BUSINESS PROCESS EXECUTION LANGUAGE BASED SOFTWARE
IMPLEMENTATION FOR SMALL TRADING FIRMS WITH
SERVICE ORIENTED ARCHITECTURE

A Project
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Pratik Chetan Mehta
Spring 2011
BUSINESS PROCESS EXECUTION LANGUAGE BASED SOFTWARE IMPLEMENTATION FOR SMALL TRADING FIRMS WITH SERVICE ORIENTED ARCHITECTURE

A Project

by

Pratik Chetan Mehta

Spring 2011

APPROVED BY THE DEAN OF GRADUATE STUDIES AND VICE PROVOST FOR RESEARCH:

Katie Milo, Ed.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

Abdel-Moaty M. Fayek
Graduate Coordinator

Leonard Fisk, Ph.D., Chair

Melody Stapleton, Ph.D.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>Abstract</td>
<td>ix</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Purpose</td>
<td>2</td>
</tr>
<tr>
<td>Problem Statement</td>
<td>2</td>
</tr>
<tr>
<td>II. Literature Review</td>
<td>3</td>
</tr>
<tr>
<td>Background</td>
<td>3</td>
</tr>
<tr>
<td>Service Oriented Architecture</td>
<td>5</td>
</tr>
<tr>
<td>SOA Characteristics</td>
<td>6</td>
</tr>
<tr>
<td>Advantages Harnessed by Use of Service-Oriented</td>
<td>8</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Business Process Execution Language in Action</td>
<td>10</td>
</tr>
<tr>
<td>The Composite Application</td>
<td>27</td>
</tr>
<tr>
<td>Case Studies</td>
<td>27</td>
</tr>
<tr>
<td>III. Design and Implementation</td>
<td>31</td>
</tr>
<tr>
<td>Design</td>
<td>31</td>
</tr>
<tr>
<td>Scenario 1: Pharmaceutical Retail Outlet</td>
<td>32</td>
</tr>
<tr>
<td>Scenario 2: Brass Parts Trading Firm</td>
<td>34</td>
</tr>
<tr>
<td>Implementation and Testing</td>
<td>36</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IV. Summary and Conclusions</td>
<td>63</td>
</tr>
<tr>
<td>Scope and Limitations</td>
<td>63</td>
</tr>
<tr>
<td>Future work and Research</td>
<td>64</td>
</tr>
<tr>
<td>Conclusion</td>
<td>64</td>
</tr>
<tr>
<td>References</td>
<td>65</td>
</tr>
</tbody>
</table>

Appendices

| A. Integration of Public Web Services to the Project | 68   |
| B. Source for Further Information                | 74   |
ACKNOWLEDGMENTS

I would like to thank all the people who have helped and encouraged me during my Master’s project work.

It is a pleasure to thank my Committee Chair Dr. Len Fisk for his guidance and valuable advice during the course of this study. His meticulous inspection is commendable and truly helped me perfect my work. I would also like to thank my committee member, Dr. Melody Stapleton for her help and support. She has been a constant source of inspiration throughout my graduate study at Chico.

My deepest gratitude goes to my family for their unflagging love and support throughout my life. My father Chetan Mehta for introducing me to the wonderful world of science and for teaching me many things, dad what I am today is because of you. My mother Kalpana, I have no suitable word that can fully describe her everlasting love to me. My sister Pooja for being my best friend and for always believing in me. I also want to thank my friend Jigar for always being there for me, I couldn’t ask for a better friend.

And finally, I am extremely grateful to all my wonderful friends and batch mates especially, Abhishek, Ankit, John, and Rahil who have made graduate school so memorable and rewarding.
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GetLoanRate Web Service Flow Chart</td>
<td>11</td>
</tr>
<tr>
<td>2. Getloanrate BPEL – 1</td>
<td>12</td>
</tr>
<tr>
<td>3. Getloanrate BPEL – 2</td>
<td>12</td>
</tr>
<tr>
<td>4. Getloanrate BPEL – 3</td>
<td>12</td>
</tr>
<tr>
<td>5. Getloanrate BPEL – 4</td>
<td>13</td>
</tr>
<tr>
<td>6. Getrate WSDL – 1</td>
<td>14</td>
</tr>
<tr>
<td>7. Getrate WSDL – 2</td>
<td>14</td>
</tr>
<tr>
<td>8. Getrate WSDL – 3</td>
<td>15</td>
</tr>
<tr>
<td>9. Getrate XML Schema</td>
<td>15</td>
</tr>
<tr>
<td>10. SOAP Request</td>
<td>16</td>
</tr>
<tr>
<td>11. SOAP Response</td>
<td>16</td>
</tr>
<tr>
<td>12. The Web Services Architecture Stack As Described by the Open Source World Wide Web Consortium</td>
<td>19</td>
</tr>
<tr>
<td>13. SOA Roles</td>
<td>20</td>
</tr>
<tr>
<td>14. BPEL Designer – Source View</td>
<td>24</td>
</tr>
<tr>
<td>15. BPEL Designer – Design View</td>
<td>25</td>
</tr>
<tr>
<td>16. BPEL Designer – Mapper View</td>
<td>26</td>
</tr>
<tr>
<td>17. Composite Application – Design View</td>
<td>28</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>18.</td>
<td>Design Scenario 1 .................................................. 33</td>
</tr>
<tr>
<td>19.</td>
<td>Design Scenario 2 .................................................. 35</td>
</tr>
<tr>
<td>20.</td>
<td>XML File to Encode .................................................. 38</td>
</tr>
<tr>
<td>21.</td>
<td>Delivery Note XSD .................................................. 39</td>
</tr>
<tr>
<td>22.</td>
<td>Delimiter List Editor ................................................. 40</td>
</tr>
<tr>
<td>23.</td>
<td>Output for the Encoded XML File as Delimited Records Text File.... 40</td>
</tr>
<tr>
<td>24.</td>
<td>WSDL Document Based on SOAP Request ........................ 41</td>
</tr>
<tr>
<td>25.</td>
<td>WSDL Document Based on File Write – 1 .......................... 42</td>
</tr>
<tr>
<td>26.</td>
<td>WSDL Document Based on File Write – 2 .......................... 43</td>
</tr>
<tr>
<td>27.</td>
<td>Supplier BPEL Process ................................................. 44</td>
</tr>
<tr>
<td>28.</td>
<td>Supplier Composite Application .................................... 45</td>
</tr>
<tr>
<td>29.</td>
<td>sun-http-binding Properties ........................................ 46</td>
</tr>
<tr>
<td>30.</td>
<td>Web Service Testing Project - Endpoint URL ..................... 47</td>
</tr>
<tr>
<td>31.</td>
<td>Web Service Testing Project Output .............................. 47</td>
</tr>
<tr>
<td>32.</td>
<td>Purchaser BPEL Process - 1 ........................................... 49</td>
</tr>
<tr>
<td>33.</td>
<td>Purchaser BPEL Process with Pick Statement .................... 50</td>
</tr>
<tr>
<td>34.</td>
<td>BPEL Mapper – Finance Queue Output ............................ 51</td>
</tr>
<tr>
<td>35.</td>
<td>BPEL Mapper – Alarm Time ........................................... 52</td>
</tr>
<tr>
<td>36.</td>
<td>Purchaser BPEL Process – 3 ........................................... 53</td>
</tr>
<tr>
<td>37.</td>
<td>Define Correlation ..................................................... 54</td>
</tr>
<tr>
<td>38.</td>
<td>Purchaser Composite Application ................................... 55</td>
</tr>
<tr>
<td>FIGURE</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>40.</td>
<td>Material Management Composite Application</td>
</tr>
<tr>
<td>41.</td>
<td>Create New Test Case</td>
</tr>
<tr>
<td>42.</td>
<td>Auto Generated SOAP Document</td>
</tr>
<tr>
<td>43.</td>
<td>JMS Queue Error Output</td>
</tr>
<tr>
<td>44.</td>
<td>JMS Queue Finance Output</td>
</tr>
</tbody>
</table>
ABSTRACT

BUSINESS PROCESS EXECUTION LANGUAGE BASED SOFTWARE IMPLEMENTATION FOR SMALL TRADING FIRMS WITH SERVICE ORIENTED ARCHITECTURE

by

Pratik Chetan Mehta

Master of Science in Computer Science

California State University, Chico

Spring 2011

ERP Systems are generally thought of as being huge, bulky software and hardware systems that hold the functioning of large enterprises together. This notion is mostly true, but the picture is changing rapidly. Over the past two decades, a major proportion of computer software development has been in the ERP domain. However, such software systems are affordable only by sufficiently large businesses. Small scale industries and startups have similar business needs, though at a smaller scale. For instance, small scale industries too want a good business analysis tool to make smart business decisions and compete in the market. They also face issues with managing the business and improving the core processes of the company. ERP systems, though not affordable, would be very beneficial to smaller businesses.
CHAPTER I

INTRODUCTION

This project aims to implement a low cost ERP system for small businesses. Service-Oriented Architecture (SOA) will form the basis of achieving this goal [1]. SOA directly evolves from modular programming and distributed computing. In a nutshell, SOA is an architecture in which all processes are service oriented i.e. every function is thought of as a service, and these services communicate with one another to perform a transaction. Also, these services should not depend on any specific programming language, operating system, database, and application server type. These services may also be scattered over a network and there should be a loose coupling between these services. Thus, SOA is very closely associated with web services in the way that they enable distributed computing. Communication between different systems has always been a challenge and now standards are being adopted for this purpose. One of these standards is, Business Process Execution Language or BPEL. IBM, Microsoft and BEA jointly introduced Business Process Execution Language for Web Services (BPEL4WS, or just BPEL), in 2004 [6]. Its specification is now being managed by OASIS. BPEL has its origin in IBM's Web services flow language (WSFL) [10] and Microsoft's XLANG.
Purpose

The goal of this project is to implement an application based on BPEL and service oriented architecture, and to make maximum use of its capabilities in a project that would otherwise be impossible or very difficult to build using other known software engineering techniques.

Problem Statement

The purpose of this project is to build an application for a small trading firm that is based on the service oriented architecture incorporating BPEL technology. This application will be highly reusable and independent of any single underlying programming language or platform.
CHAPTER II

LITERATURE REVIEW

Background

“Service Oriented Architecture” is a new buzz word used to describe an industry that is supported heavily by ever increasing web services. SOA is more of a software paradigm, like object oriented programming. And, like object oriented programming, it is capable of revolutionizing the way in which we build software systems. With SOA, we can imagine a world in which all software systems can talk with one another without the underlying platform or language being a communication barrier. SOA is in its nascent stage of development. It needs to overcome many technological barriers to make this paradigm universally acceptable and profitable [1]. The book “Understanding SOA with Web Services” is an excellent source of information for SOA and its relation to web services [2]. It also places BPEL in the hierarchy of all web services. Technologists debate about the future path of SOA and its current standing in relation to other programming paradigms [12].

BPEL compliments SOA as it is used in orchestration of the web services in a SOA application. BPEL provides orchestration for both synchronous (client server) and asynchronous (peer to peer) web services. It is an open standard which is built on top of XML to provide communication between web services. It is portable across many development environments. It sets various guidelines for integrating separate services to
form a complete business application. Many approaches have been recognized to prove the completeness of BPEL [4], [5]. Some features, currently not implemented by BPEL, are also listed in research papers. Some practical examples have also been documented to illustrate the complete set of features of BPEL [3].

Many real life implementations of BPEL to solve issues similar to this project are available for reference [8], [9]. Tools are also available for mapping from UML to BPEL [14] and BPMN (business process modeling notation) to BPEL [13]. We can create UML diagrams to represent the business process and then transform it directly to BPEL code using the above transformations and tools. On the other hand, there are tools to translate from BPEL back to a graphical format to aid business analysts for process re-engineering [11]. All these tools are used on top of BPEL, to provide a complete graphical experience for programming an application utilizing this technology. The bulk of the programming involved in creating applications based on BPEL will be exercised using a graphical interface instead of the manual writing of code in XML, which can be error prone and time consuming. In the future, many graphical tools and development environments will be available for building BPEL and SOA based applications.

Research focusing on the uses of BPEL suggests that it is even applicable for complex scientific workflow [7]. Similar workflow patterns can be used for business development. The scope of BPEL is indeed very wide. A standard like this, which is supported by major vendors, will be extremely beneficial to manage and build web based applications. The direct extension to SOA is Software as a Service (SaaS). With the entire ERP application being service based, the client can now pay only for services which they use. So, unlike legacy systems, there will be no need to have complex and expensive
systems to enable ERP. Smaller businesses will be empowered with better services and inexpensive applications based on SOA. This new paradigm will change the way we design and implement software.

Service Oriented Architecture

Businesses have become very dynamic. With the ever growing use of technology and the Internet as a medium to compete in the market place, it has become a challenge for IT systems to keep up with the growing demands of businesses world over. IT systems need to respond quickly to changes in the business landscape and at the same time keep the cost of maintaining these systems to a minimum. Also, with mergers and acquisitions becoming a common practice, it is important to have the ability to integrate IT systems of different organizations. Integration of diverse IT systems is one of the biggest challenges today for the software industry. SOA plays a key role in such a vibrant market place. SOA based applications are more flexible to account for the ever changing needs of businesses.

Java is a platform independent programming language. XML is a self-describing, meta-language. Thus, data is platform independent when written in XML. HTTP is an Internet standard by means of which communication is achieved without platform dependence. Put all of these together and we get web services which can interconnect applications, data and programming logic in a platform neutral way. Applications programmed in any language, using any database, implemented over any application server, can talk over the Internet to a totally different set of applications located anywhere else in the world. This is the essence of any SOA implementation based
on web services. On the Internet we never really know or care about where the site we are surfing is actually hosted. Similarly, in a web services based SOA application we don’t need to know exactly where the server hosting the services is located. We also don’t care about what programming language the service is built in, and what backend database technology it utilizes. All that we see is that a service request was made and an appropriate response was provided.

Like object oriented programming, SOA is more of a programming methodology. In former practice, the main focus was on building modular code, such as classes and packages, and in the latter, it was on building modular and highly reusable code, available over the Internet and built out of multiple technologies or subsystems. It is very important to note that SOA does not automatically make the code more reusable or more modular. In object oriented programming, it is possible to program highly non modular code by writing the entire logic in one function. But this is a bad practice and over a period of time programmers learn to implement “oops” accurately and take full advantage of the features it offers. Similarly, with SOA, the programmer needs to follow certain design principles to take full advantage of the features of SOA.

**SOA Characteristics**

**Interoperability**

Service interoperability is very important. Services that are part of a collection or a pool of services need to communicate with one another in order to provide a complete end-to-end solution. To make interoperability possible, it is paramount to have a common interface so that each service can communicate with one another.
Loose Coupling

This is a concept similar to the “loose coupling” found in object oriented programming. Just as objects and classes are loosely coupled, SOA services should not be dependent on one another. Every service performs a single task, and many services together, orchestrated at runtime, perform the complete business process. Services can be aware of other services, but should not depend on them for data or any other service. There is a subtle difference between service awareness and service dependency.

Reusability

This, again, is a concept that is similar to “reusability” as used in object oriented programming, wherein classes and libraries can be reused. However, services hosted on the web have a further advantage, as they can be available to any application the world over. So these services can be called and reused not only by a single application but by any application which intends to call it and understands the interface of the hosted web service. Reusability is one of the key benefits of SOA. Service modules are designed to provide maximum reusability.

Communication

As described earlier, communication between services is paramount for interoperability. In this project, services will be invoked using SOAP (Simple Object Access Protocol) over HTTP. The interface language will be WSDL (Web Services Description Language).

Discoverability

Now since we have established that the services can be made available over the Internet and that these services also communicate with each other using a fixed
protocol, we can add one more concept on top of this. These services available on the Internet can be discovered using a dictionary. A discovery tool would enable us to find the required service at run time over the Internet.

**Vendor Lock-in Avoided**

The connections must be based on vendor independent standards. The technology stack of SOA is based on open standards and XML. It, thus, ensures there is no vendor lock-in and provides a highly compatible solution. Most previously distributed computing methods failed to achieve this. Examples are DCOM, CORBA and RMI.

---

**Advantages Harnessed by Use of Service-Oriented Architecture**

**Reuse of legacy system:** This is one of the biggest advantages of SOA. Legacy systems that are already in place cannot be replaced overnight. A lot of effort would go into recreating a new system from scratch every time an organization wants to upgrade its technology. Thus, reusing the existing system is not only efficient, but also important for organizations. The reuse of existing assets is at the core of SOA development. Various legacy systems in the enterprise can be reused and integrated to give maximum output, with minimum effort, in a SOA design. Services can be remodeled to interconnect software systems built in different programming languages and running on different application servers.

**Incremental Implementation**

This architecture allows for implementing an enterprise-wide solution in small increments. Services can be added and removed as required in a “plug-and-play” manner. Down-scaling and up-scaling the project is comparatively very simple in a SOA
implementation. This provides a huge benefit for managing and maintaining software systems in such enterprise-wide applications.

**Good Reuse of Existing Code**

SOA promotes maximum reuse of code and existing software pieces. In a manner similar to OOP, which helps the developer in building more manageable code, SOA helps the programmer in developing code that is generic and reusable. Note that SOA and OOP complement each other. Both these methodologies of programming are intended to be used together, to provide maximum benefit.

**Faster Time to Market**

New features and applications can hit the market very quickly. Reuse of code would minimize time to market and reduce costs. The building of new applications and the making of changes to current applications is very quick and efficient in an SOA environment, thus, allowing for timely upgrades to newer technologies.

**Reduced Costs**

As a result of reuse of tested code, there are lesser errors in new projects. This reduces the cost of development and maintenance. Also, since changes are easy to implement to loosely coupled services, maintenance costs greatly reduce over time.

**Continuous Business Process Improvement**

Business processes can be improved and extended component by component. With changing business needs, there are bound to be changes required in the application. These changes can be implemented quickly and managed more easily without the need for system redesign.
Process centric Approach to Development

Every component is developed from a process point of view. This enables more reuse of code for similar processes. Also, during development, there is greater conformity to requirements.

SOA allows implementation of new computing models like Grid computing and on-demand computing.

Business Process Execution Language

Business Process Execution Language (BPEL) is a XML based programming language which is used to program business processes. Every programming language has three core components which are logic, data types, and input/output. In the BPEL world, programming logic is written in BPEL, input/output is defined by WSDL (Web Service Description Language) and Data types are defined in XSD (XML Schema Definition).

Consider a simple GetLoanRate application to demonstrate the working of BPEL as shown in Figure 1. This BPEL application consumes a web service which does the calculation of the rate based on the input string provided. The web service functions as seen in Figure 1.

BPEL Process GetLoanRate Code

Figure 2 shows the BPEL code specifying the XML version and the encoding format. It also specifies the process name and the import namespaces. The import statement gives the location of the WSDL or the XSD files. It works similar to importing a library in a java class. The WSDL which defines the interface to the application is written in a separate file and is imported using this command. Thus, this WSDL can be
re-used multiple times with other applications. Also we will later see that this WSDL can in turn import another WSDL or XSD.

A WSDL defines the interface to a web service but does not actually establish the connection between the web service and its consumer. Partner link as seen in Figure 3 above does this task of establishing a connection to the web service’s end point. A partner link can be thought of as an instance of this web service. The partner link utilizes the getRateWSDL in the above example. Also we can see that it is associated to the web service’s port.
<process name="getLoanRateBPEL"
    targetNamespace="http://enterprise.netbeans.org/bpel/getLoanRateBPEL/getLoanRateBPEL"
    xmlns:tns="http://enterprise.netbeans.org/bpel/getLoanRateBPEL/getLoanRateBPEL">
    <import namespace="http://j2ee.netbeans.org/wsdl/getLoanRateBPEL/getRateWSDL"
        location="getRateWSDL.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
</process>

The variable names declared above as seen in Figure 4 are GetRateIn and GetRateOut. The type of these variables is described in the messageType tag. As we can tell from the type, these are custom variables.

<variables>
    <variable name="GetRateOut" xmlns:tns="http://Rate/"
        messageType="tns:getRateResponse"/>
    <variable name="GetRateIn" xmlns:tns="http://Rate/"
        messageType="tns:getRate"/>
</variables>
The code in Figure 5 demonstrates the orchestration of a business process. The business logic is written within the ‘sequence’ tag. It defines a sequence of steps to be performed one after the other. In the above sequence we can see that there is a receive operation, followed by an invoke operation, followed by a reply operation. There are also a couple of assign operators. Variable assignment is executed within this ‘assign’ tag. Also within the first receive tag we see that the CreateInstance flag is set to true. For every orchestration there needs to be at least one receive statement with CreateInstance as

```xml
<sequence>
  <receive name="Receive1" createInstance="yes" partnerLink="PartnerLink1"
    operation="getRateWSDLOperation" />
  <assign name="Assign1">
    <copy>
      <from variable="GetRateWSDLOperationIn" part="part1"/>
      <to>$GetRateIn.parameters/loanType</to>
    </copy>
  </assign>
  <invoke name="Invoke1" partnerLink="PartnerLink2" operation="getRate"
    xmlns:tns="http://Rate/" portType="tns:DetermineRateService"
    inputVariable="GetRateIn" outputVariable="GetRateOut"/>
  <assign name="Assign2">
    <copy>
      <from>$GetRateOut.parameters/return</from>
      <to variable="GetRateWSDLOperationOut" part="part1"/>
    </copy>
  </assign>
  <reply name="Reply1" partnerLink="PartnerLink1"
    operation="getRateWSDLOperation"
    xmlns:tns="http://j2ee.netbeans.org/wsd/loanRateBPEL/getRateWSDL"
    portType="tns:getRateWSDLPortType" variable="GetRateWSDLOperationOut"/>
</sequence>
</process>
```

Fig. 5. getLoanRate BPEL – 4.
true. The invoke statement consumes a web service called getRate which calculates and returns the rate.

The getRate WSDL

Figure 6 shows the getRate WSDL file’s content. A closer look will help us to understand the working of a WSDL document. XML namespace and target namespace seen in Figure 6 is used to distinguish between different web services. This is similar to the concept of namespaces as defined in XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="getRateWSDL"
targetNamespace="http://j2ee.netbeans.org/wsd1/getLoanRateBPEL/getRateWSDL"
xmlns:tns=http://j2ee.netbeans.org/wsd1/getLoanRateBPEL/getRateWSDL>

Fig. 6. getRate WSDL – 1.

Figure 7 shows two operations defined within the WSDL document. The operation getRateRequest takes one parameter with data type string. The operation getRateResponse returns one parameter with data type double. WSDL describes the input and output datatypes required to invoke a web service.

```xml
<types/>
<message name="getRateWSDLRequest">
  <part name="part1" type="xsd:string"/>
</message>
<message name="getRateWSDLResponse">
  <part name="part1" type="xsd:double"/>
</message>

Fig. 7. getRate WSDL – 2.
```
Figure 8 is the code for port type which defines a web service’s interface. It specifies the input and output operation under the operation tag. The specific WSDL operations are linked to input and output by the port type. Port type can be thought of as

```
<portType name="getRateWSDLPortType">
  <operation name="getRateWSDLOperation">
    <input name="input1" message="tns:getRateWSDLRequest"/>
    <output name="output1" message="tns:getRateWSDLResponse"/>
  </operation>
</portType>
```

Fig. 8. getRate WSDL – 3.

an abstract class. The implementation is not specified here but the interfacing architecture is described. There are a couple of other components which are currently blank for this WSDL, namely binding and service. The binding defines the type of communication. It could be SOAP, File, JMS, Database etc. Service defines the endpoint URL and the binding port.

**XML Schema for getRate**

Lastly, the loanRate XML Schema is as written below. Two elements are described, loanType as a string and getRate as double as seen in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/loanRateSchema"
    xmlns:tns="http://xml.netbeans.org/schema/loanRateSchema"
    elementFormDefault="qualified">
  <xsd:element name="loanType" type="xsd:string"/>
  <xsd:element name="getRate" type="xsd:double"/>
</xsd:schema>
```

Fig. 9. getRate XML Schema.
On creating a test case for the above web service, a SOAP Request and Response is created. The SOAP Request and Response is shown in Figure 10 and Figure 11 respectively.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:getRate xmlns:ns2="http://Rate/">
            <loanType>Home</loanType>
        </ns2:getRate>
    </S:Body>
</S:Envelope>
```

Fig. 10. SOAP Request.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:getRateResponse xmlns:ns2="http://Rate/">
            <return>15.0</return>
        </ns2:getRateResponse>
    </S:Body>
</S:Envelope>
```

Fig. 11. SOAP Response.

In the SOAP request, we supply the input loan type as “Home.” The response gives the value as 15.0 for the rate. The Request SOAP document is automatically created for testing the web service by the NetBeans IDE.
BPEL Features

BPEL supports structured programming constructs like if-then-else, while, sequence and flow. Above example was arranged as a sequence where a single web service was called out and the results returned back to the user. Processes can be created and terminated in this language. Future releases may support process suspend and resume. BPEL also supports synchronous and asynchronous processes.

Some advanced features that are commonly found in BPEL engines are as follows: endpoint status monitoring, process logging and alerts, multiple thread execution, debugging of business processes, database persistence of business processes for reliable recovery on system failure, load balancing and failover in clustered applications, configuration of quality of service properties and message redelivery configurations. The core features of BPEL are listed below.

**Orchestration.** BPEL or Business Process Execution Language is an orchestration language. Loosely coupled services available over the web are managed and invoked by this orchestration engine. Business rules and logic can be implemented into the software by using this orchestration system. BPEL is the intermittent system over which the various services talk to each other to provide a complete business solution. BPEL can implement both synchronous (client-server) and asynchronous (peer-to-peer) web services transactions. Asynchronous handling of web services allows for long running transactions, which are common in business scenarios.

Some of the most powerful features of BPEL include asynchronous message handling, reliability and recovery. Also, BPEL is an open standard technology based on XML, making it highly portable and interoperable across various environments.
**Message Handling.** BPEL supports synchronous as well as asynchronous messaging. It gives developer control over what messages are transmitted and when they are processed. BPEL engine stores the messages in a database, thus freeing up other resources. This is how it can handle transactions over long periods of inactivity.

**Reliability and Recovery.** BPEL provides fault handlers to deal with run time faults that may occur, such as unavailability of service or database reading error. It also provides compensation handlers that can be used to roll back transactions or entire business processes.

**Tool Support.** Commercial BPEL engines today provide various tools, such as monitoring of business process states, effective utilization of services, monitoring of quality parameters of services and load balancing to name a few. To be able to see the data flow and processes at a fine grained level, is one of the biggest advantages in these new technologies.

**Open Standard.** BPEL, being open standard, is highly portable. Various scripts can be developed, shared and reused within and outside the organization.

**Web Services**

According to W3C, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [14]

Web services can be thought of as a software component available over the Internet and communicating using open standards technology based on XML.
The technologies considered in the web services architecture stack as seen in Figure 12 are XML, WSDL, SOAP and other communication technologies like HTTP, SMTP, FTP etc. These form the core technology stack. There are many more technologies and standards, not described here, related to web services.

Fig. 12. The web services architecture stack as described by the Open Source World Wide Web Consortium.

A web service in a SOA implementation has three roles and three operations, as shown in Figure 13. The three roles are that of a service provider, a service requester and a service registry, and the three operations are publish, find and bind.

![Fig.13. SOA roles.](image)

A service provider publishes a service on the service registry. This way, it makes the service available for any consumer request. It can be thought of as a server side agent in a 2 tier client-server communication. A client requests a service and the service provider offers the service over the Internet.

A service requester consumes a service and can be thought of as a client in a 2 tier client-server communication. The requester uses the find operation to search for a service in the service registry. Once the service is found, the client can consume the service and perform its task.
A service registry is like a broker that sits between the client and the server. Its task is to make the services searchable over the Internet. The requester can search for the web service on the registry, and once the service is found, it can directly bind to the respective service provider server. The overall working of the system is just like a simple client server communication with a broker in between which is a lookup table of all the published services.

The main architectural components of web services in a SOA are transport, description and discovery.

The transport layer consists of all the communication protocols by means of which different web services talk to each other. The supported communication protocols are HTTP, FTP, SMTP, JMS and others, as shown in the architectural diagram of the SOA. Web services send messages over the Internet using these protocols. The data is commonly in XML form.

The description layer is supported by WSDL (Web Service Description Layer). This describes what the web service is and what operations it is set to perform. It also provides other details such as, what are the parameters needed to commence the service. Apart from that, it also describes the format of the input parameters and results.

The Discovery layer is provided by UDDI (Universal Description Discovery and Integration). This layer is responsible for enabling service discovery. Published services can be searched and attached to the application at compile time (static) or run time (dynamic).
Open ESB, short for Open Enterprise Service Bus, is an open source implementation based on Java created by Sun Microsystems. Open ESB is implemented by the GlassFish server which again is an open source server, made available by Sun Microsystems. This SOA project is implemented on the GlassFish server which supports Open ESB and Java Enterprise Edition components. An Enterprise service bus is a middleware technology. Its primary goal is to provide a framework for communication between services in an SOA.

Currently the NetBeans IDE supports many tools for development in Open ESB. The tools include editors for BPEL, XSLT, XSD and WSDL. Support for eclipse is limited as of now. A management console is present at this level. It performs web services management related tasks like monitoring, logging, auditing etc. Also, Open ESB supports complex event processing like pattern matching and correlation. Quality of service parameters can also be established by Open ESB. Open ESB supports web services, HTTP, FTP, LDAP, XSLT, various Database technologies, XML-based technologies and more.

The BPEL Designer

The BPEL Designer is a graphical tool used for BPEL coding. It adheres to the standards and guidelines laid down by WS-BPEL 2.0 technology. The BPEL Designer is available within the NetBeans IDE. It is used for orchestrating services based on business rules. It looks similar to a flow chart diagram showing the composition of various services in the application. This graphical representation of the composition of various web services is extremely beneficial to visualize the working of the entire
business process. It is more intuitive than directly looking at the code to understand the business rules and processes. This service orchestration is the point where business users and technology developers join hands and discuss to see if the application functions as expected.

The BPEL Designer consists of four views.

1. **Source View** (Figure 14): This displays the underlying source code behind the BPEL orchestration. Coding and editing can be performed on the source code directly in this view. Generally, coding directly in the source is tedious and error prone. The code looks like XML data and is difficult to work with directly. So this view is mainly used for editing and making small changes in the existing code.

2. **Design View** (Figure 15): This is a graphical view of the BPEL source code. It looks like a flow diagram when complete. Services can be dragged and dropped onto the designer canvas. Many other components, for managing the business process flow, are available in a template that helps to design the orchestration of the services. These components can be structural activities such as if, while, repeat until, for-each, flow, pick and so on, or web service invocation, reply, receive, assign and so on. The BPEL source code is automatically generated when designing in this mode.

3. **Mapper View** (Figure 16): The mapper view is used for assignments and conditions. It provides boolean, string, date & time and other operators. In the example below a concatenation operator is used. At this level, we can control what structure of data flows from one process to another.
Fig. 14. BPEL Designer – Source View.
Fig. 15. BPEL Designer – Design View.
Fig. 16. BPEL Designer – Mapper View.
4. Logging View: This view looks similar to the mapper view. As the name suggests, this view allows the developer to log real time data on the server. This is mainly used to keep a track of all necessary information related to the application, which may later be scrutinized, if an error is noticed at runtime. It is a graphical view used for logging and alerting the user of runtime information.

The Composite Application

The Composite Application outlined in Figure 17 is a service assembly which can be deployed on glassfish server or in general to any JBI runtime environment. The composite application gives a connection point for various project types having different communication protocols. Various SOA project types can be assembled in the composite application. The SOA project types may include a BPEL module, a XSLT module, web services testing project or intelligent event processing module.

WSDL bindings like ftp, http, jms, soap, soap12, ldap, database, file and scheduler are available to interconnect between various communication protocols. Quality of service parameter like max attempts, wait time, max concurrency limit and redirect or suspend on failure can be set for each invocation. Within this composite application, test cases can be executed on the deployed BPEL processes.

Case Studies

Case Study 1

- Domain: Trading firm (brass products, 4000 different brass items)
- Type of firm: Privately owned start up.
- Staff: 20 people
Fig. 17. Composite Application – Design View.
• Turnover: 1.5 Million dollars
• Net Profit: 250,000 dollars
• Branches: 3 branches in different cities.
• Number of executive at head branch: 1 (managing director)
• Number of sales manager: 2, one at each sub branch.
• Competition: About 500 to 800 similar and bigger brass factories in the same city.
• Near future goals: Exporting brass products, expanding the manufacturing plant.
• Special note: Dealing with over 200 clients. Very little automation and use of computer in the company.

• IT needs:
  - Statistical information/reports like average profit percent from each company.
  - Management of all communications/emails/drawings to refer/query easily at a later date.
  - Delayed payments by clients highlighted.
  - Progress of all products in manufacturing. i.e., at what stage of manufacturing they have reached and when will final product be estimated to ship and/or be ready for further processing and quality assurance.
  - A database of clients and products to generate reports about profitable firms and profitable products to make management decisions based on these reports.
Case Study 2

- Domain: Retail firm (pharmacy outlet)
- Type of firm: privately owned, small family business. Business commenced in 1950. Sales declined considerably during the last decade.
- Staff: 10 people
- Turnover: 750,000 dollars
- Net Profit: 190,000 dollars
- Branches: 1
- Number of executives: 1.
- Competition: Many pharmaceutical outlets in the surrounding locality.
- Near future goals: Divert business into medical equipments. Collaboration or merger is likely in the near future.
- Special note: No process automation, hence scope for a more efficient system.
- IT needs:
  - Computerizing the entire paper based system.
  - Management of stock i.e., keeping a track of how much is present in storage, and which medicines to order in what quantity based on past sales figures.
  - Expired medicines stock management.
  - Creating small procedures so that accounting and billing is easier.
  - Implement a system to print bills rather than handing out hand written bills to customers.
CHAPTER III

DESIGN AND IMPLEMENTATION

Design

Goals of the Implementation

1. Building an ERP application out of loosely coupled services.

2. These services should be reusable for different business scenarios. Maximize reusability whenever possible.

3. Services could be present on the local machine or hosted on a remote server.

4. Businesses might want to implement only a part of the solution provided by this project. That is different business scenarios will compose the application from a subset of the services that this project offers.

5. Support for long running transactions.

6. Support for legacy system integration. That is legacy system already implemented by the business should be integrated with this project.

7. Public web services available over the Internet should be integrated in the application to save coding man hours.

8. Business process failure or web service failure should be handled by the application.

9. Quality of service factors need to be addressed. That is how much load could be handled or how many concurrent transactions possible etc.
10. Security issues with web services need to be addressed.

11. Event processing support to track special type of patterns in the business. This would help the company in making better business decisions.

12. Providing automatic Email capabilities for example an email to the manager to read and approve a new purchase order for an under stocked item.

Scenario 1: Pharmaceutical Retail Outlet

The following diagram (Figure 18) shows the business process for the retail outlet.

This business requires an implementation for a simple purchase - supply business model. It does not require the material management module and accounting module.

There are two main processes, the purchaser process and the delivery process. In business terms these represent two departments in the company.

Each process is composed of web services and associated database. Also they might have other external connections for example to a JMS queue or a file system.

The following steps are performed by the system:

1. A customer can place an order to the front office.
2. The front office forwards the request to the delivery office.
3. The delivery department processes the request and writes the order to a file along with a time stamp of the current date and time.
4. The front office receives the completed order from the delivery department after a certain time lag.
5. The customer is served by the front office.

6. If the front office gets the delivery from the back office in a timely manner then the order is written to the finance JMS Queue.

7. If the order is not processed in a timely manner then the order is written to the error queue. The timing in this scenario is set according to the business rules of the company. For example 10 minutes would be a sufficient counter because we don’t want the customer to wait for more than 10 minutes to process his or her order.
Following steps are not explicitly programmed but are intended to be performed by the business.

1. The delivery department writes a delivery note to a flat file. This file could be used by a legacy system to perform more tasks for example say material management and accounting.

2. The error queue gives the management a good set of data to analyze how they can improve customer satisfaction. It gives a record of the transactions that took more than accepted time to process.

3. The finance queue can be connected to an accounting legacy system to maintain records and it can be printed out as a bill receipt for the customer.

Scenario 2: Brass Parts Trading Firm

Figure 19 shows the business process for the trading office. In this business scenario, the company wants to implement the whole stack of ERP system. Apart from the front office and back office we have added the materials management department and a process which orders the finished products from the factory against incoming customer orders.

The following steps are performed by the system:

1. A customer can place an order to the front office.

2. The front office forwards the request to the delivery office.

3. The delivery department processes the request and checks with the materials management process to see if the order required is in stock or not.
Fig. 19. Design Scenario 2.
4. If the order is in stock then the completed order proceeds from the delivery office to the front office. If the order is not in stock then a new factory order is created and the required vendor is notified for supplying the company the finished product. (There can be a long time lag when the order goes to the factory. Here we need support for long running transactions.)

5. A currency converter web service is also integrated which is useful if the vendor factory is located in another country and cost value needs to be calculated based on different currencies (not shown in diagram)

The following steps are not explicitly programmed but are intended to be performed by the business.

1. Legacy system could be connected to this application with the flat file or the JMS queues forming the input.

2. The front office could be connected to a web based application such that the customer can give orders directly online.

3. Also the web based interface to give orders can be used by the sales manager at other branches to submit orders.

Implementation and Testing

Implementation of the system includes several processes, each made up of web services and other application end point like JMS queues and database connections. The Business processes implemented are Supplier Process, Purchaser Process, Materials Management Process and Vendor Order (Factory Order) Process. Below are some of the key configuration steps used during implementation.
Initial Configuration

Some development tools as listed below have been installed.

- NetBeans IDE 6.7.1 along with JDK 6.14 and GlassFish Server Version 2.2–
  This is the preferred integrated development environment which implements a glassfish
  server and openESB for SOA development in Java. All the screen shots in this
  documentation are captured using this software.

- HermesJMS – This is an open source tool used to view messages in the JMS
  queues.

- SoapUI Plugin – This is a useful plugin for web service testing and
  development which is downloaded and installed in NetBeans IDE.

- Java Email Server – This is an open source tool which enables to send and
  receive emails directly from this java based application.

Encoding and Decoding a Purchase Order

Encoding is converting an XML file to a delimited record. Decoding is
converting back the delimited record to an XML file.

For example, we will convert a Purchase Order XML outlined in Figure 20 to
a delimited record flat file.

First we need to create an XML Schema definition which supports the above
format. For this we will create a Delivery Note XSD and a Purchase Order XSD. A
graphical representation of the delivery note schema is shown in Figure 21.

Delivery note Schema is a complex element consisting of a string (delivery
note date), integer (Purchase Order Number) and a collection of items. Each item has an
item number and quantity.
Once we have created the above delivery note schema, we can right click on the schema and choose to apply encoding. In the properties window we will apply the four levels of encoding.

As seen in the Figure 22, we add four levels. Level 1 ‘\r\n’ will mark the end of the record. Level 2 ‘|’ seperates the Delivery date, the order and the purchase order number. Level 3 ‘~’ seperates each item in the collection of items. Level 4 ‘^’ seperates the product number from the product quantity.
The output of the above encoding will result in a delimited text file as shown in Figure 23.

This means that the order number 2001 has 3 products, product 101 with quantity 10, product 102 with quantity 25 and product 103 with quantity 15. The remaining section is the date and time. Note that the reverse of this technique is also
Fig. 22. Delimiter List Editor.

Fig. 23. Output for the encoded XML file as delimited records text file.

Possible. Given the delimited text file as input and using the same rules as above, we get back the XML file. This process is called as decoding.

Creating WSDL File from XML Schema Definition

Purchase order WSDL. The XML Schema Definition (XSD) is used to define the structure of the data for the purchase order. The WSDL will utilize this definition to
understand the data structure. This WSDL is configured as a one-way operation. For the input message part, the purchase order schema element is selected as shown in Figure 24. The binding type for this WSDL is SOAP Request.

Fig. 24. WSDL Document based on SOAP Request.

Delivery Notes WSDL. A Concrete WSDL Document is created as shown in Figure 25. The binding type is selected as File Write. This WSDL document will also use
Fig. 25. WSDL Document Based on File Write – 1.

the Schema created for purchase order. Also we will use the encoder to convert from XML to flat file.

As seen in Figure 26 a pattern is supplied for the file name. Also, a directory location is provided on the local machine. Now, every new order which runs through this WSDL will get written to a folder on the local machine as a delimited record text file. We also provide the encoding type and the XSD element.

Creating a BPEL Process and a Composite Application

Creating the Supplier BPEL Process. The BPEL process is created as shown in Figure 27. Receive, assign and invoke elements are added to the canvas. A for each loop is added which will loop through each item in the purchase order. The input to the
business process is via the Purchase Order Input WSDL. The invocation operation takes place via the Delivery Notes write WSDL which utilizes the encoding logic (XML to flat file).

Creating the Supplier Composite Application

Right click on the supplier BPEL process and build it, the Supplier composite application as shown below in Figure 28 will be created automatically. This Composite Application is a service assembly which can be deployed on the glassfish server. Quality of service parameter like max attempts, wait time, max concurrency limit and redirect or
suspend on failure can be set for each invocation. Within this composite application, test cases can be executed on the deployed BPEL processes. Here we can see that a SOAP based input port is communicating with the supplier BPEL module. The output of this BPEL module is written to a file.

Test 1 – Creating a Web Service Test Project

The SoapUI plugin installed earlier will help us create a web service testing project to test the supplier process. We need to give the SOAP operation which will be an entry point for this test. This is done by adding a request on Input SOAP Operation. We need to modify the end point URL on which this request will run. For this we first find
the port on which the HTTP Binding Component is running. The component Sun-http-binding’s property will reveal this information. As shown in Figure 29, 9080 is the default HTTP port number in this case.

Edit the end point URL for the request we created and insert the port number 9080 in place of httpDefaultPort as shown in Figure 30. Enter the item number and quantity along with the purchase order number and date. Then click on submit request to specified endpoint URL.

The Output for this request is as displayed in Figure 31. HTTP code 202 means a successful execution. The time stamp is marked along with some more information.

Now check in the directory (user’s operating system) at the stored location for a flat file. Also the flat file name will be FileName_%d.txt. The following steps took place in a sequence during this test run.
1. A web service testing project which understands the communication protocol for input to the supplier process helped us to create an input SOAP document.

2. This SOAP document used the HTTP protocol to communicate with the endpoint URL and port number that we provided.

3. The sun-http-binding component of the glassfish server was listening on this port.
HTTP/1.1 202 Accepted  
Content-Type: text/plain; charset=iso-8859-1  
Content-Length: 0  
Date: Wed, 16 Mar 2011 06:32:59 GMT

Fig. 30. Web Service Testing Project - Endpoint URL.

Fig. 31. Web Service Testing Project Output.
4. The purchase order input WSDL which utilizes this binding component understood the incoming request and passed it on to the BPEL process via the partner link.

5. The BPEL process then forwards this input purchase order to the delivery note output WSDL which converts the XML to a delimited text record.

6. The delimited record is then written to a file on the server’s machine which can then be accessed by the store manager.

This is the supplier module in action. Now we will build a purchaser module which will demonstrate asynchronous business transaction processing.

**Asynchronous Transaction Processing**

Create the BPEL Process for Purchaser. Create partner links for five WSDLs as shown in Figure 32. Drop a receive and an invoke component onto the process canvas. The asynchronous processing will be handled using the pick command. The purcahser input WSDL will be the entry point for this process. Later we will define an asynchronous pick command which will utilize the poll WSDL to pick up an incoming purchase order. Depending on the specified time condition, the purchase order will then be forwarded to the Finance queue utilizing the JMS based finance WSDL or the Error queue utilizing the JMS based error WSDL.

Add a Pick Component to the BPEL Process. Add a pick statement from the palette on to the canvas. This is represented in Figure 33. The purchaser poll WSDL polls on a directory on the local machine to receive an incoming delivery note. As soon as the Delivery note matches a purchase order, the JMS Finance queue is invoked with the
Delivery Note details. This mapping from the delivery notes to the finance queue is seen in Figure 34.

The Delivery Note and the purchase order are both converted to a string using the Do Marshal operator. This string is concatenated and sent over to the finance queue for book keeping. Now a second possibility is that the order is not completed within a stipulated amount of time. For this we write an alarm code which will specify the time out condition. If the request gets timed out, the order will be forwarded to the error queue. Click on the OnAlarm function in the BPEL process and then enter the mapper view to set the designated time duration. As seen in Figure 35, the time is set to 30 seconds using the duration operator.
If the time out condition is met, the Purchase Order is marshaled into a string and sent to the error queue for book keeping. The entire Purchaser BPEL process will look as shown in Figure 36. We have thus simulated an asynchronous process which will poll a directory for incoming order delivery. If the delivery is completed by the back office in a timely manner, the request is forwarded to the finance department. If the delivery takes more time than the alarm condition, the error queue is updated. Later in the project separate emails will be created to automatically notify a manager of the completed order or an incomplete order.

**Add a Correlation.** One more important aspect of the asynchronous call remains to be coded. The polling mechanism needs to match up the delivery notes with
Fig. 34. BPEL Mapper – Finance Queue Output.

the purchase order. Orders and deliveries are asynchronous. For this we implement the correlation by right clicking on OnMessage and then clicking define correlation. We define a correlation on the PONumber which is present both in Delivery Note and Purchase Order as seen in Figure 37.

Create a New Purchaser Composite Application. Right click on the purchaser BPEL process and build it, the purchaser composite application as shown below will be created automatically. This Composite Application is a service assembly which can be deployed on the glassfish server. Quality of service parameter like max attempts, wait time, max concurrency limit and redirect or suspend on failure can be set for each invocation. Within this composite application, test cases can be executed on the deployed BPEL processes. Here we can see that there are two JMS queue ports, one for the finance department and the other for monitoring errors. Also there are two SOAP ports, one gives
the input to the purchaser BPEL process (similar to order at a front office) and the other
in turn calls the supplier BPEL process (similar to order completion by back office).
There is one file binding port which polls on the delivery note directory for an incoming
completed order. Once a match is established between the given order and completed
order, the finance JMS queue is filled in. If there is a timeout then the error JMS queue is
filled in.

After deployment, a test case is created to test these processes. If the files are
physically not transferred to the required folder within 30 seconds after the order has
been placed then the error queue is filled in otherwise the finance queue is filled in. The
Final purchaser composite application is shown below in Figure 38.
Summary of Steps Performed


2. Configure encoding and decoding based on delivery note Schema. That is conversion from flat file to XML and XML to flat file.

3. Test the decoding and encoding of the Delivery notes.

4. Create WSDL for purchase orders and delivery notes.

5. Create a BPEL module for supplier. Design the supplier with receive, invoke, assign and for each loop.

6. Create a supplier composite application.
7. Create a web service testing project. This helps in testing the composite application.

8. Construct the following WSDL files:
   a. To accept purchase orders from the front office.
   b. To accept Delivery Notes from supplier.
   c. To send confirmation to finance department using JMS send queue.
   d. To send error report using JMS send Queue.

9. Create a purchaser BPEL module. Design the purchaser with receive, invoke, assign and pick statements.

10. The pick statement is used for asynchronous processing. Also add a correlation to relate the purchase order and the supplied order.
11. Build the application. Deploy it. Create a test case and test it. If the purchase order is not transferred to the required folder within 30 seconds then the error queue is filled in otherwise the finance queue is filled in.

**Materials Management Process**

Figure 39 displays the BPEL process associated with material management. The essence of this process is to check the current stock and order extra stock if required to fulfill an order. As can be seen from the figure, there is a two way SOAP port called MMIn. MMIn provides the input to start the process. The first invoke statement checks against the database for the current stock of the product. Next, there is a conditional
A statement which checks if the current stock will suffice or an order to the vendor needs to be placed for more raw materials. If the current stock is sufficient, then the order is processed else a vendor order is created. The reply will contain the status of the current
process. Also a public web service for currency conversion is used to convert all currencies to a base US Dollars currency.

The Material management composite application is shown in Figure 40. There are two SOAP based ports and one database connection port and a JMS server port. The application is built and deployed on a glassfish server. Testing modules are written on this to perform unit testing.

![Material Management Composite Application](image)

**Test 2 – Testing Design Scenario: Pharmaceutical Retail Outlet.** A customer places an order for three items of different quantities.
To simulate this, we will create a new test case by right clicking on the Purchaser Composite Application | Test Folder.

Next we need to provide the WSDL document for which a test script will be created. We select the purchaser input WSDL outlined in Figure 41. Next select the operation to test on the next tab.

Fig. 41. Create new test case.

A SOAP document will be created automatically for the corresponding WSDL operation. We need to fill in the required inputs in the SOAP Document. The auto generated SOAP document is shown in Figure 42.
In the above document, we fill in the purchase order number, the purchase order date and a list of item numbers and respective quantity. This is the same as last test case. Now when we run this test, a file called DN_1.txt is created in the folder ‘..\ProjectName\fromSupplier\’ within 30 seconds to be processed by the finance department. Failing to do so would mean that the file will be forwarded to the Error JMS queue. This simulates the handling of an asynchronous process. The process of transferring the file from one location to another simulates a real world scenario of processing and delivering the purchase order.
The output of the JMS queue can be viewed using the HermesJMS tool.

Figure 43 displays the output for an order which was not processed in 30 seconds. Note the first line which tells the error queue manager that there has been a timeout during this transaction.

```
<pur:PurchaseOrder xmlns:pur="http://xml.netbeans.org/schema/PurchaseOrder">
  <pur:PONumber>2001</pur:PONumber>
  <pur:PODate>TBD</pur:PODate>
  <pur:Items>
    <pur:Item>
      <pur:ItemNumber>201</pur:ItemNumber>
      <pur:ItemQuantity>10</pur:ItemQuantity>
    </pur:Item>
    <pur:Item>
      <pur:ItemNumber>202</pur:ItemNumber>
      <pur:ItemQuantity>25</pur:ItemQuantity>
    </pur:Item>
    <pur:Item>
      <pur:ItemNumber>203</pur:ItemNumber>
      <pur:ItemQuantity>15</pur:ItemQuantity>
    </pur:Item>
  </pur:Items>
</pur:PurchaseOrder>
```

Fig. 43. JMS Queue Error Output.

Figure 44 displays the output for a successful transaction. It is forwarded to the finance queue telling the manager to go ahead and process the order.

The detailed steps occurring during the above process are as follows.

1. A customer sends an order request to the front office. This request is captured in a SOAP document and supplied as the start point to the purchaser business process.
Finance, Please Process This Order:  

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pur:PurchaseOrder xmlns:pur="http://xml.netbeans.org/schema/PurchaseOrder">
  <pur:PONumber>3001</pur:PONumber>
  <pur:PODate>TBD</pur:PODate>
  <pur:Items>
    <pur:Item>
      <pur:ItemNumber>301</pur:ItemNumber>
      <pur:ItemQuantity>5</pur:ItemQuantity>
    </pur:Item>
    <pur:Item>
      <pur:ItemNumber>302</pur:ItemNumber>
      <pur:ItemQuantity>15</pur:ItemQuantity>
    </pur:Item>
    <pur:Item>
      <pur:ItemNumber>303</pur:ItemNumber>
      <pur:ItemQuantity>25</pur:ItemQuantity>
    </pur:Item>
  </pur:Items>
</pur:PurchaseOrder>
```

Fig. 44. JMS Queue Finance Output.

2. The purchase order input WSDL passes this incoming request to the purchaser BPEL process via the partner link.

3. The purchaser process then invokes the supplier process along with the purchase order information.

4. Now all the steps that were described in test 1, earlier in this documentation are recited again. In summary the supplier process converts the purchase order to a delimited record text file. It then writes this to a file on the local machine.

5. Now we physically take this file with the delimited record and transfer it to the directory on which the purchaser process is polling on.
6. The WSDL which is polling on this directory picks up the file and forwards it to the purchaser process.

7. Using the correlation described on purchase order number, the process matches the delivery note to the purchase order.

8. Now the finance JMS queue gets updated with the purchase order completed successfully. However if there would be a timeout then the alarm condition would pick it up and send it to the error queue.
CHAPTER IV

SUMMARY AND CONCLUSIONS

Scope and Limitations

The project was implemented successfully with three business processes. These processes are architected to run independently from each other as well as collaboratively along with each other. Thus we get a plug and play kind of a scheme where different clients can use only services that interest them. BPEL orchestration was realized in each of these processes. The application can read and write to a file system on the local machine. It is also integrated with a java database system. The application seamlessly merges with publicly available web services. Asynchronous business transactions are handled. Interface to the various services is created in a reusable way using WSDL and XSD. JMS queues are integrated in the application. Interoperability is the primary feature of this application. The architectural philosophy for this project is to allow for interoperable loosely coupled web services orchestrated by business rules. This application has been greatly successful in achieving this. One major feature not accomplished during this implementation is the use of a web based user interface. One major cause of concern for this project is the lack of development on the glassfish server and the openESB project in the last one year due to a lack of funding and resources after Oracle Inc bought out Sun Microsystem. A roadmap for the future of openESB is not clear. Another aspect not studied in detail for this implementation is the security model.
Future work and Research

This project has a lot of future potential to build on. The primary aim of this project is to show the benefits of a SOA implementation in the ERP domain. With the tremendous amount of research in this field, it is probable that future applications will start using the principles of SOA to model software systems in almost every industry.

Conclusion

The project was implemented successfully demonstrating the use of a service oriented architecture for a small scale ERP project. The open source GlassFish BPEL engine was used to demonstrate the orchestration of various web services to create a business work flow. This project works as a proof of concept for a SOA based implementation of a business application. The scope of this project remains open ended as many more services and technologies can be integrated over the current basic framework to enhance the application.
REFERENCES


INTEGRATION OF PUBLIC WEB SERVICES TO THE PROJECT

List of Figures

Figure A-1: Import XSD From URL
Figure A-2: WSDL Based on HTTP Get with urlEncoding – 1
Figure A-3: Yahoo! GeoCoder BPEL Process
Figure A-4: BPEL Mapper - Yahoo! GeoCoder - 1
Figure A-5: Yahoo! GeoCoder Composite Application

Implementation

The information page for Yahoo! Geocoder used for geographical mapping is present at http://developer.yahoo.com/maps/rest/V1/geocode.html

The XML Schema Definition file for this web service is present at the following URL. http://local.yahooapis.com/MapsService/V1/GeocodeResponse.xsd This XSD file needs to be imported into the project as shown in Figure A-1.

Next we create a WSDL file from this XSD. The binding used in this WSDL is HTTP Get with urlEncoding as shown in Figure A-2.

The service is located at the given URI http://local.yahooapis.com/MapsService/V1/geocode. This will be the end point address on which the WSDL will connect. Create another SOAP based WSDL to trigger the BPEL process. The BPEL process looks as shown in Figure A-3. A request – response SOAP triggers the activation of this process. The invoke statement connects to the
Figure A-1. Import XSD from URL.

publicly available yahoo! Web service. Based on the input address parameters supplied, the yahoo! Web service gives back the latitude and longitude of the address.

The mapping is shown in Figure A-4. Apart from the street address, city and state we also supply a string code which the yahoo! Web service recognizes as a security token. This token is available on the information page for this web service.

The completed composite application which can be build, deployed and tested looks as shown below. We can see that a SOAP based WSDL triggers the start of this process which then invokes a public web service over the internet using the HTTP protocol (Figure A-5).
Figure A-2. WSDL Based on HTTP Get with urlEncoding – 1.
Figure A-3. Yahoo! GeoCoder BPEL Process.
Figure A-4. BPEL Mapper - Yahoo! GeoCoder – 1.

Figure A-5. Yahoo! GeoCoder Composite Application.
APPENDIX B


