detail.
2. Use Case - Get Customer .Net Client

![Get Customer .Net Client - Use Case Diagram](image)

*Figure 13: Get Customer .Net Client - Use Case Diagram*

### 2.1 Description

The Customer Service Rep enters in customer Social Security number and or Enterprise Id number into the System by a user interface. The System then retrieves information from the appropriate Business Service and displays customer information to the Customer Service Rep. Figure 14 depicts the high level activity diagram for this use case.

### 2.2 Actors

- Customer Service Rep
- Business Service (Supporting Actor)
2.3 Main Flow of Events

2.3.1 Customer Service Rep receives a faxed application containing the customer’s Social Security number and or Enterprise Id number.

2.3.2 Customer Service Reps enters in Social Security number or Enterprise Id number into the System.

2.4 Alternative Flows

2.4.1 In 2.3.2, the System can not find customer information, display fault message.

2.5 Pre-Conditions

- Faxed application with Social Security number and or Enterprise Id number

2.6 Post-Conditions

- The System displays the Customer information
3. Use Case – Customer Profile Java Service

![Diagram of Customer Profile Java Service - Use Case Diagram]

*Figure 15: Customer Profile Java Service - Use Case Diagram*

### 3.1 Description

The Client sends the System an agreed upon SOAP request message. The agreed upon message is usually defined in a Web Service Description Language (WSDL). The System then retrieves the customer information based on the request message and sends a response message back to the Client. Figure 16 depicts the high level activity diagram for this use case.

### 3.2 Actors

- Client System
3.3 Main Flow of Events
3.3.1 Client sends the System a SOAP request message.

3.4 Alternative Flows
3.4.1 In 3.3.1, the Client sends System wrong SOAP request message. System sends fault message back to Client.

3.5 Pre-Conditions
3.6 Post-Conditions

The System sends a SOAP response message back to Client.

Logical Model
1. Overview

The Logical Model will consist of two views: Analysis Model and Design Model.
The Analysis Model is the first step in identifying the Design Model classes, which is then one step closer to identifying the Physical Implementation Model which will reflect what is being built.

The Analysis Model will use the Rational Unified Process' (RUP) approach that identifies three types of classes for each Use Case: **Boundary**, **Controller**, and **Entity** (AKA, Model-View-Controller) – See Figure 17.

A boundary class represents the interface between an actor and the system. The interface can serve to interact with a user or an external system. A user interface will allow actors to view, update, delete, add, or edit system data. In addition, it will be responsible for data validation. An external interface will allow the system to interact with other external systems and will have responsibilities of handling communication with external system, formatting the input parameters to the external system, and validating the output from the external system.

A controller class represents the bulk of the business and non-business functionality. It is responsible for moving data from the user boundary class to the entity class(s) and vice versa. It is also responsible for manipulating data, e.g., formatting, serializing, and or mapping / adapting data. Further, it is responsible for storing data and interacting with external boundary or controller classes.

An entity class is responsible for holding data that can be moved within the various parts in the system. It is also responsible for validating its own data.

The Design Model will expand on the Analysis Model and incorporate any previous experience best practices to the models. Models no longer fall into just a broad category of boundary, control, and entity classes for a particular use case. And classes may and may not
expand multiple use cases. Further, both functional and non-functional requirements will be modeled, defining a more granular model for the classes during this phase.

![Model-View-Controller](image)

*Figure 17: Model View Controller*

2. Analysis Model

Figure 18 shows the analysis model classes and their responsibilities as stated in the overview (boundary, control, entity) that were realized from Get Customer .Net Client Use Case, in the Use Case Model, section 2. Figure 19 classes are realized from Customer Profile Java Service Use Case in previous section.
3. Design Model

The design models are derived from the analysis models providing a more granular view of the various components.
Layered Architectural Approach

The MSIS SOA demo systems use the *Layers*, in combination with, the *Pipes and Filter* architectural pattern. The latter pattern allows components to be grouped in different layers according to their responsibilities. For example, interface, business logic, data access, and data store layer. The former pattern allows data to be process in various pipes or layers, filtered and or manipulated, and passed to the next layer for processing. This is depicted in Figure 20 and Figure 21 for the .Net and Java systems respectively, showing the flow through the various layers. Note: The Framework and Business Entity layer will be used throughout the various layers.

*Figure 20: Get Customer .Net Client – Layers*
Figure 21: Customer Profile Java Service – Layers

The design model types can be mapped to the analysis model classes and will be explained in the following paragraphs.

The interface layer is responsible for providing an appropriate interface for the actor to the system. It maps to the boundary class in the analysis model. In the case of the .NET system the user interacts with a graphical user interface, for the Java system it has to adhere to the service interface.

The business logic layer is responsible for most of the work for the system, for example, applying business rules and logic, manipulating data, and passing data back and forth throughout the various layers. It maps to the control and entity classes in the analysis model.

The data access layer is responsible for connecting to various data sources and adhering to its various types and formats and reading and writing to the data source. It maps to the boundary and control classes in the analysis model. For the .NET system, the data access layer connects to another system, therefore, it falls somewhere between boundary and
control classes. On the other hand, the data access layer for the Java system is a control class, since it connects to the customer profile data store, which is considered part of the system.

The framework layer is responsible for fulfilling the functional and non-functional requirements for the system. Non-functional requirements are requirements that are not specified by the business requirements or use cases, but are needed for the system to function. They act as support components for the functional requirements in the other layers. For example, any utilities, existing framework code, system exception handling, security code, etc., will fall within this layer.

The data store layer is responsible for storing business and system data. In this case, the customer profile data.

**Design Component Types**

The design component types are decomposition of the analysis types, thus, providing more types in comparison to the analysis types. Each design component type should have a clear definition of its responsibilities and purpose. From the design component types the physical implementation objects are then created.

Figure 22 and 23 shows the design types and where it exists within the different layers for the .Net System. And Figure 24 and 25 shows the design types and where it exists within the different layers for the Java Service.

Table 1 is a traceability matrix mapping design types to analysis types; the layer that it exists in; and its responsibilities for the .Net System. And Table 2 is a traceability matrix mapping design types to analysis types; the layer that it exists in; and its responsibilities for the Java Service.
Figure 22: Get Customer .Net Client UI, BL, DA, and DS Layer - Design Model
Figure 23: Get Customer .Net Client FW and BE - Design Model

Table 1: Design, Layer, and Analysis Traceability Matrix for .NET Client

<table>
<thead>
<tr>
<th>Design Types</th>
<th>Responsibilities</th>
<th>Layer</th>
<th>Analysis Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Profile Form</td>
<td>Responsible for allowing users to enter customer lookup data.</td>
<td>User Interface</td>
<td>Get Customer UI</td>
</tr>
<tr>
<td>Customer Profile Controller</td>
<td>Responsible for business logic and controlling flow for the form.</td>
<td>Business Logic</td>
<td>Get Customer Control</td>
</tr>
<tr>
<td>Customer Profile Adapter</td>
<td>Responsible for adapting user input data to models, .Net Generated Schema types to Models and vice versa.</td>
<td>Business Logic</td>
<td>Get Customer Control</td>
</tr>
<tr>
<td>Display Customer Info Form</td>
<td>Responsible for displaying Customer Info to the end user.</td>
<td>User Interface</td>
<td>Display UI</td>
</tr>
<tr>
<td>Display Customer Controller</td>
<td>Responsible for business logic and controlling flow for the form.</td>
<td>Business Logic</td>
<td>Display Control</td>
</tr>
<tr>
<td>Display Customer Adapter</td>
<td>Responsible for adapting Models and vice versa.</td>
<td>Business Logic</td>
<td>Display Control</td>
</tr>
<tr>
<td>Fault Info Form</td>
<td>Responsible for displaying fault information to user.</td>
<td>User Interface</td>
<td>Display UI</td>
</tr>
<tr>
<td>Fault Info Controller</td>
<td>Responsible for controlling flow for its form.</td>
<td>Business Logic</td>
<td>Display Control</td>
</tr>
<tr>
<td>Fault Info Adapter</td>
<td>Responsible for adapting Models to UI.</td>
<td>Business Logic</td>
<td>Display Control</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>Location</td>
<td>Integration</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>SOAP Info Form</td>
<td>Responsible for displaying SOAP data to user.</td>
<td>User Interface</td>
<td>NA</td>
</tr>
<tr>
<td>SOAP Info Controller</td>
<td>Responsible for controlling flow for its form.</td>
<td>Business Logic</td>
<td>NA</td>
</tr>
<tr>
<td>Menu Form</td>
<td>Responsible for overall navigation for site.</td>
<td>User Interface</td>
<td>NA</td>
</tr>
<tr>
<td>Customer Profile</td>
<td>Responsible for managing the access of the web service.</td>
<td>Data Access</td>
<td>Business Service Interface</td>
</tr>
<tr>
<td>Helper</td>
<td>A utility type that can be used throughout the different types.</td>
<td>Framework</td>
<td>NA</td>
</tr>
<tr>
<td>Session Manager</td>
<td>Responsible for maintaining state for the system. Will allow passing of objects within the various types.</td>
<td>Framework</td>
<td>NA</td>
</tr>
<tr>
<td>SOAP Logger</td>
<td>Responsible for the capturing of SOAP data that gets passed across the wire.</td>
<td>Framework</td>
<td>NA</td>
</tr>
<tr>
<td>Request</td>
<td>A data container to define a Request structure.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Response</td>
<td>A data container to define a Response structure.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Fault</td>
<td>A data container to define a Fault structure inherits from System.Exception.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Customer Name</td>
<td>A data container to define a Customer structure.</td>
<td>Business Entity</td>
<td>Customer Data</td>
</tr>
<tr>
<td>Customer Phone Number</td>
<td>A data container to define a Phone Number structure.</td>
<td>Business Entity</td>
<td>Customer Data</td>
</tr>
<tr>
<td>Customer Employment</td>
<td>A data container to define an Employment structure.</td>
<td>Business Entity</td>
<td>Customer Data</td>
</tr>
<tr>
<td>Customer Address</td>
<td>A data container to define an Address structure.</td>
<td>Business Entity</td>
<td>Customer Data</td>
</tr>
<tr>
<td>.Net Web Reference</td>
<td>Responsible for converting WSDL to generated .NET Types. It’s also the SOAP Engine responsible for the serialization and deserialization of types to the wire and vice versa.</td>
<td>Data Access</td>
<td>Business Service Interface</td>
</tr>
</tbody>
</table>
Figure 24: Customer Profile Java Service I, BL, DA, and DS Layer - Design Model
Figure 25: Customer Profile Java Service FW and BE Layer - Design Model

Table 2: Design, Layer, and Analysis Traceability Matrix for Java Service

<table>
<thead>
<tr>
<th>Design Types</th>
<th>Responsibilities</th>
<th>Layer</th>
<th>Analysis Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Axis</td>
<td>Responsible for converting WSDL to generated Java Types. It’s also the SOAP</td>
<td>Interface</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Engine responsible for the serialization and deserialization of the types to the wire and vice versa.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated Java Types From Schemas</td>
<td>Java package generated from WSDL defining Java types from schema types.</td>
<td>Interface</td>
<td>Customer Profile Service Interface</td>
</tr>
<tr>
<td>Customer Profile Controller</td>
<td>Controls the flow for the service.</td>
<td>Business Logic</td>
<td>Customer Profile Service Control</td>
</tr>
<tr>
<td>Customer Profile Adapter</td>
<td>Adapts data to model and model to schema types and vice versa.</td>
<td>Business Logic</td>
<td>Customer Profile Service Control</td>
</tr>
<tr>
<td>Request</td>
<td>A data container to define a Request structure.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Response</td>
<td>A data container to define a Response structure.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Fault</td>
<td>A data container to define a Fault structure.</td>
<td>Business Entity</td>
<td>NA</td>
</tr>
<tr>
<td>Customer</td>
<td>A data container to define a Customer structure.</td>
<td>Business Entity</td>
<td>Customer Profile Service Data</td>
</tr>
<tr>
<td>Name</td>
<td>A data container to hold customer data.</td>
<td>Business Entity</td>
<td>Customer Profile Service Data</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Component Type</td>
<td>Interface Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Phone</td>
<td>A data container to define a Phone structure.</td>
<td>Business Entity</td>
<td>Customer Profile Service Data</td>
</tr>
<tr>
<td>Employment</td>
<td>A data container to define an Employment structure.</td>
<td>Business Entity</td>
<td>Customer Profile Service Data</td>
</tr>
<tr>
<td>Address</td>
<td>A data container to define an Address structure.</td>
<td>Business Entity</td>
<td>Customer Profile Service Data</td>
</tr>
<tr>
<td>Customer</td>
<td>A data access object for the Customer Model.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>Name</td>
<td>A data access object for the Name Model.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>Phone</td>
<td>A data access object for the Phone Model.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>Employment</td>
<td>A data access object for the Employment Model.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>Address</td>
<td>A data access object for the Address Model.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>DAO</td>
<td>Data access object’s super class.</td>
<td>Data Access</td>
<td>Customer Profile Service Data Interface</td>
</tr>
<tr>
<td>Hibernate Package</td>
<td>An Object Relational Mapping tool to map objects to database tables.</td>
<td>Data Access</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Service Oriented Architecture**

In order to provide a Service Oriented Architecture, the MSIS demo Java Customer Profile service system exposes various interfaces for its clients and a recommended service component name, see figure 11. The only interface implemented for demonstration is the GetCustomer interface that is part of the CustomerProfileService component. The SearchCustomer and UpdateCustomer are possible future interfaces to add to the service.
Physical Model

1. Overview

The Physical Model will detail out standards (names, etc), guidelines, including physical component grouping (namespaces, packages) as well as Web Service Artifacts, Database Models, Deployment Models, and Implementation Models (actual component types, sequence diagrams). The physical model should reflect what is to be built and deployed to production.

2. Web Service Artifacts

Table 3 provides a list of the various web service artifacts names, responsibilities, and recommended namespace that are used in this project. The web service artifacts are Enterprise, Entity, and Message XML schemas, and a WSDL document. The Enterprise
XML Schemas are composed of common data types that are reusable across all Schemas. For example, a social security type would be defined here. The Entity XML Schemas defines the data structure (the name, type, occurrences, etc.) for a particular data entity. For example, Address schema would define address, city name of type string, state code enumeration, etc. The message schema defines a particular interface for the service. For this project it is the GetCustomer interface. The message schema uses data types defined in the Enterprise and Entity XML schemas to build its request and response messages data. This becomes the “gut” of the SOAP body that gets passed via HTTP. It is recommended to have a common reusable schema repository for an enterprise. It provides a common data definition for messages for the enterprise. The WSDL document describes the Customer Profile Java Service. This document is the contract for the clients. Ultimately, the Java web service is built upon the schemas defined below in Table 3.

<table>
<thead>
<tr>
<th>Web Service Artifact</th>
<th>Name</th>
<th>Responsibilities</th>
<th>Recommended Namespace</th>
</tr>
</thead>
</table>
3. Database Model

The database model, Figure 27, shows the tables (column names, types, constraints) that are contained within the enterprise database and their relationships. For this project there are five simple tables, PHONE, EMPLOYMENT, CUSTOMER, ADDRESS, and NAME. Each of the table has a primary key, usually the table name with _ID, for example, PHONE_ID. All the tables except for the ADDRESS table have foreign keys. An address record can exist by itself. PHONE, EMPLOYMENT, and NAME tables has a CUST_ID foreign key. This means that in order for a record to exist in the tables (PHONE, EMPLOYMENT, and NAME) it must tie to an existing customer record. And CUSTOMER and EMPLOYMENT must tie to an existing address record for each record in their table.
4. Deployment Model

A deployment model shows how and where the system will be deployed. Physical machines and processors are reflected as nodes (rectangular boxes). Usually a deployment specification is specified to detail out nodes, by specifying, node names, processing power, memory, node type, etc. For this project the nodes below are located on one physical machine.
– my laptop. But for demonstration purposes, the diagram depicts them as being on several machines. The nodes are Client, IIS, MySQL, and TomCat.

The client node is a workstation that contains an instance of Internet Explorer 6 web browser. The web browser is how the end user will interact with the system. A Microsoft IIS web server node is below the client node. The ASP.NET web user interface components are deployed to this node. There is exactly one instance of the IIS web server for zero-to-many clients, connection via HTTP. To the right of the IIS web server node is the Apache TomCat node. This is the application server that houses the Java CustomerProfile web service and various supporting components. There is a one-to-one relationship between the IIS and the TomCat node. The connection is HTTP using SOAP. The last node is the MySQL node, which is the relational database that houses the customer data. This is also a one-to-one relationship to the application server using a local area network JDBC calls.

Figure 28: Deployment Model
5. Implementation Model

The implementation model presents the source code files namespace / package and assemblies / jars corresponding to the major classes and interfaces defined in the Logical Model Design Component Types. In other words, it expands by giving the types qualified namespaces / packages and physical assemblies / jars that it will be contained in. The implementation model also shows the sequence diagrams for the .NET system.

Two important concepts in .NET are namespace and assembly. Namespaces are needed to organize types into logical units, whereas assemblies provide physical containment. The concepts are the same in Java but instead are packages and jars respectively to namespace and assembly.

In Figure 29, namespaces are named by using the formula

<CompanyName>.<Project>.<Module>.<Component>. Therefore, using Sonnypconsulting for <CompanyName>, and MSIS for <Project>, most namespaces will carry the prefix Sonnypconsulting.MSIS. Table 4 shows the .Net namespace and its assembly name and the node and layer (from design model) that it resides in.
**Figure 29: Namespace Model .Net**

**Table 4: Node: Layer to Namespace: Assembly Traceability Matrix for .NET**

<table>
<thead>
<tr>
<th>Node: Layer</th>
<th>Namespace: Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIS: User Interface</td>
<td>Sonnyconsulting.WebUI.ClientMSIS.View: Sonnyconsulting.WebUI.ClientMSIS.dll</td>
</tr>
<tr>
<td></td>
<td>Sonnyconsulting.WebUI.ClientMSIS.dll</td>
</tr>
<tr>
<td>IIS: Business Logic</td>
<td>Sonnyconsulting.MSIS.Business.Adapter: Sonnyconsulting.MSIS.Business.dll</td>
</tr>
<tr>
<td></td>
<td>Sonnyconsulting.MSIS.Business.Controller: Sonnyconsulting.MSIS.Business.dll</td>
</tr>
<tr>
<td>IIS: Business Entity</td>
<td>Sonnyconsulting.MSIS.Model: same as namespace but with a .dll extension</td>
</tr>
<tr>
<td>IIS: Data Access</td>
<td>Sonnyconsulting.MSIS.DataAccess: Sonnyconsulting.MSIS.DataAccess.dll</td>
</tr>
<tr>
<td></td>
<td>Sonnyconsulting.MSIS.DataAccess.dll</td>
</tr>
</tbody>
</table>
In Figure 30, packages are named by using the Java formula

`com.<CompanyName>.<Module>.<ComponentType>`. Therefore, using sonnypconsulting

for `<CompanyName>`, and service for `<Module>`, most namespaces will carry the prefix

`com.sonnypconsulting.service`. Table 5 shows the Java package and its jar name and the node

and layer (from design model) that it resides in.
<table>
<thead>
<tr>
<th>Node: Layer</th>
<th>Package: Jar</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>com.sonypconsulting.service.provider.customerprofile.getcustomer2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.phone2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.employment2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.fault2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.address2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.name2005: GenCode.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.entity.individual2005: GenCode.jar</td>
</tr>
<tr>
<td>TomCat: Business Logic</td>
<td>com.sonypconsulting.service.customerprofile.impl: CustomerProfileService.jar</td>
</tr>
<tr>
<td></td>
<td>com.sonypconsulting.service.customerprofile.adapter: CustomerProfileService.jar</td>
</tr>
<tr>
<td>TomCat: Business Entity</td>
<td>com.sonypconsulting.service.customerprofile.model: CustomerProfileService.jar</td>
</tr>
<tr>
<td>TomCat: Data Access</td>
<td>com.sonypconsulting.service.customerprofile.dao: CustomerProfileService.jar</td>
</tr>
</tbody>
</table>

The sequence diagrams in Figure 31, 32, 33, and 34, shows the structured representation of the .Net system's behavior as a series of sequential steps over a period of time. It is used to depict work flow, message passing and how objects in general cooperate over time to achieve a result. That is, calling the Java Web Service and displaying the data.

Table 6, 7, 8, and 9 explains the sequence diagrams (Figure 31, 32, 33, and 34) in steps, in regards to messages, who initiated the message (from object), to whom is it requesting the message (to object), and comments about the messages.
Figure 31: Get Customer View Sequence Model

Table 6: Get Customer View Sequence Model Steps

<table>
<thead>
<tr>
<th>Id</th>
<th>Message</th>
<th>From Object</th>
<th>To Object</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Page_Load</td>
<td>Actor</td>
<td>CustomerProfileFrm</td>
<td>When actor navigates to the page this is the first thing that happens</td>
</tr>
<tr>
<td>2</td>
<td>CustomerProfileController</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileController</td>
<td>The customer profile form calls the constructor of the form controller</td>
</tr>
<tr>
<td>3</td>
<td>CustomerProfileAdapter</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileAdapter</td>
<td>The controller constructs the customer profile adapter</td>
</tr>
<tr>
<td>4</td>
<td>AddValidation</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileFrm</td>
<td>Method adds validation to page</td>
</tr>
<tr>
<td>5</td>
<td>AddRegExValidator</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileFrm</td>
<td>Adds Regular Expression validations to text boxes</td>
</tr>
<tr>
<td>6</td>
<td>AddRegFieldValidator</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileFrm</td>
<td>Adds Required Field validations to the text boxes</td>
</tr>
<tr>
<td>7</td>
<td>AddValidationSummary</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileFrm</td>
<td>Adds validations to validation summary for page</td>
</tr>
<tr>
<td>8</td>
<td>GetCustomerBut_Click</td>
<td>Actor</td>
<td>CustomerProfileFrm</td>
<td>Actor clicks on submit button on page</td>
</tr>
<tr>
<td>9</td>
<td>Submit</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileController</td>
<td>Submits page values to controller builds request object</td>
</tr>
<tr>
<td>10</td>
<td>GetData</td>
<td>CustomerProfileController</td>
<td>CustomerProfileController</td>
<td>Method contains 11, 12, 13, 14, 15, 16</td>
</tr>
<tr>
<td>11</td>
<td>ModelToSchemaType</td>
<td>CustomerProfileController</td>
<td>CustomerProfileController</td>
<td>Adapts the request model to the generated schema type</td>
</tr>
<tr>
<td>12</td>
<td>GetCustomer</td>
<td>CustomerProfileController</td>
<td>CustomerProfileDAO</td>
<td>Calls the DAO passing in the adapted schema type</td>
</tr>
<tr>
<td>13</td>
<td>getCustomer</td>
<td>CustomerProfileDAO</td>
<td>CustomerProfileDAO</td>
<td>This calls the web service</td>
</tr>
<tr>
<td>14</td>
<td>SchemaTypeToModel</td>
<td>CustomerProfileController</td>
<td>CustomerProfileAdapter</td>
<td>Adapts the schema type to the model</td>
</tr>
<tr>
<td>15</td>
<td>ResponseModeToCustomerModel</td>
<td>CustomerProfileController</td>
<td>CustomerProfileAdapter</td>
<td>Adapts the model to a customer object</td>
</tr>
<tr>
<td>16</td>
<td>Customer</td>
<td>CustomerProfileController</td>
<td>SessionManager</td>
<td>Puts the customer object with response info in the session manager</td>
</tr>
<tr>
<td>17</td>
<td>HttpRedirect</td>
<td>CustomerProfileFrm</td>
<td>CustomerProfileController</td>
<td>Redirects the page to a fault page or else the display customer information page</td>
</tr>
</tbody>
</table>
Figure 32: Display Customer Information Sequence Model

Table 7: Display Customer Information Sequence Model Steps

<table>
<thead>
<tr>
<th>Id</th>
<th>Message</th>
<th>From Object</th>
<th>To Object</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Page_Load</td>
<td>CustomerProfileFrm</td>
<td>DisplayCustomerInfoFrm</td>
<td>This method is called every time the page loads</td>
</tr>
<tr>
<td>2</td>
<td>DisplayCustomerInfoController</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoController</td>
<td>Initializes controller</td>
</tr>
<tr>
<td>3</td>
<td>DisplayCustomerInfoAdapter</td>
<td>DisplayCustomerInfoController</td>
<td>DisplayCustomerInfoAdapter</td>
<td>Initializes the adapter</td>
</tr>
<tr>
<td>4</td>
<td>IsCustomerSessionSet</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoController</td>
<td>Checks to see if customer object is set within the session manager</td>
</tr>
<tr>
<td>5</td>
<td>PopulateGrid</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoFrm</td>
<td>Populates the grids on the page by wrapping steps 6-10</td>
</tr>
<tr>
<td>Id</td>
<td>Message</td>
<td>From Object</td>
<td>To Object</td>
<td>Comments</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>InitializeGrid</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoFrm</td>
<td>17 Clears the grid</td>
</tr>
<tr>
<td>7</td>
<td>GetDataView</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoController</td>
<td>Calls GetDataView method in controller with specified data type</td>
</tr>
<tr>
<td>8</td>
<td>Customer</td>
<td>DisplayCustomerInfoController</td>
<td>SessionManager</td>
<td>Gets the customer object from session manager</td>
</tr>
<tr>
<td>9</td>
<td>CustDataView</td>
<td>DisplayCustomerInfoController</td>
<td>DisplayCustomerInfoAdapter</td>
<td>Adapts the customer object, specifically the customer info to a data view so the form can bind it to the specific grid on the page</td>
</tr>
<tr>
<td>10</td>
<td>InitializeGrid</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoFrm</td>
<td>clears the grid</td>
</tr>
<tr>
<td>11</td>
<td>GetDataView</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoController</td>
<td>Calls GetDataView method in controller with specified data type</td>
</tr>
<tr>
<td>12</td>
<td>Customer</td>
<td>DisplayCustomerInfoController</td>
<td>SessionManager</td>
<td>Gets the customer object from session manager</td>
</tr>
<tr>
<td>13</td>
<td>CustAddressDataView</td>
<td>DisplayCustomerInfoController</td>
<td>DisplayCustomerInfoAdapter</td>
<td>Adapts the customer object, specifically the address info to a data view so the form can bind it to the specific grid on the page</td>
</tr>
<tr>
<td>14</td>
<td>InitializeGrid</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoFrm</td>
<td>clears the grid</td>
</tr>
<tr>
<td>15</td>
<td>GetDataView</td>
<td>DisplayCustomerInfoFrm</td>
<td>DisplayCustomerInfoController</td>
<td>Calls GetDataView method in controller with specified data type</td>
</tr>
<tr>
<td>16</td>
<td>Customer</td>
<td>DisplayCustomerInfoController</td>
<td>SessionManager</td>
<td>Gets the customer object from session manager</td>
</tr>
<tr>
<td>17</td>
<td>EmploymentDataView</td>
<td>DisplayCustomerInfoController</td>
<td>DisplayCustomerInfoAdapter</td>
<td>Adapts the customer object, specifically the employment info to a data view so the form can bind it to the specific grid on the page</td>
</tr>
</tbody>
</table>

Figure 33: Display SOAP Information Sequence Model

Table 8: Display SOAP Information Sequence Model Steps
<table>
<thead>
<tr>
<th>Id</th>
<th>Message</th>
<th>From Object</th>
<th>To Object</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Page_Load</td>
<td>CustomerProfileFrm</td>
<td>FaultFrm</td>
<td>Method is called every time page is loaded</td>
</tr>
<tr>
<td>2</td>
<td>FaultInfoController</td>
<td>FaultFrm</td>
<td>FaultInfoController</td>
<td>Initializing the controller</td>
</tr>
<tr>
<td>3</td>
<td>FaultInfoAdapter</td>
<td>FaultInfoController</td>
<td>FaultInfoAdapter</td>
<td>Initializing the adapter</td>
</tr>
<tr>
<td>4</td>
<td>PopulateGrid</td>
<td>FaultFrm</td>
<td>FaultFrm</td>
<td>Method to populate the grid</td>
</tr>
<tr>
<td>5</td>
<td>InitializeGrid</td>
<td>FaultFrm</td>
<td>FaultFrm</td>
<td>Clears the grid</td>
</tr>
<tr>
<td>6</td>
<td>GetDataGridView</td>
<td>FaultFrm</td>
<td>FaultInfoController</td>
<td>Calls controller the retrieve the data view</td>
</tr>
<tr>
<td>7</td>
<td>GetFaultList</td>
<td>FaultInfoController</td>
<td>SessionManager</td>
<td>Gets the fault object from the session manager</td>
</tr>
<tr>
<td>8</td>
<td>FaultInfoAdapter</td>
<td>FaultInfoController</td>
<td>FaultInfoAdapter</td>
<td>Adapts the fault object to the data view so form can bind data to grid</td>
</tr>
</tbody>
</table>

**Figure 34:** Display Fault Information Sequence Model

**Table 9:** Display Fault Information Sequence Model Steps
RESULTS AND DISCUSSION

As a result from the System Design phases a .NET Client System and a Java Customer Profile Service were built. Tools used were Enterprise Architect, Visual Studio .NET, IIS, XML Spy, MyEclipse, Apache TomCat 4.1, MySQL 5.0, Apache Axis 1.2.1, and Hibernate.

The design for both systems was modeled in Enterprise Architect. Enterprise Architect integrated development environment (IDE) tool allows UML modeling and forward engineering to source code. Twenty six diagrams, from sequence, to activity, to class diagrams were created to model the solutions.

The .NET Client system was built using Visual Studio .NET, using the 1.1 framework. Seven .aspx files, nineteen C# source code files, one cascade style sheet, and one html file were created to build the ASP .NET Client front-end system. Microsoft's IIS 5.1 web server is used to host the front-end system.

To provide a common messaging data language for the Java Web Service, XML Spy 2004 Enterprise Edition was used to create the Web Service artifacts, i.e., XML Schemas and WSDL. XML Spy is a visual IDE tool for the creation of XML, XML Schema, WSDL, and SOAP documents. A total of nine Web Service artifacts were created.

The Java Customer Profile Service was built using MyEclipse IDE. MyEclipse is an IDE built upon the open source Eclipse IDE offering more plugins to make java development easier. Twenty five java source code files were generated using one build XML ant script, using Apache Axis WSDL2Java tool, to generate java types from the WSDL document. The XML ant script is used to build source code, jar up compiled code, reference and use tools, and deploy code. The Java Customer Profile Service implementation code resulted in twenty three java source code files, six hibernate XML files, and a build XML ant script. Hibernate
is an object relational mapping tool. It allows a developer to point to tables in the database and hibernates creates corresponding entity java types and manages the persistence for the developers. In other words, developers only have to know how to deal with Java objects and not database objects such as stored procedures, etc. Hibernate XML files manages the mapping between the table columns and the java properties. Apache TomCat 4.1 J2EE application server was used as the container for the service. Apache Axis web application and supported libraries was also deployed to TomCat to help with SOAP handling for the Java Customer Profile Service. In other words, it takes the raw SOAP data from the wire and converts it to generated java types for the Customer Profile Service to work with.

MySQL 5.0 is used as the database system. From their website, MySQL is “the world’s most popular open source database.” An ENTERPRISE database was created to house the customer data. Five tables were created, NAME, PHONE, EMPLOYMENT, ADDRESS, and CUSTOMER. Hibernate was used to manage retrieving of customer data. SQL scripts were used to load the data.

For all the artifacts, source codes, scripts, models, tools, etc. please refer to appendix E: CD Contents and also the attached CD.

The screen shots for the .NET client system are referenced in appendix D. Figure 35, Get Customer Information Web Page, is the default page when you pull up the system. At this point the user can follow the steps on the screen or navigate by selecting one of the links on the side. If they click on one the links, the page will display with no information. If the user chooses to follow the steps on the screen they are then directed to Figure 36, Customer Detail Information Web Page. At this point, the Java Service was called, and the data retrieved from the database and displayed on the page, based on the user entered information.
The customer data is displayed in three tables, one for general customer information, and another for customer address information, and the last for customer employment information. If the user requested customer information is not in the database and or an error was thrown the user will be redirected to the error page, see Figure 38. This page will display friendly error messages, technical messages, stack trace, advice text, embedded exception, and host name that threw the error in a table. In real world applications, only the friendly messages would be displayed to the user. The rest of the information would be logged and an automatic email would be sent behind the scenes to the support group. However, this demo system captures the essence of what should happen. And lastly, after viewing the customer information page the user can think click on the SOAP Info Page link to view the raw SOAP request and response messages that are sent across the wire from the .NET client system to the Java Customer Profile Service. Again, this is probably not of interest to most users. This information should only be logged behind the scenes for troubleshooting purposes.

CONCLUSIONS

The objective of showing interoperability between heterogeneous systems using SOA with Web Service for the project was achieved by the .NET client system and the Java Customer Profile Service demo. All four deliverables mentioned above, that is, power point, .NET Web Page and artifacts, Java Customer Profile Web Service and artifacts, and MySQL database back end system and artifacts, in the objectives/deliverables section have been met.
Lessons Learned

One person playing the role of the Project Manager, Analyst, Designer, Developer, and DBA for a project will definitely add a tremendous amount of time and overhead that was not estimated up front due to no checks and balances and too big of a scope for project. In other words, no Project Manager really to manage the work of the other roles to make sure deliverables are met on time and some of the scope could have been removed from the project. With that said, the original time estimated in the work breakdown structure did not reflect the actual time it took to implement the project. Actual time was closer to two times than what was originally estimated. The lesson learned is not to try to accomplish everything in one project. Maybe I could have built the front end and stubbed out fake web service data to demo, or not build such a “complete” front end and create just the web service. However, if I would have removed some of the projects scope, I would have not learned holistically in regards to SOA and Web Services. A take away would be to have a better time estimates, next time a similar project comes around. This can be accomplished by interviewing various people that have done it before or if I am giving the estimates I know now to multiply the original time by two and stick with the project plan.

Future or Additional Work

When it comes to Service Oriented Architecture there are always new and upcoming additional works that can be researched, new topics such as, Enterprise Service Bus, Web Service Security, and Business Process Execution Language and UDDI to name a few buzz words. Further, additional interfaces could be added to the Java Customer Profile Service,
such as, UpdateCustomer and SearchCustomer. In addition, a better security model could also be implemented for both the .NET client and the Java Service.
REFERENCES


http://searchwebservices.techtarget.com/sDefinition/0,290660,sid26_gci929153,00.html
APPENDICES

APPENDIX A: HIGH LEVEL OVERVIEW

- Service-oriented architecture (SOA)
  - Description
  - Terminology
  - Business Drivers
- XML
  - Description
  - Role it plays
  - Schema
- Web Service
  - Description
  - Role it plays
  - Terminology
  - SOAP
  - Web Services Definition Language (WSDL)
- Demo
  - .Net Client
  - Steps in creating a J2EE Web Service – “the provider”
  - Interoperability between .Net and J2EE
    - .Net invoking / consuming a Java Web Service
APPENDIX B: WBS AND GANTT CHART

Work Breakdown Structure
<table>
<thead>
<tr>
<th>ID</th>
<th>Outline Num</th>
<th>May 15, '05</th>
<th>May 22, '05</th>
<th>May 29, '05</th>
<th>Jun 5, '05</th>
<th>Jul</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>SOA using Web service Project</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.1</td>
<td>Point Slide Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.1.1</td>
<td>Research Artifacts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.1.2</td>
<td>Put together Slides</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.2</td>
<td>MySQL Database Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.2.1</td>
<td>Design Document</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.2.2</td>
<td>Implement Physical Data Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.2.3</td>
<td>Populate Physical Tables</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.2.4</td>
<td>Unit Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1.3</td>
<td>Java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C: SCHEMA TECHNICAL DOCUMENTATION

Linked Document

Schema getCustomer2005.xsd

schema location: C:\MSIS\Development\Schemas\sonnyconsulting\service\provider\sp\customerprofile\2005\getCustomer2005.xsd


Elements

- getCustomer
- getCustomerResponse

Complex types

- GetCustomer Output Type
- GetCustomer Request Type
- GetCustomer Response Type

Elements

- fault
- faultActor
- faultDetail
- faultList
- faultReasonText
- faultType

Complex types

- Fault Type
- FaultDetail Type
- FaultList Type

Simple types

- FaultSeverity Enum
- FaultType Enum

schema location: C:\MSIS\Project\Development\Schemas\sonnyconsulting\service\entity\fault\fault.xsd

targetNamespace: http://service.sonnyconsulting.com/entity/fault/2005/

Elements

- address
- addressIndicator
- addressLineList
- cityName
- domesticAddress
- stateCode
- zipCode

Complex types

- Address Type
- AddressLineList Type
- DomesticAddress Type

Simple types

- StateCode Enum
- ZipCode Type

schema location: C:\MSIS\Project\Development\Schemas\sonnyconsulting\service\entity\address\address.xsd

targetNamespace: http://service.sonnyconsulting.com/entity/address/2005/

Elements

- enterpriseID
- ssn

Complex types

- EnterpriseID Type
- SSN Type

Simple types

-

schema location: C:\MSIS\Project\Development\Schemas\sonnyconsulting\service\entity\employment\employment.xsd

targetNamespace: http://service.sonnyconsulting.com/entity/employment/2005/
APPENDIX D: SCREEN SHOTS

Figure 35: Get Customer Information Web Page

Figure 36: Customer Detail Information Web Page
**Figure 37: SOAP Information Web Page**

**Figure 38: Error Information Web Page**
APPENDIX E: CD CONTENTS

Figure 39: CD Folder Structure

Table 10: CD Folder Structure Description

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSIS</td>
<td>Root Folder</td>
</tr>
<tr>
<td>Project</td>
<td>Project Folder</td>
</tr>
<tr>
<td>Design</td>
<td>This folder contains the Enterprise Architect model project file and word documents.</td>
</tr>
<tr>
<td>getCustomer</td>
<td>Folder contains generated models from Enterprise Architect to be viewable in MS Word.</td>
</tr>
<tr>
<td>Development</td>
<td>Contains the EclipseWorkspace, Sonnypconsulting.WebUI.ClientMSIS, and Tomcat 4.1 folders</td>
</tr>
<tr>
<td>EclipseWorkspace</td>
<td>Contains Customer Profile web service source code, Generated Java source code from XML Schema, XML Schemas, SQL Scripts, and all 3rd party libraries used to create the Java System.</td>
</tr>
<tr>
<td>Sonnypconsulting.WebUI.ClientMSIS</td>
<td>Folder contains all source code for .NET Client System.</td>
</tr>
<tr>
<td>Tomcat 4.1</td>
<td>Tomcat install and deployment folder for Java System.</td>
</tr>
<tr>
<td>Paper</td>
<td>Folder contains the final project paper and PowerPoint.</td>
</tr>
<tr>
<td>Tools</td>
<td>Folder contains all 3rd party tools used to implement demo.</td>
</tr>
<tr>
<td>axis2</td>
<td>Apache Axis web project</td>
</tr>
</tbody>
</table>