Databases Integration through a Web Services Orchestration with BPEL using Java Business Integration

Wiranto Herry Utomo[1], Subanar[2], Retantyo Wardoyo[3], Ahmad Ashari[4]

[1]Faculty of Information Technology
Satya Wacana Christian University
Jl. Diponegoro 52-60
Salatiga 50711
Indonesia
(6298) 321212
wiranto.uksw@gmail.com

[2]Statistics, FMIPA,
Gadjah Mada University
Gedung SIC Lt. 3 FMIPA
Yogyakarta 55281
Indonesia
(6274) 555133
subanar@yahoo.com

[3]Computer Science Program, Gadjah Mada University
Gedung SIC Lt. 3 FMIPA
Yogyakarta 55281
Indonesia
(6274) 555133
rw@ugm.ac.id

[4]Electronics and Instrumentation Lab.
Physics Department
Gadjah Mada University
Yogyakarta 55281
Indonesia
(6274) 555133
ashari@ugm.ac.id

ABSTRACT: *Database Binding Component is a Java Business Integration component that provides database operations as services. The services exposed by Database Binding Component are actually SQL operations on tables. Database Binding Component exposes both DML and DDL operations as web services (Insert, Update, Delete, Find (Select) and Poll (Select Inbound)).*

This article wants to present an example of how to perform the select table from a remote database, how to send the result from the select table to the XML file, and how to insert extracted records into the extract table on the local database.

We have successfully extracted record table from remote database and written it into the XML file and inserted it to the local database in the Java Business Integration framework and we have composed the SOA Application through a Web Services Orchestration with BPEL (Business Process Execution Language).

Keywords: Java Business Integration, Database Binding Component, web services, SOA, BPEL

Received: 19 April 2009, Revised 29 June 2009, Accepted 1 July 2009

1. Introduction

In the past several years we have seen some significant technology trends, such as Service Oriented Architecture (SOA), Enterprise Application Integration (EAI), Business-to-Business (B2B), and web services. These technologies have attempted to address the challenges of improving the results and increasing the value of integrated business processes. The Enterprise Service Bus (ESB) draws the best traits from these and other technology trends.

The ESB concept is a new approach to integration that can provide the underpinnings for a loosely coupled, highly distributed integration network that can scale beyond the limits of a hub-and-spoke EAI broker. An ESB is a standard-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity.

There are a number of open source ESBs available now, including OpenESB from Sun, Mule ESB from MuleSource, and Apache ServiceMix, Spring Integration, Apache Tuscany, JBoss ESB, Apache Synapse. They have some key differences among themselves which we will not explore in this article. We have used OpenESB to test the demonstration lab in this article.

Open ESB is an ESB initiative started by Sun Microsystems and is hosted as a Java.net project. Open ESB is also an implementation of the JBI specification. JBI is the new integration API introduced in the J2EE world. It is a great enabler for SOA because it defines ESB architecture, which can facilitate the collaboration between services. It provides for loosely coupled integration by separating out the providers and consumers to mediate through the bus.

Java Business Integration (JBI) is a Java standard (JSR 208) for structuring business integration systems along Service Oriented Architecture (SOA) lines. It defines an environment for plug-in components that interact using a service model based directly on Web Service Description Language (WSDL) 2.0. **(Hove, 2006)** JBI is a messaging-based plug-in architecture. This infrastructure allows third-party components to be “plugged in” to a standard infrastructure and enables those components to interoperate seamlessly. The plug-in components function as service providers, or service consumers, or both. Components that supply or consume services locally (within the JBI environment) are termed “service engines.” Components that provide or consume services via some sort of communications protocol or other remoting technology are called “binding components”.

Database Binding Component is a Java Business Integration component that provides database operations as services. The services exposed by Database Binding Component are actually SQL operations on tables. Database Binding Component exposes both DML and DDL operations as web services (Insert, Update, Delete, Find (Select) and Poll (Select Inbound)). These webservices would be invoked by other JBI components acting as consumers **(Anonim, 2008)**.

This article wants to demonstrate a database select table from a remote server. The result set from the select table from a remote server is to be used to populate an extract table that exists on local database. So, we want to grab the records from the remote database server and insert them into a local database for convenient development and testing.

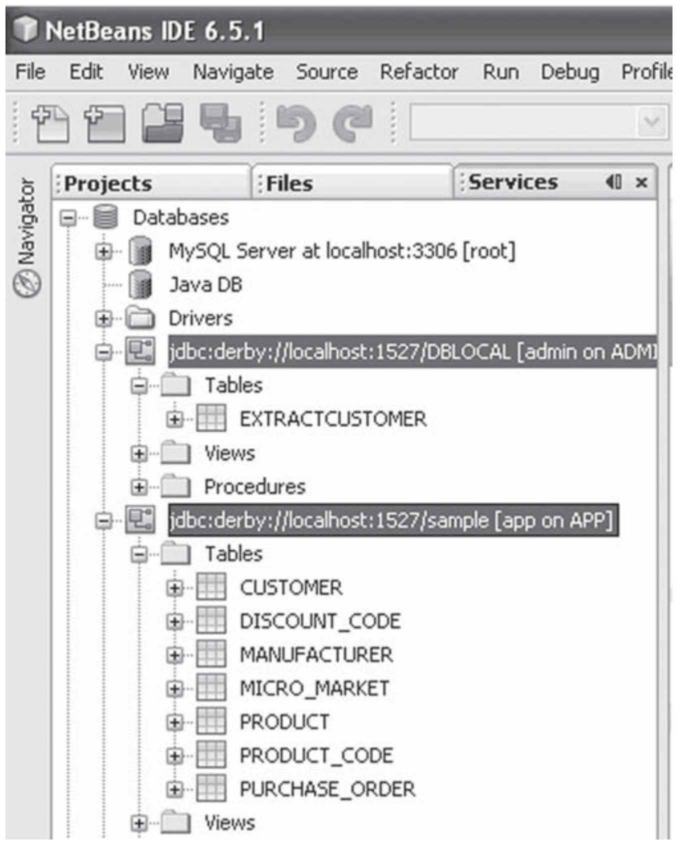


Figure 1 . Sample Java DB database as a remote database and DBLocal as a local database

We will assume that the both the "local" and "remote" databases actually exist on laptop. After all, accessing a remote database is just a matter of changing the URL for the database connection pool and having the right database driver installed in GlassFish. The demo really doesn't change based upon where the remote database resides.

So this article wants to feature the Database Binding Component to: **1)** Perform the select table from a remote database (the built-in "**sample**" Java DB database), **2)** Send the result from the select table to the file (output.xml), and **3)** Insert extracted records into the extract table on the local database (new Java DB database called **DBLOCAL** that we will create). (See figure 1)

2. Theoretical Foundation

2.1 Services Oriented Architecture

Service-oriented architecture (SOA) is an architectural style that modularizes information systems into services. Collections of these service will be orchestrated to bring business processes to life. In a successful SOA, these services will be recombined in various ways to implement new or improved business processes **(Brown, 2008).** SOA is a software system structuring principle based on the idea of self-describing service providers. In this context, a service is a function (usually a business function) that is accomplished by the interchange of messages between two entities: a service provider, and a service consumer **(Hove, 2006)**.

A service provider publishes a description of the services it makes available. A service consumer discovers and reads the service description, and, using only that description, can properly make use of the service by mechanism of message exchange. This is illustrated in **figure 2**. Note that the service provider and consumer share only two things: the service description, and the message exchange infrastructure **(Hove, 2006)**.

A major advantage of SOA is decoupling: the interface between the consumer and provider is very small. This permits controlled changes to the provider, that will not have unforeseen effects on the consumer. It also allows the consumer to switch providers easily, provided only that a compatible service interface is provided. Decoupling allows changes to occur incrementally. **(Hove, 2006)**

Service descriptions is JBI are made using the web services description language (WSDL). JBI uses a service model that is based on WSDL 2.0. Descriptions are published by service provider components, and read by service consumer components.

As shown in **figure 2**, service invocation is accomplished by a service consumer by sending an appropriate request message, via a messaging infrastructure (in this case JBI), to the service provider. Interaction between consumer and provider is always by means of messages. Thus, if the service provider is to provide a response to the service invocation, it will be in the form of a message. If the service provider wishes to return an error (fault) response to the request, it also is in the form of a message. **(Hove, 2006)**

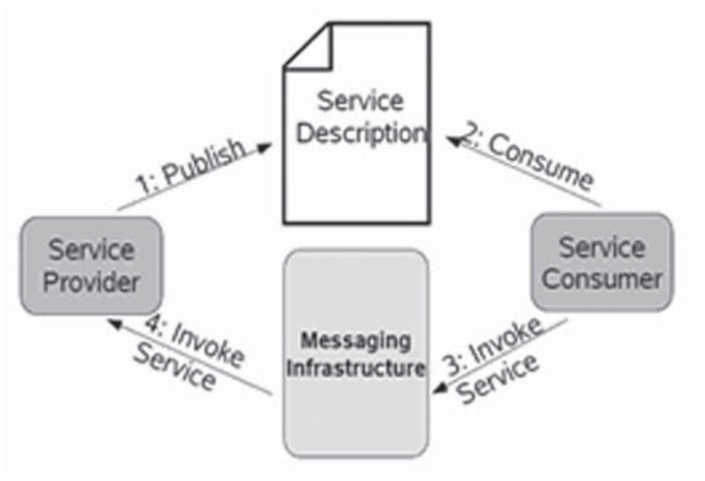


Figure 2. Service Consumer/Provider Interaction (Hove, 2006)

SOA is an architectural style for building enterprise solutions based on services. More specifically, SOA is concerned with the independent construction of business-aligned services that can be combined into meaningful, higher-level business processes and solutions within the context of the enterprise. **[7]**

2.2 Realizing SOA with Web Services

The requirement to expose functionalities of applications and access them remotely has resulted in several distributed architectures and middleware products, which emerge over time. The latest distributed architecture, which combines both synchronous and asynchronous communications, is **Web Services**. Web services are the most suitable distributed architecture for exposing the functionality of applications as services **(Juric, 2006)**. Web Service is a software system designed to support interoperable machine to machine interaction over a network. (W3C).

Web services are software systems designed to support interoperable machine-to-achine interaction over a network. This interoperability is gained through a set of XML-based open standards, such as WSDL, SOAP, and UDDI.. SOAP (Simple Object Access Protocol) is XML-based, message envelope format, WSDL (Web Services Description Language) is XML format for describing service interfaces. Typically used to generate server and client code, and for configuration. UDDI (Universal Description Discovery and Integration) is protocol for publishing and discovering metadata about Web services. These standards provide a common approach for defining, publishing, and using web services.

The final, and probably the most important, SOA concept is composition of services into business processes. Services are composed in a particular order and follow a set of rules to provide support for business processes. Composition of services allows us to provide support for business processes in a flexible and relatively easy way. It also enables us to modify business processes quickly and therefore provide support to changed requirements faster and with less effort. For composition, we will use a dedicated language, BPEL, and an engine on which business process definitions will be executed. Only when we reach the level of service composition can we realize all the benefits of SOA **(Juric, 2006)**.

2.3. Composition of Services into Business Processes

A business process is a collection of coordinated service invocations and related activities that produce a business result, either within a single organizations or across several. Business Process Execution Language for Web Services (BPEL, WS-BPEL, or BPEL4WS) is the new standard for defining business processes with composition of services **(Juric, 2006)**. It is the cornerstone of Service Oriented Architecture (SOA). With its ability to define executable and abstract business processes it opens new doors in business process management and represents the top-down approach to the realization of SOA.

The final aspect is the composition of exposed services of integrated applications into business processes. The most popular, commonly accepted, and specialized language for business process definition is **BPEL**. BPEL promises to achieve the holy grail of enterprise information systems—to provide an environment where business processes can be developed in an easy and efficient manner and quickly adapted to the changing needs of enterprises without too much effort **(Juric, 2006)**.

Composition of services into business processes requires the definition of collaboration activities and data-exchange messages between involved web services. WSDL provides the basic technical description and specifications for messages that are exchanged. However, the description provided by WSDL does not go beyond simple interactions between the client (sender) and the web service (receiver). These interactions may be stateless, synchronous, or asynchronous. These relations are inadequate to express, model, and describe complex compositions of multiple web services in business activities, which usually consist of several messages exchanged in a well-defined order.

2.4. Java Business Integration (JBI)

Java Business Integration (JBI) is a Java standard (JSR 208) for structuring business integration systems along SOA lines. It defines an environment for plug-in components that interact using a services model based directly on WSDL 2.0. **(Hove, 2006)**

The major goal of JBI is to provide an architecture and an enabling framework that facilitates dynamic composition and deployment of loosely coupled composite applications and service-oriented integration components. It allows anyone to create JBI- compliant integration plug-in components and integrate them dynamically into the JBI infrastructure **(Hove, 2006)**.

JBI is a messaging-based plug-in architecture. This infrastructure allows third-party components to be “plugged in” to a standard infrastructure and enables those components to interoperate seamlessly. It does not define the pluggable components themselves, but defines the framework, container interfaces, behavior, and common services. The meta- container is itself a service-oriented architecture. JBI components describe their capabilities through the Web Service Definition Language (WSDL).

JBI provides a lightweight messaging infrastructure, known as the **normalized message router**, to provide the mechanism for actual exchange of messages in a loosely-coupled fashion, always using the JBI implementation as an intermediary. The Normalized Message Router is “Backbone” of JBI that facilitates interoperation between JBI Components using WSDL-based service descriptors.

The plug-in components function as service providers, or service consumers, or both. Components that supply or consume services locally (within the JBI environment) are termed “service engines.” Components that provide or consume services via some sort of communications protocol or other remoting technology are called “binding components.” (See **Figure 3**). This distinction is important for various pragmatic reasons, but is secondary to the components’ roles as providers and consumers of services **(Mahmoud, 2005)**.

Service Engines and Binding Components are only logically and functionally different – technically both implement the same interfaces. Service Engines and Binding Components register the services they provide with the JBI framework using WSDL-based service descriptors **(Juric, 2006)**.

The BPEL Service Engine is a standard JBI Compliant component as defined by JSR 208. The BPEL Service Engine enables orchestration of WS-BPEL 2.0 business processes. This enables a work flow of different business services to be built **(Rosen, 2008)**.

The Java EE service engine acts as a bridge between the JBI container allowing Java EE web services to be consumed from within JBI components. Without the Java EE service Engine, JBI components would have to execute Java EE Web Services via remote calls instead of via in-process communication. The Java EE Service Engine allows both servlet and EJB-based web services to be consumed from within JBI components **(Rosen, 2008)**.

SQL service engine allows SQL statements to be executed against relational databases and allows the results of SQL statements to be returned to the client application or other Service Engines for further processing. SQL service engine allows SQL

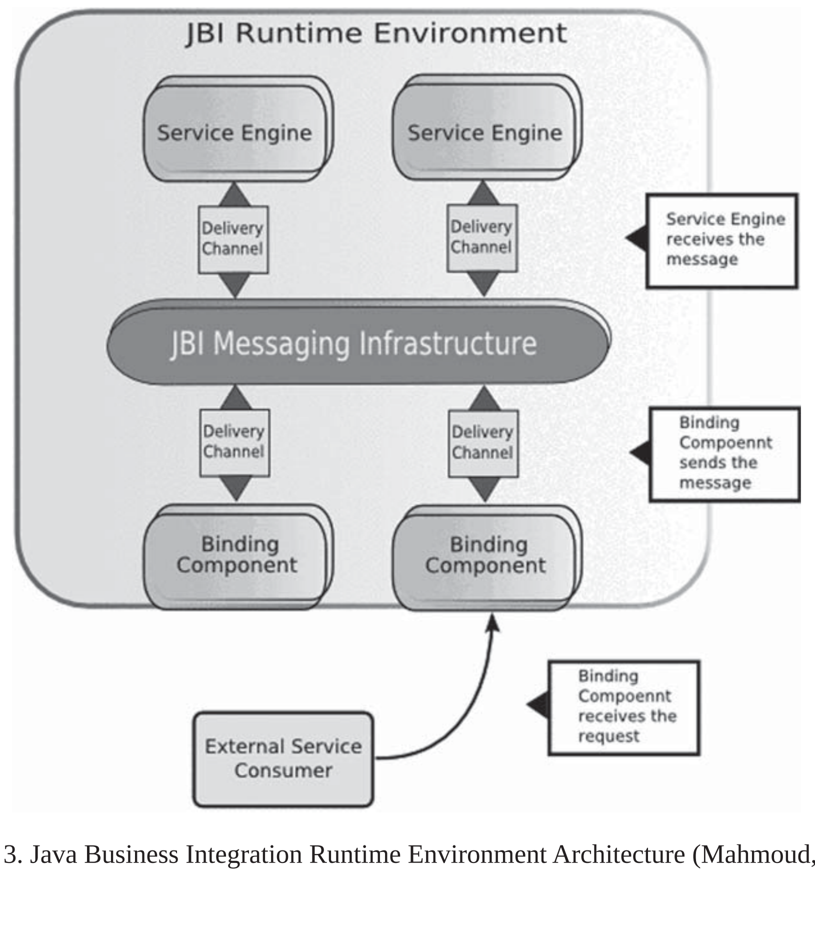

JBI Runtime Environment
Service Engine
Service Engine
Delivery Channel
Delivery Channel
Service Engine receives the message
JBI Messaging Infrastructure
Delivery Channel
Delivery Channel
Binding Compoennt sends the message
Binding Component
Binding Component
Binding Compoennt receives the request
External Service Consumer

Figure 3. Java Business Integration Runtime Environment Architecture (Mahmoud, 2005)

DDL (Data Definition Language), SQL DML (Data Manipulation Language), and stored procedures to be executed against a database. This, therefore, allows different scenarios to be executed against the database **(Rosen, 2008)**.

The Intelligent Event Processing service engine allows data to be read from an input source and then processed into a format that can be used for a variety of different purposes such as reporting or business intelligence information.

XSLT Service Engine enables transformations of XML documents from one format to another using XSL stylesheets. The service engine allows XSL transformations to be deployed as web services which can then be used by external clients.

Binding components are also standard JSR 208 components that plug in to NMR and provide transport independence to NMR and Service Engines. The role of binding components is to isolate communication protocols from JBI container so that Service Engines are completely decoupled from the communication infrastructure. For example, BPEL Service Engine can receive requests to initiate BPEL process while reading files on the local file system. It can receive these requests from SOAP messages, from a JMS message, or from any of the other binding components installed into JBI container **(Rosen, 2008)**.

The file binding component provides a communications mechanism for JBI components to interact with the file system. It can act as both a **Provider** by checking for new files to process, or as a **Consumer** by outputting files for other processes or components **(Rosen, 2008)**.

The SMTP Binding Component provides email services to the JBI Server and can act as either a provider by receiving inbound SMTP messages or as a consumer by sending SMTP email to external email addresses. The FTP binding component provides FTP transport services to the JBI container allowing messages to be received and sent via the FTP protocol. The SOAP binding component (also known as the HTTP binding component) allows JBI messages to be sent and received using SOAP over HTTP and HTTPS. The component supports RPC Literal, RPC Encoded, and Document Literal encoding schemes. The JDBC binding component can act as a provider or a consumer. When acting as a provider, the component can issue these different DML commands to the database either to select information from the database or to change data: Select, Insert, Update, and Delete. The JMS Binding Component allows the JBI container to communicate with JMS message queues and topics. The component can act as a provider and/or as a consumer of JMS messages, and as such can subscribe to a topic and wait for JMS messages, or it can send messages to a predefined Queue or Topic **(Rosen, 2008)**.

2.5. Database Binding Component

The Database BC helps users to handle databases with flexibility. It also provide Web Services in conjunction with other OpenESB components. The following components are part of the Database BC **(Anonim, 2008)**.

- TheWSDL fromDatabase wizard, which supports Table, Prepared Statements, Procedure,
- and SQL File (NetBeans plug-in).
- CustomWSDL extensions for configuring the Web Service (NetBeans plug-in).
- Database Binding Runtime component (JBI runtime component).

Database BC is a JBI runtime component that provides a comprehensive solution for configuring and connecting to databases. Database BC provides data operations as Services. It supports the JDBC from within a JBI environment. Other JBI components invoke these Web Services acting as consumers.Database BC considers bothData Manipulation Language (DML) and DataDefinition Language (DDL) operations as Web Services **(Anonim, 2008)**.

The services that theDatabase BC exposes are actually SQL operations on Table, Prepared Statements, and Procedures. TheDatabase BC supports the following database artifacts to be exposed as Services.

- Table
- Prepared Statements
- Procedures
- SQL File

The Database BC can assume the role of either a JBI consumer (polling inbound requests) or a JBI provider (sending outbound messages).

Database BC acts as a provider in case of outbound message flow.Database BC acts as an external service provider when other engines and components 'invoke' it. In this role, when it receives a normalized message as part of the message exchange, it converts and extracts the SQL operation. The SQL operation is then executed on the specified database. In other words, when theDatabase BC acts as a JBI provider, it extracts the SQL query from a JBI message received from the JBI framework.

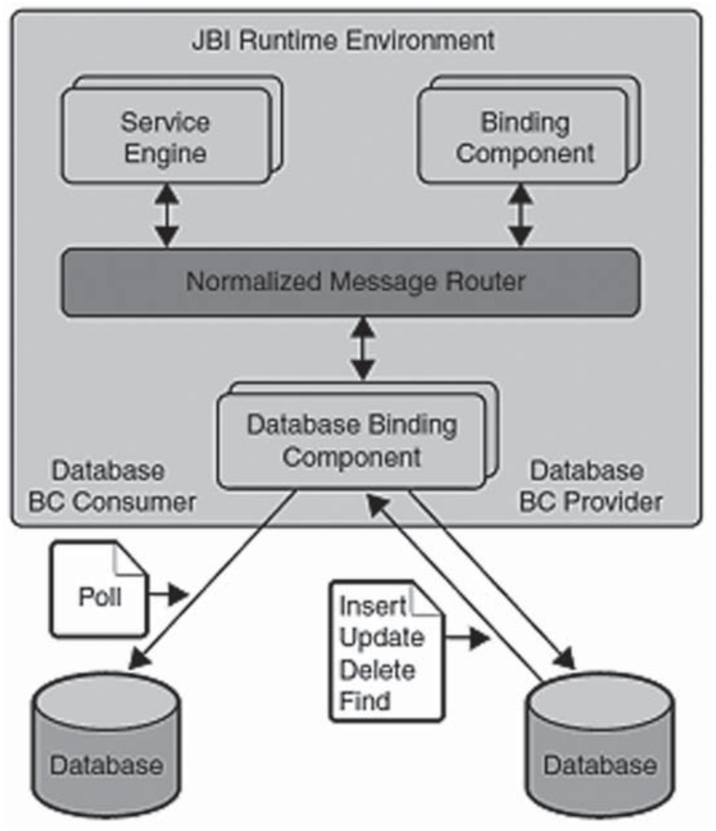


Figure 4. Database Binding Component of JBI (Anonim, 2008)

It then executes the query on a specified database. It converts the reply from the database into a JBI message that other JBI components can service **(Anonim, 2008)**.

Database BC also acts as a consumer incase of inbound functionality whereDatabase BC polls for records from a particular table, converts them into normalized message, and sends to the NormalizedMessage Router (NMR). This process is analogous to the inbound connections implemented in CAPS 6. In other words, When theDatabase BC acts as a JBI consumer, it polls a specified database for updates to a table in the database. When a new record is stored in the table, the database polls for the record for the specified time interval and the Database BC picks up that record, constructs a JBI message, and sends the message to the JBI framework so it can be serviced by other JBI components **(Anonim, 2008)**.

3. The Case Study

This section presents an example of how to perform the select table from a remote database, how to send the result from the select table to the file, and how to insert extracted records into the extract table on the local database. As our "remote" database, we will use the "sample" database from Java DB database installed with NetBeans. And as our local database, we will need to create our new DBLOCAL database. We will select a name and email from Customer table of remote database. The result of select table will be sent to the output.xml file. The result of select table also will be inserted into the EXTRACT-CUSTOMER table on the local database. (See figure 5).

This research uses these JBI components: the HTTP/SOAP Binding Component (simple RPC literal, request/response message exchange), the BPEL Service Engine (one BPEL process calling another BPEL process), File Binding Component (write operation overwriting previous contents of the output file) and the Database Binding Components (select and insert operations).

The activity satisfies the following process: 1) Create four web services description language (WSDL) (TriggerDBLocalWSDL, DBGlobalWSDL, DBLocalFileOut, and DBLocalWSDL), 2) Build a BPEL process to orchestrate the four WSDL created

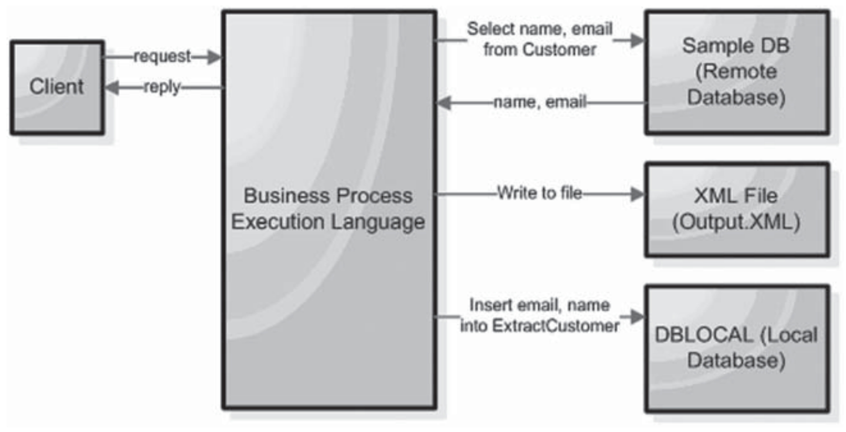
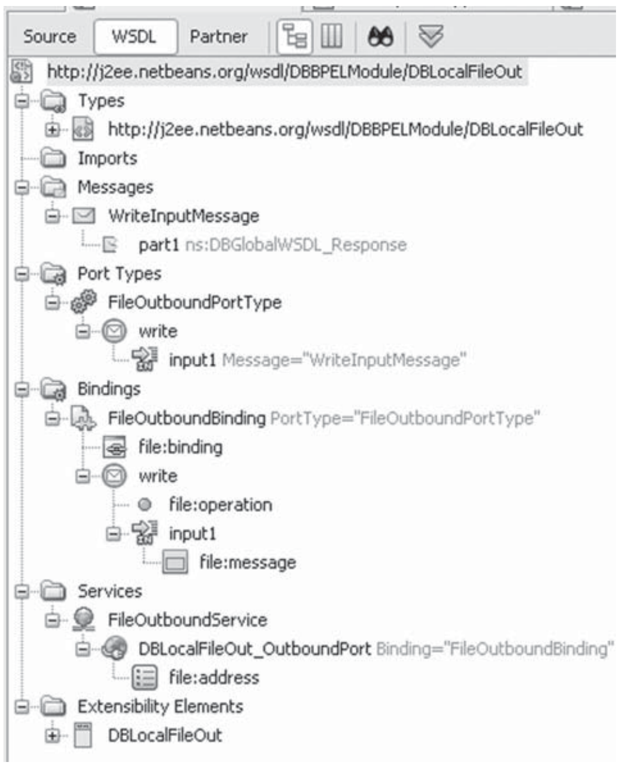

Client
request
reply
Business Process Execution Language
Select name, email from Customer
name, email
Sample DB (Remote Database)
Write to file
XML File (Output.XML)
Insert email, name into ExtractCustomer
DBLOCAL (Local Database)

Figure 5. Scenario of message flow from remote DB to XML File and local DB

earlier 3) Build a composite application containing our BPEL process and four WSDL, 4) Deploy application to application server (Glassfish 2.1), and 5) Test application.

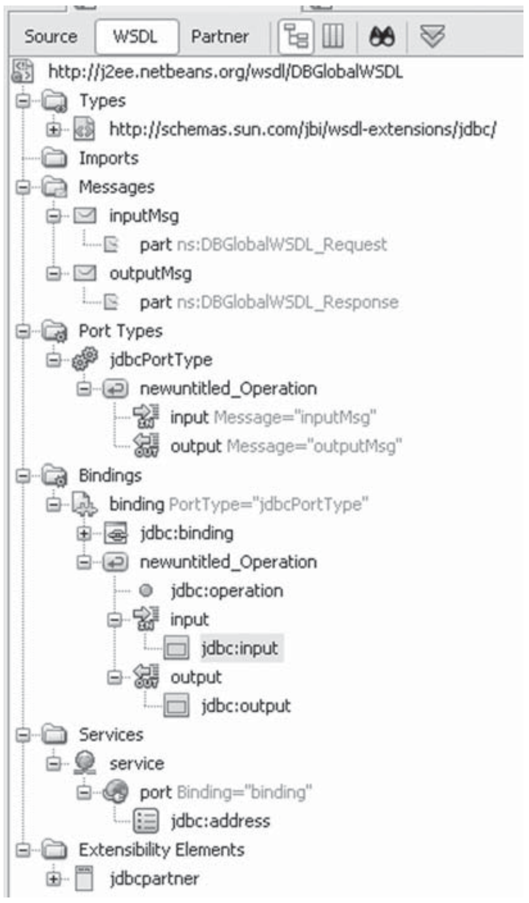
3.1. Create WSDL

First step is to create a new WSDL Document called **DBGlobalWSDL (Figure 6), DBLocalFileOut** (**Figure** 7), to describe the interface that will be used to write the file system. We need a WSDL to describe the interface and use it to insert the extracted records into the local database. Create a new WSDL called **DBLocalWSDL (Figure 8).** And finally, we need to be able to initiate our BPEL process. So, we need a WSDL to describe how to describe its interface. Create a new WSDL called **TriggerDBLocalWSDL.**



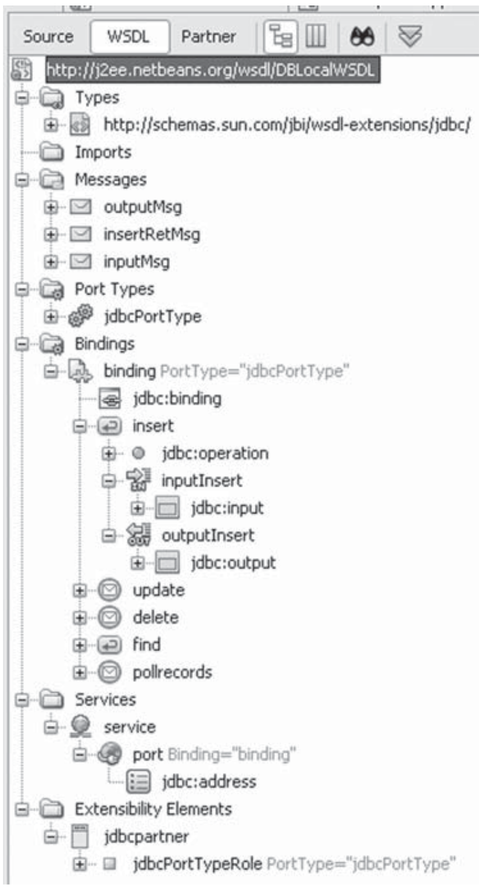
```
http://j2ee.netbeans.org/wsdl/DBBPELModule/DBLocalFileOut
  Types
    http://j2ee.netbeans.org/wsdl/DBBPELModule/DBLocalFileOut
  Imports
  Messages
    WriteInputMessage
      part1 ns:DBGlobalWSDL_Response
  Port Types
    FileOutboundPortType
      write
        input1 Message="WriteInputMessage"
  Bindings
    FileOutboundBinding PortType="FileOutboundPortType"
      file:binding
      write
        file:operation
        input1
          file:message
  Services
    FileOutboundService
      DBLocalFileOut_OutboundPort Binding="FileOutboundBinding"
        file:address
  Extensibility Elements
    DBLocalFileOut
```

Figure 6. DBGlobalWSDL that defines interface for select table into the database



```
http://j2ee.netbeans.org/wsdl/DBGlobalWSDL
  Types
    http://schemas.sun.com/jbi/wsdl-extensions/jdbc/
  Imports
  Messages
    inputMsg
      part ns:DBGlobalWSDL_Request
    outputMsg
      part ns:DBGlobalWSDL_Response
  Port Types
    jdbcPortType
      newuntitled_Operation
        input Message="inputMsg"
        output Message="outputMsg"
  Bindings
    binding PortType="jdbcPortType"
      jdbc:binding
      newuntitled_Operation
        jdbc:operation
        input
          jdbc:input
        output
          jdbc:output
  Services
    service
      port Binding="binding"
        jdbc:address
  Extensibility Elements
    jdbcpartner
```

Figure 7. DBLocalFileOut that describes the interface that will be used to write the file system



```
http://j2ee.netbeans.org/wsdl/DBLocalWSDL
  Types
    http://schemas.sun.com/jbi/wsdl-extensions/jdbc/
  Imports
  Messages
    outputMsg
    insertRetMsg
    inputMsg
  Port Types
    jdbcPortType
  Bindings
    binding PortType="jdbcPortType"
      jdbc:binding
      insert
        jdbc:operation
        inputInsert
          jdbc:input
        outputInsert
          jdbc:output
      update
      delete
      find
      pollrecords
  Services
    service
      port Binding="binding"
        jdbc:address
  Extensibility Elements
    jdbcpartner
      jdbcPortTypeRole PortType="jdbcPortType"
```

Figure 8. DBLocalWSDL that describes the interface that will be used to insert the extracted records into the local database

The WSDL document will be used to describe the Web Service interface of our BPEL orchestration process. The WSDL represents the “external” WS Interface for the whole orchestration we are going to compose. We used four WSDL documents with the following functions : 1) TriggerDBLocalWSDL to initiate our BPEL process, 2) DBGlobalWSDL to define interface for select table from the remote database, 3) DBLocalFileOut to describe the interface that will be used to write the file system, and 4) DBLocalWSDL to describe the interface that is used to insert the extracted records into the local database.

3.2. Composing services with BPEL

Composing services with BPEL process starts when TriggerDBLocalWSDL, a client that invokes its operation. If the client has been generated from the WSDL (TriggerDBLocalWSDL), we can then specify the operation invoked and the variables input. All operation data types are retreived from the process WSDL. A visual link is created in order to show the messages flow between parts (See **Figure 9**).

In this **figure 9** below, a request is made from the client to the JBI Container. This request is passed via NMR to the BPEL Service Engine. The BPEL Service Engine then makes a request to the Database Binding Component via NMR. The Database Binding Component returns a message to the File Binding Componet again via NMR. The File Binding Component then makes a request to the Database Binding Component via NMR. Finally the message is routed back to the client through NMR and JBI framework. The important concept here is that NMR is a message routing hub not only between clients and binding component, but also for intra-communication between different binding components. The entire architecture we have discussed is typically referred to as an Enterprise Service Bus.

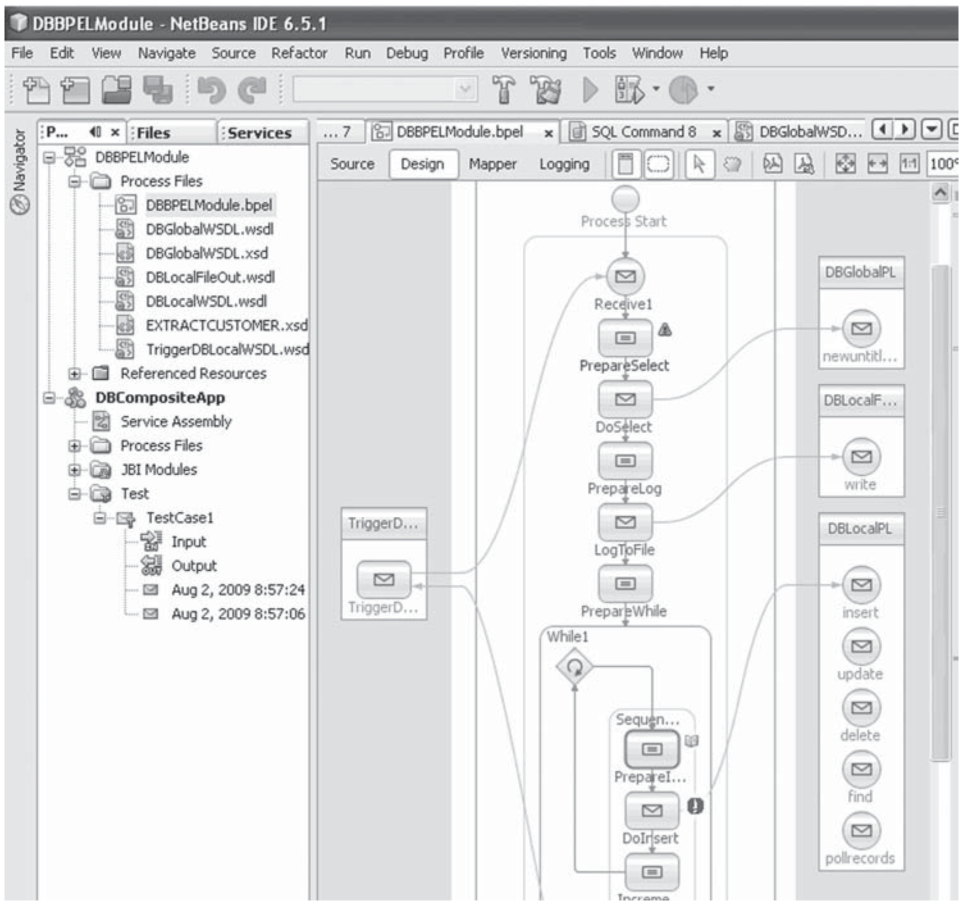


Figure 9. DBBPELModule to composing services

3.3. Composite Application

In order to be able to deploy the BPEL orchestration in the Application Server, we have to create a Composite Application project and add the BPEL application as a JBI module. Now, our SOA application is successfully deployed in the Application Server (See **Figure 10**).

Composite applications are both a form of integration as well as application development. The main objective of the composite application project system is to provide a deployment container for various types of JBI component projects.

In figure 10, the first pane of the editor lists all the WSDL Ports. The second pane shows all available JBI Modules. Any JBI module including BPEL modules, can be draged-and-dropped, in this space, Service Units can be added as part of other service assemblies in the third pane. Note that our composite application has four WSDL Ports. The WSDL ports are exposed through SOAP binding, File binding, and Database Binding.

The Composite Application project can be enhanced by adding test cases, binding the operation, supplying input, and then using the Tester. The test case shows that the result is a success. The test is passed, the execution flow has been performed correctly. (See **Figure 11**). Now we can see the SOAP response message coming from our SOA application.

Look in the output directory to find the output.xml file that has been created. Open the file to see the records that were read from the database via the select from a table (Figure 12).

After that, check the result from the test case and view the contents of the extract table (See figure 13). We see that the table was inserted successfully by the execution of our DBLocalWSDL.

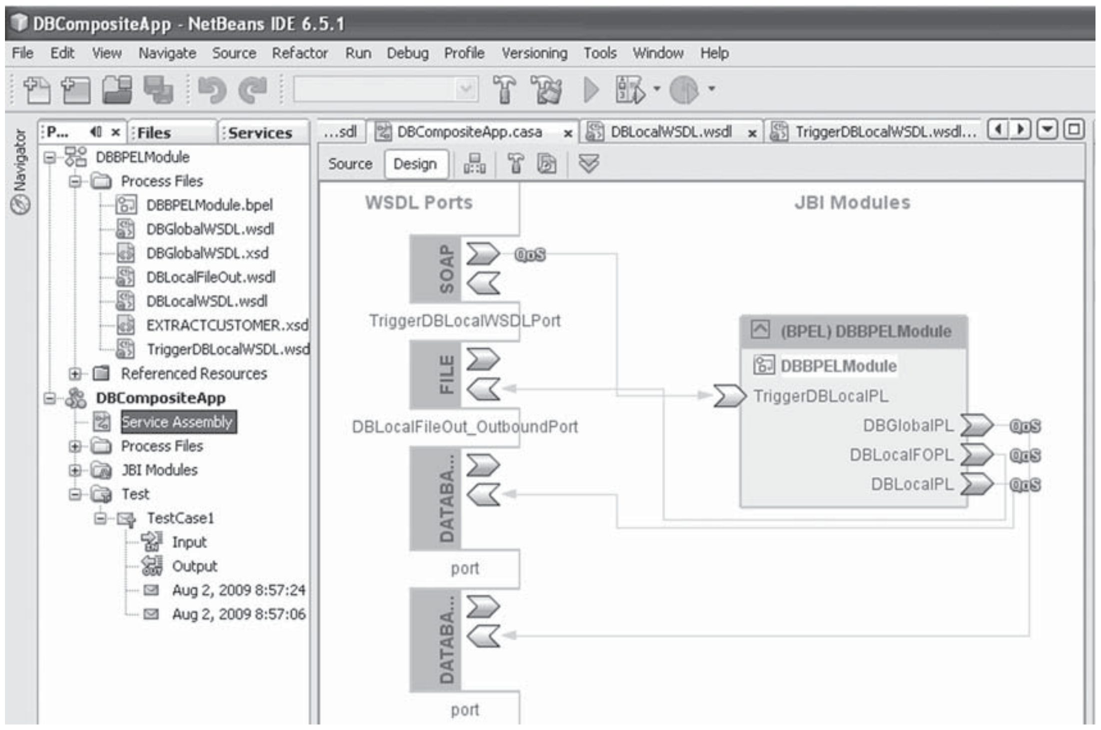
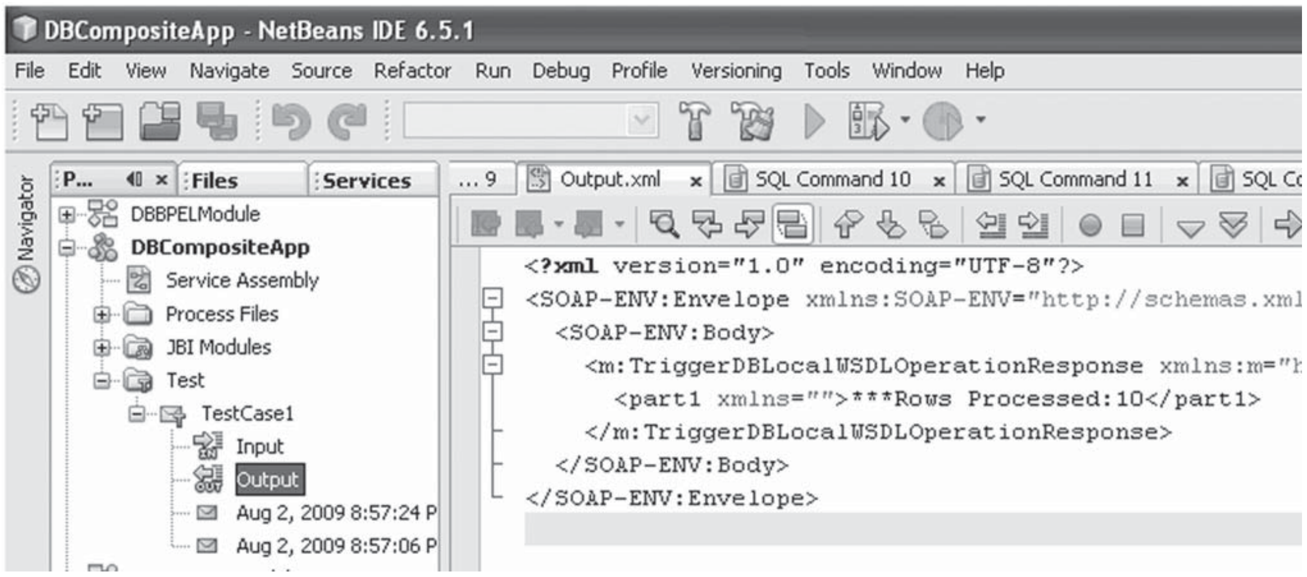


Figure 10. SOA Composite Application

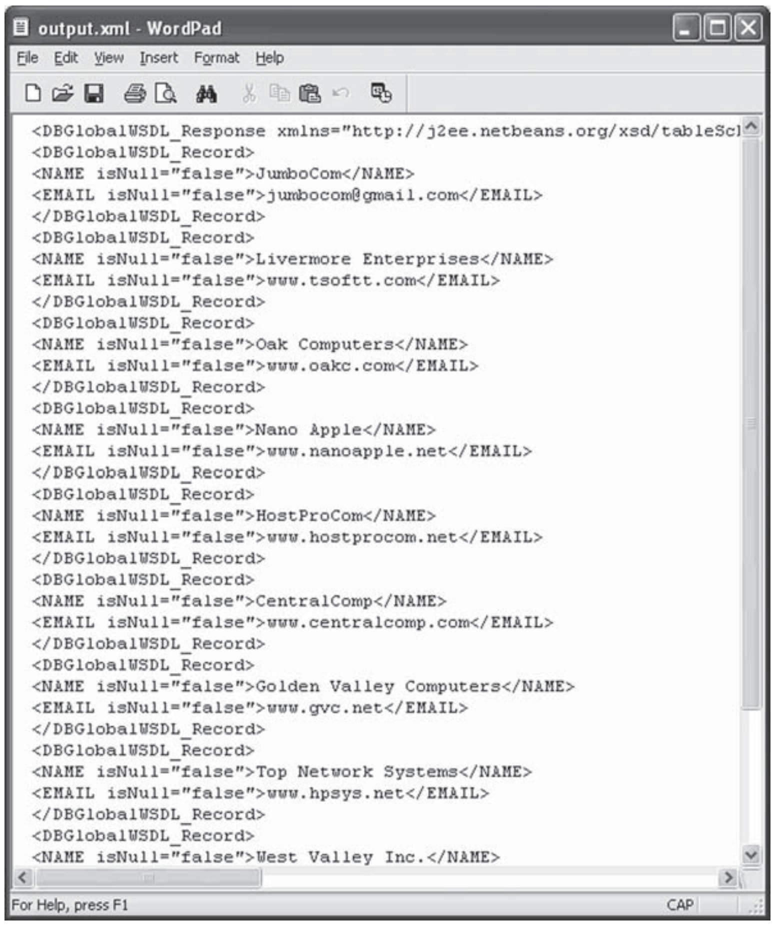

DBCompositeApp - NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Navigator
P... Files Services
DBBPELModule
DBCompositeApp
Service Assembly
Process Files
JBI Modules
Test
TestCase1
Input
Output
Aug 2, 2009 8:57:24 P
Aug 2, 2009 8:57:06 P
... 9 Output.xml SQL Command 10 SQL Command 11 SQL Co

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xml
  <SOAP-ENV:Body>
    <m:TriggerDBLocalWSDLOperationResponse xmlns:m="h
      <part1 xmlns="">***Rows Processed:10</part1>
    </m:TriggerDBLocalWSDLOperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 11. Test passed

3.3. Practical result

Construction of applications using a service-oriented architecture differs significantly from techniques used in other software architectures. SOA requires that the software developer create services (or integrate existing business functions as services) that are reusable in different applications, including the creation of new services. Each service is designed for future frequent reuse. Reuse is possible by decoupling the service from the consumer. Applications are composed from different services. SOA is being implemented on ESB infrastructure using extensible suites from vendors. Web services and BPEL are the key



```
<DBGlobalWSDL_Response xmlns="http://j2ee.netbeans.org/xsd/tableSc
<DBGlobalWSDL_Record>
<NAME isNull="false">JumboCom</NAME>
<EMAIL isNull="false">jumbocom@gmail.com</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">Livermore Enterprises</NAME>
<EMAIL isNull="false">www.tsoftt.com</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">Oak Computers</NAME>
<EMAIL isNull="false">www.oakc.com</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">Nano Apple</NAME>
<EMAIL isNull="false">www.nanoapple.net</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">HostProCom</NAME>
<EMAIL isNull="false">www.hostprocom.net</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">CentralComp</NAME>
<EMAIL isNull="false">www.centralcomp.com</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">Golden Valley Computers</NAME>
<EMAIL isNull="false">www.gvc.net</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">Top Network Systems</NAME>
<EMAIL isNull="false">www.hpsys.net</EMAIL>
</DBGlobalWSDL_Record>
<DBGlobalWSDL_Record>
<NAME isNull="false">West Valley Inc.</NAME>
```

Figure 12. The output.xml file is the result from the select table

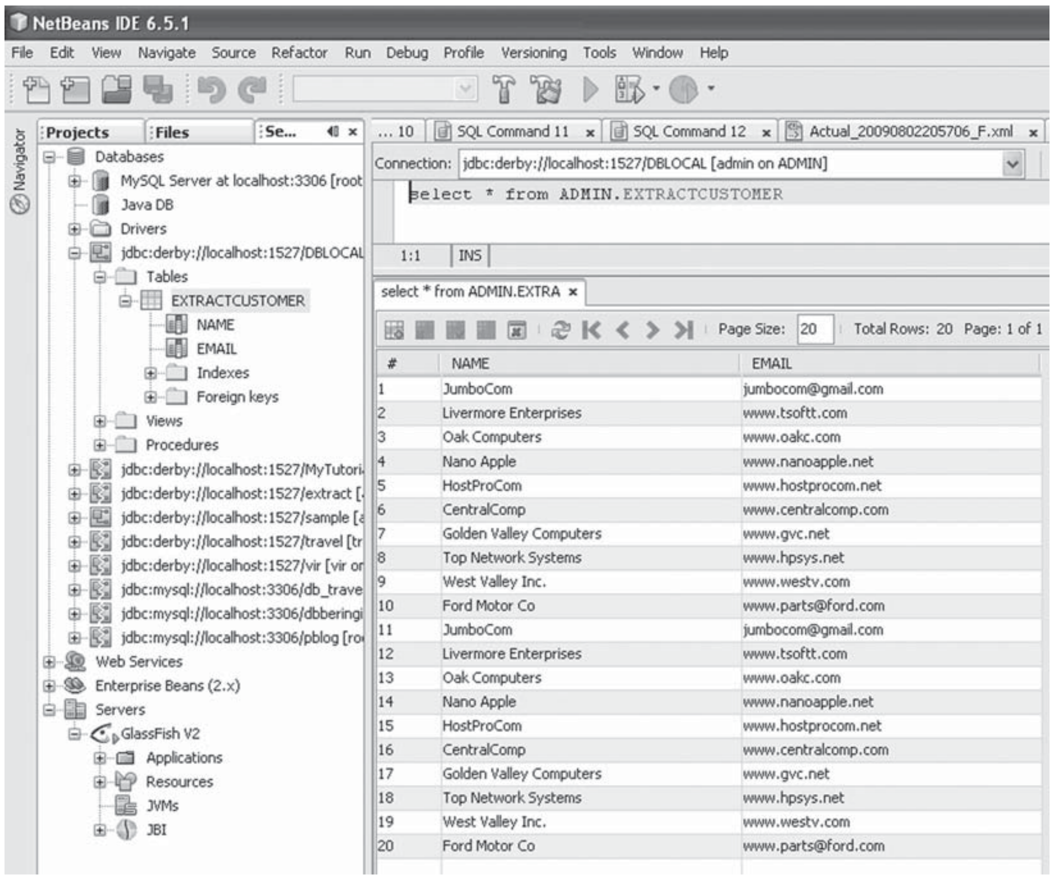
technologies of SOA. Web services use XML, WSDL, XSD and SOAP to provide interoperability and to provide service delivery. BPEL provides orchestration for composite applications that use web services. JBI provides a middleware standard for integration in the Java community and OpenESB is the reference implementation for JBI.

JBI provides integration of existing business functions as services. In this project research, the remote database is "wrapped" as a service using WSDL document called DBglobalWSDL. The local database is "wrapped" as a service using WSDL document called DBLocalWSDL, and the file XML is "wrapped" as a service using DBLocalFileOut. This WSDL will be used as the "external" WS Interface for the whole orchestration that we are going to compose. And we have seen that we have successfully extracted record table from remote database and written it to the XML file and inserted it to the local database So this research can be considered as a proof of concept of implementation for a feasible and efficient databases integration using Java Business Integration through web services orchestrations with BPEL.

Databases Integration is an entry point opened to other integations. Thus this research is an entry point to conduct intra-business as well as inter-business integrations. Various buisness integration protocols have been provided by JBI framework, inter alia Database, SOAP, JMS, File, SMTP, etc. The test case using framework JBI shows very satisfying results of the chosen approach.

Conclusion

Java Business Integration (JBI) provides a foundation for construction a SOA. It provides for integration of existing business functions as services, and decoupled interaction between service providers and consumers. It provides direct support for composite application creation through the mechanism of JBI service assemblies, which allow applications to be composed directly from the service-based interfaces of JBI service units.



Connection: jdbc:derby://localhost:1527/DBLOCAL [admin on ADMIN]

```
select * from ADMIN.EXTRACTCUSTOMER
```

select * from ADMIN.EXTRA

Page Size: 20 Total Rows: 20 Page: 1 of 1

#	NAME	EMAIL
1	JumboCom	jumbocom@gmail.com
2	Livermore Enterprises	www.tsoftt.com
3	Oak Computers	www.oakc.com
4	Nano Apple	www.nanoapple.net
5	HostProCom	www.hostprocom.net
6	CentralComp	www.centralcomp.com
7	Golden Valley Computers	www.gvc.net
8	Top Network Systems	www.hpsys.net
9	West Valley Inc.	www.westv.com
10	Ford Motor Co	www.parts@ford.com
11	JumboCom	jumbocom@gmail.com
12	Livermore Enterprises	www.tsoftt.com
13	Oak Computers	www.oakc.com
14	Nano Apple	www.nanoapple.net
15	HostProCom	www.hostprocom.net
16	CentralComp	www.centralcomp.com
17	Golden Valley Computers	www.gvc.net
18	Top Network Systems	www.hpsys.net
19	West Valley Inc.	www.westv.com
20	Ford Motor Co	www.parts@ford.com

Figure 13. The result from the select table inserted to the local database

Finally, we have extracted record table from remote database and written it to the XML file and inserted it to the local database in the Java Business Integration framework and we have composed the SOA Application through a Web Services Orchestration with BPEL (Business Process Execution Language).

References

[1] Hove, R. (2006). Using JBI for Service-Oriented Integration (SOI), Sun Microsystems Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. DOI = http://www.java.sun.com

[2] Hove, R., (2006). JBI Components: Part 1 (Theory), Sun Microsystems Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. DOI = http://www.java.sun.com

[3] Anonim, (2008). Understanding the Database Binding Component. Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A DOI = http://www.java.sun.com

[4] Mahmoud, Q.H., (2005). Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). 4150 Network Circle, Santa Clara, CA 95054 U.S.A. DOI = http://www.java.sun.com.

[5] Brown, P.C., (2008). Implementing SOA: Total Architecture in Practice, United States of America : Addison Wesley Professional.

[6] Juric, M. B., Mathew, B., and Sarang, P. (2006). Business Process Execution Language for Web Services, Birmingham-Mumbai: Packt Publishing.

[7] Rosen, M., Lublinsky, B., Smith, K., and Balcer, M., (2008). Applied SOA: Service-Oriented Architecture and Design Strategies, Indianapolis: Wiley Publishing, Inc.

[8] Salter, D. and Jennings, F., (2008). Building SOA-Based Composite Applications Using NetBeans IDE 6, Birmingham-Mumbai: Packt Publishing.