

1. Open Source J2EE Enterprise Service Bus Investigation

By Dr Ant Kutschera, Blue Infinity SA, Geneva, Switzerland.

1. Objective

The objective of this study is to specify the meaning of Enterprise Service Bus (ESB), to investigate what ESB products are in the open source market and to evaluate selected products to determine which could be recommended for greenfield software projects that require an ESB.

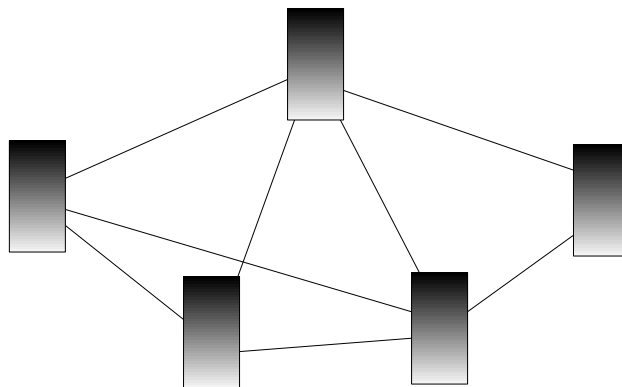
2. What is the Enterprise Service Bus?

The history of the ESB comes from the coming together of several previous technologies. These are discussed in chronological order.

a) Point to Point Integration

Point to point integration was the first methodology used to integrate multiple systems that needed to communicate. Typically it starts with two or three applications needing to share information. A technology to integrate them is chosen, for example CORBA or a messaging system, or indeed a low level Remote Procedure Call (RPC) style connectivity technology. However, the problem with such an integration methodology is that as the number of systems being integrated increases, the number of potential connections grows exponentially.

As the number of connections grows, the costs associated with developing them, the costs associated with maintaining them, as well as the additional complexity introduced to the landscape means that it becomes less favorable to use a point to point integration.



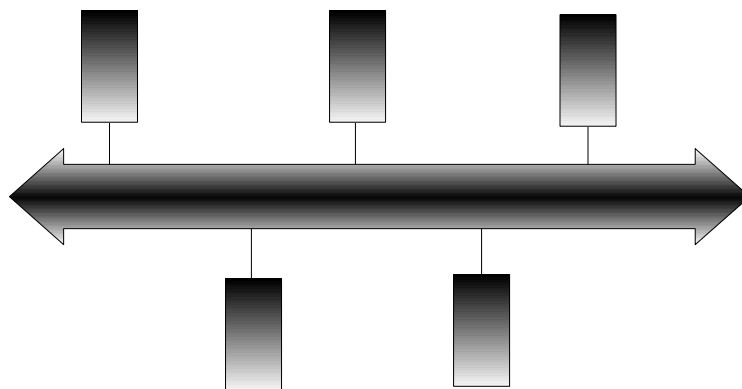
The figure above shows how systems connect to each other in a point to point architecture and that the number of connections could be as high as $(n^2 - n)/2$, although in reality, not all the systems need to connect to each other. In a typical enterprise organisation with tens or perhaps hundreds of systems requiring integration, the number of interfaces that are required soon becomes unmanageable.

The solution to these problems came in the form of Enterprise Application Integration (EAI).

b) Enterprise Application Integration

EAI attempts to achieve integration by reducing the number of connections between external

systems. It does this by introducing a 'Messaging Bus'. The messaging bus provides a single place where each system connects to. The interface between the bus and the external system formats the data into a common structure known as a pivot structure. While EAI theory might dictate that this pivot structure should encapsulate all possible data fields / types that can pass through the system, in practise, multiple pivot structures are required. This is because multiple structures allow decoupling of independent data / software systems. In relation to EAI, loose coupling is particularly desirable because typically each external system has a separate customer or department of responsibility. If these systems became unnecessarily coupled through a single common pivot structure, it would mean that modifying one interface would require testing and other efforts from all other systems that connect to EAI. Since there is no business reason for them to be related, it becomes rather political to justify why these extra efforts are required.



The figure above shows how the systems connect to the message bus, in an EAI architecture.

This solution has indeed been proven to save enterprises a lot of effort in terms of software development and maintenance because it inherently allows the re-use of existing and legacy systems. Depending upon the chosen EAI framework or product, integration of additional systems becomes quick, cheap and without too greater addition of complexity.

According to JBoss.org¹, “Traditional EAI stacks consist of: Business Process Monitoring, Integrated Development Environment, Human Workflow User Interface, Business Process Management, Connectors, Transaction Manager, Security, Application Container, Messaging Service, Metadata Repository, Naming and Directory Service, Distributed Computing Architecture.”

The problem with EAI is however that it relates mainly to integration, and relies heavily on the product / framework vendors to provide the technology (connectors) to connect these systems. Hence vendor lock-in is common.

To overcome these problems, two things have happened. First, the rise of XML and SOAP meant that a new open standard for connecting / messaging appeared. Furthermore, it could fit into existing communication protocols like HTTP, which is not typically blocked firewalls maintained by corporate systems administration.

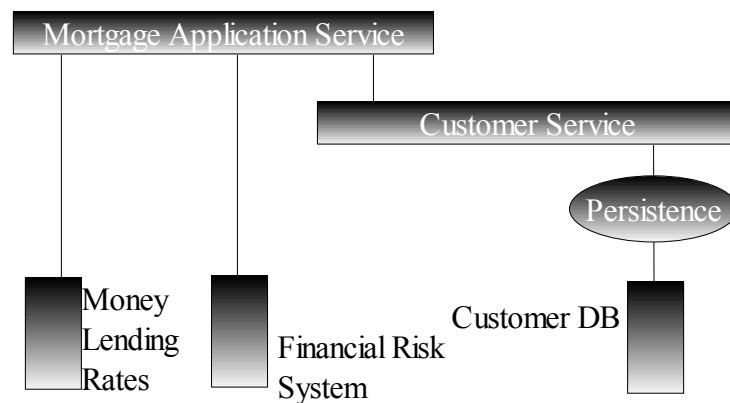
Second, there was a desire to move from technical integration solutions to higher level business process related solutions. This allows systems to be understood at a higher level and assist enterprise architecture because it becomes less difficult to understand all the interfaces that exist in a landscape. Finally, the age old adage of re-usability also pushed such a business process oriented architecture to the forefront.

All of these things lead to the rise of Service Oriented Architecture.

c) Service Oriented Architecture

SOA is about changing the traditional tight coupling between systems and partners into loose coupling. SOA exists at two levels. At the high level, it relates to business processes, and getting software to provide services related to these processes. Typically, a business process might relate to the processing of for example a mortgage application. A service related to this might be the 'Mortgage Application Service'. This service would expose methods such as submittal of the application, retrieval of a partially completed application, completion and acceptance of an application, etc. An associated service might be the Customer Service, used for adding new customers, retrieving customers, updating their information, closing their accounts, etc.

At a lower level, SOA relates to a software architecture whereby scalable components like J2EE EJBs can be wrapped with a SOAP Web Service, meaning that they can be easily called from any external system that has APIs allowing it to do so. It is built on open standards such as SOAP, and uses open protocols such as HTTP or SMTP. A SOA can also be implemented without the use of SOAP. Building an EJB layer in the form of a service layer is known to be a good design, and encourages re-usability and easy maintainability within the application.



The above figure shows how a typical SOA architecture might expose its services.

SOA does not necessarily have anything at all to do with integration. The author has completed many projects where SOA is used to provide a service layer within a single application, which is in turn used by a presentation layer. The implementation of the SOA in such cases helps to make the application maintainable and allows it to have a clear design which is easy for new project members to understand.

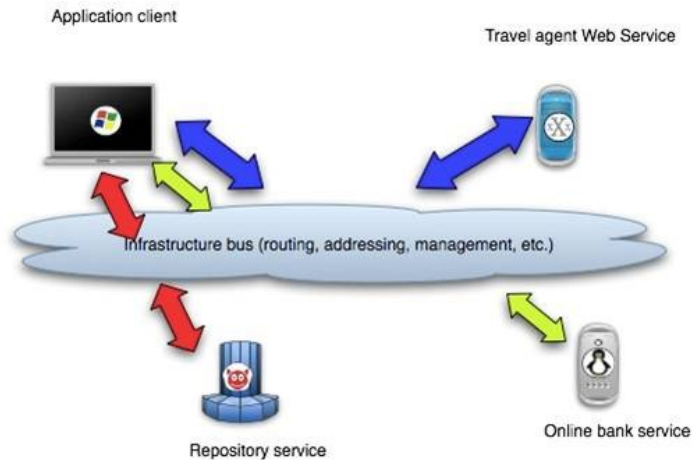
However, if considering SOA as a means of providing a facade that does integration behind the scenes, then the similarity between SOA and EAI begins to become clear. Both are exposing interfaces for outside systems to connect with. Whether the outside system is a presentation layer, or an ERP system, there is a similarity. However, one is about integration and the other is about business processes. So the Enterprise Service Bus (ESB) has evolved to combine them.

d) Service Oriented Integration

Service Oriented Integration (SOI) (reference 4) is the means of using an SOA to integrate a number of applications to create a *composite application*. As such, it could be seen as a stepping stone between SOA and ESB. However, this is something that has only been discussed recently, and in relation to the Sun JBI standard (JSR-208, reference 5) is effectively an ESB. Apart from this reference, there is little else in the domain, so while important, it will be skipped.

a) Enterprise Service Bus

According to JBoss.org¹, “The ESB is seen as the next generation of EAI – better and without the vendor lock-in characteristics of old”. As an example, consider this figure from the JBoss website:



Compared to traditional EAI, the message bus has become an infrastructure bus. Looking at this critically though, the only difference is that the ESB is more focused on the use of Web Services for its implementation, and it potentially provides a registry for the discovery of its services. With EAI, each interface was focused at a particular external system. With ESB, each interface is focused at a particular service, or business process.

Furthermore, there are two kinds of ESB, depending upon what is chosen as the central core of the ESB. The first is called a message-centric ESB and is one that focuses on integration via messaging. The second is called a service-centric ESB and is one that focuses on integration via the implementation of services. For this latter ESB type, the underlying implementation is not Messaging Oriented Middleware (MOM) dependent like the former, and in fact can sit on top of a number of independent communication protocols such as JMS, HTTP or others. Further information about EAI/MOM/Application Servers/ESBs can be found in the white paper in reference 2.

Finally, a definition of an ESB needs to be chosen. From reference 3, a good definition that extends the Gartner definition is chosen:

According to Gartner's definition, an ESB is standards-based middleware that uses a Service-Oriented Architecture (SOA) and that has messaging, intelligent routing, and transformation capabilities. Some industry experts validly extend that definition to include features like orchestration, security federation, and a common management framework.

1.2. Product Search

Now that an ESB has been defined, the next phase of this paper is to search for suitable open source, J2EE products which can be compared to each other.

This was done using Google. A simple set of searches for combinations of terms such as 'ESB', 'Open Source' and 'Enterprise Service Bus' was conducted. Furthermore, known open source product vendors like JBoss were included in the search.

The results were as follows, in no particular order:

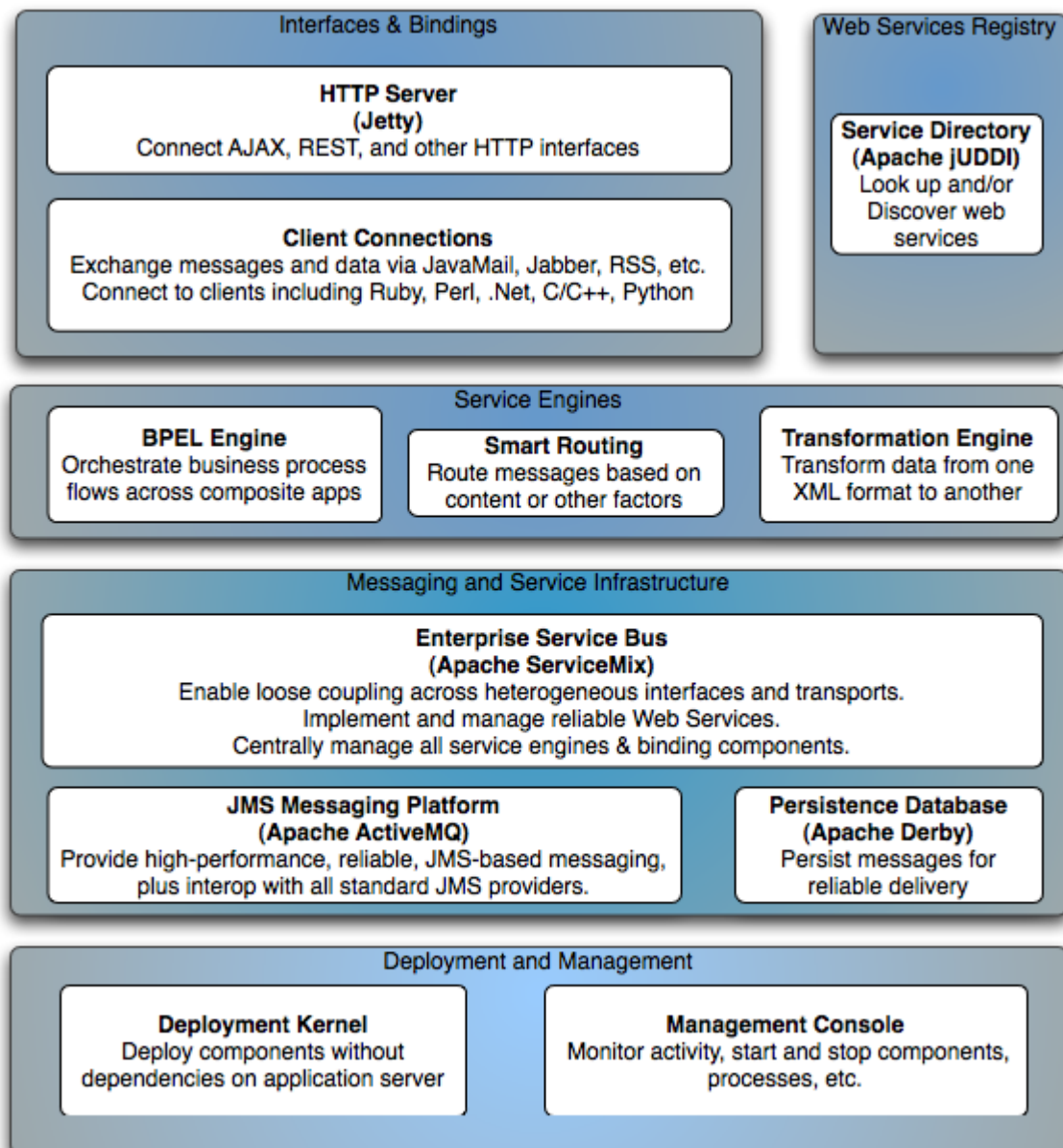
- 1) FUSE from LogicBlaze: <http://www.logicblaze.com>
- 2) Synapse from Apache: <http://wiki.apache.org/incubator/SynapseProposal>
- 3) Celtix from Iona: <http://www.ionac.com/opensource/>

- 4) JBossESB from JBoss.org: <http://labs.jboss.com/portal/jbossesb/>
- 5) OpenESB from java.net: <https://open-esb.dev.java.net/>
- 6) HomeRolled from Ant (the Author)

a) **FUSE from LogicBlaze**

This product is a collection of Apache open source products. The ESB is called Apache Service Mix and is an implementation of the JBI standard from Sun. It can run on its own, or on any J2EE Application Server. During the proof of concept, it was tested as a stand alone application. It also uses a product called Apache ActiveMQ as its message queue which is an implementation of JMS1.1. While the product is composed of a number of separate products, the idea is that the vendor, LogicBlaze certifies that the delivered versions all work together and that all the documentation is complete for a given version of FUSE. The company also sells support services and consulting which is how it makes its money. This could be useful for potential clients who implement a FUSE solution.

The documentation shows the architecture as follows.



The diagram above shows how all of the components in the FUSE product fit together.

Apache ServiceMix is the Apache implementation of the JBI standard from Sun (JSR208).

b) Synapse from Apache

<to be done>

b) Celtix from IONA

<to be done>

c) JbossESB from JBoss

<to be done>

d) OpenESB from java.net

<to be done>

e) HomeRolled from Ant

<to be done>

1.2. Software Selection

In order to select which of the products found in the product search can be recommended for use by consulting clients, a set of selection criteria need to be defined.

c) Selection Criteria and Weightings

See the selection matrix spread sheet. It includes all the criteria and weighting used to evaluate the products.

1.3. Product Evaluation

Selection criteria have been defined in the spreadsheet, but a way to grade each product still requires definition. A 'Proof of Concept' (PoC) style project that has an amount of scope that is fair to all products being evaluated, will allow such grading to be done. Based on previous experience of defining, implementing, managing, and interpreting the results of PoCs, the author has developed the following requirements for this ESB PoC.

a) PoC Requirements

- 1) The system will be based on an business process similar to that used with ordering products from a vendor with an online sales channel.
- 2) The system will have a web service as an entry point. The web service will use the WSDL provided below.
- 3) The web service will take the customer number, product codes and relative quantities as inputs. It will return a unique order ID.
- 4) The system will populate the order with the customers credit limit and the total value of the order.
- 5) If the customer has more credit limit that the total order value, and the total order value is lower than a configurable threshold, then it will be output by the sytem as an XML file, conforming to the XSD schema below.
- 6) If the customer has no credit limit or the total order value is higher than the configurable

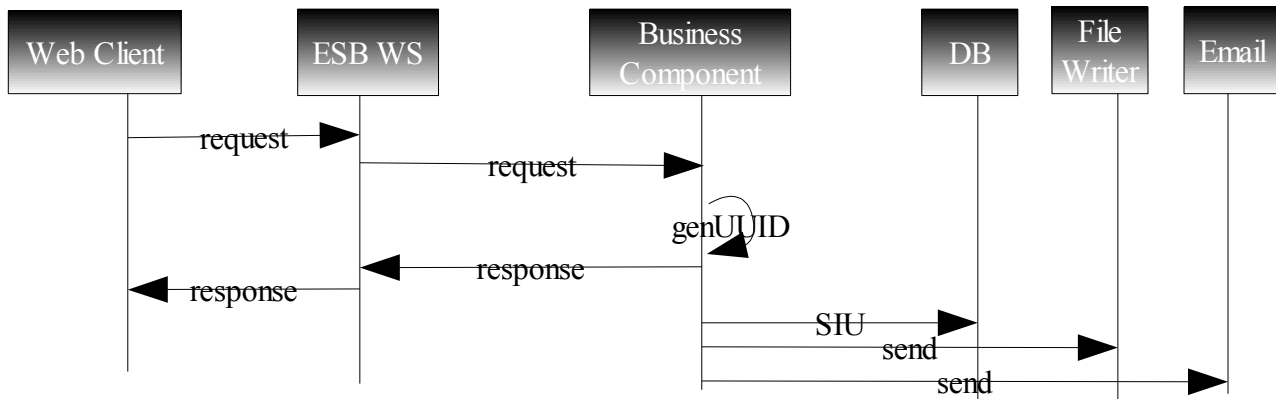
limit, then an email will be sent to a manager, informing them that the order needs authorization based on the customers order history (the fulfillment of this authorization is then a different business process which is out of scope for this PoC).

- 7) Customer credit details and item details (e.g. prices) are stored in a database, as defined with the following MySQL DDL code.

These requirements ensure that a business process of medium complexity is implemented, which includes integration with a web service, email, a database and a file system.

As the objective of this PoC is not to test the web server functionality of the ESB (since by the definition being used in this project, an ESB product may not include a web server or presentation layer), a separate extremely simple website will be built that implements a hard coded product lookup, hard coded customer login, and includes a Web Service client that allows it to send the order request to the ESB. This Web Service client will be generated against a standard WSDL document that all ESBs being evaluated must use in exposing their Web Service.

These requirements can be drawn with a sequence diagram as follows.



The WSDL describing the web service entry point into the ESB is as follows.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/OrdersWebService.jws"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://localhost:8080/axis/OrdersWebService.jws"
xmlns:intf="http://localhost:8080/axis/OrdersWebService.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
<wsdl:types>
<schema targetNamespace="http://localhost:8080/axis/OrdersWebService.jws"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="ArrayOf_xsd_string">
<complexContent>
<restriction base="soapenc:Array">

```

```

    <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
  </restriction>
</complexContent>
</complexType>
<complexType name="ArrayOf_xsd_int">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]"/>
    </restriction>
  </complexContent>
</complexType>
</schema>
</wsdl:types>

<wsdl:message name="createOrderResponse">
  <wsdl:part name="createOrderReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="createOrderRequest">
  <wsdl:part name="username" type="xsd:string"/>
  <wsdl:part name="password" type="xsd:string"/>
  <wsdl:part name="customerID" type="xsd:string"/>
  <wsdl:part name="productNumbers" type="impl:ArrayOf_xsd_string"/>
  <wsdl:part name="quantities" type="impl:ArrayOf_xsd_int"/>
</wsdl:message>

<wsdl:portType name="OrdersWebService">
  <wsdl:operation name="createOrder" parameterOrder="username password
customerID productNumbers quantities">
    <wsdl:input message="impl:createOrderRequest" name="createOrderRequest"/>
    <wsdl:output message="impl:createOrderResponse"
name="createOrderResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="OrdersWebServiceSoapBinding" type="impl:OrdersWebService">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

```



```

<wsdl:operation name="createOrder">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createOrderRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="createOrderResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/OrdersWebService.jws" use="encoded"/>
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="OrdersWebServiceService">
    <wsdl:port binding="impl:OrdersWebServiceSoapBinding"
name="OrdersWebService">
        <wsdlsoap:address location="http://localhost:8080/axis/OrdersWebService.jws"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

This WSDL maps to a Java equivalent method signature of

```

public String createOrder(    String username,
                             String password,
                             String customerID,
                             String[] productNumbers,
                             int[] quantities
                             ) throws Exception

```

where the username and password are checked against hard coded values of *affiliate_holland / as32l23f* or *affiliate_suisse / asdf*. The WS implemented in the ESB needs to take this into consideration. In the real world, this WS would be hosted over SSL to secure it, and the credentials would come from a database or directory server. In this PoC it is acceptable for them to be hard coded.

The XSD schema describing the output file containing the order is as follows.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="items" type="Items"/>
    <xsd:element name="customer" type="Customer"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
  <xsd:attribute name="uuid" type="xsd:string"/>
  <xsd:attribute name="totalValue" type="xsd:decimal"/>
</xsd:complexType>

<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productCode" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="price" type="xsd:decimal"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Customer">
  <xsd:attribute name="customerNumber" type="xsd:string"/>
  <xsd:attribute name="availableCredit" type="xsd:decimal"/>
</xsd:complexType>

</xsd:schema>

```

The DDL describing the database schema is as follows.

```

drop database esbproject_poc;
create database esbproject_poc;

use esbproject_poc;

create table if not exists items (
  item_number varchar(50) not null,
  unit_price decimal not null,
  primary key (item_number)
) engine=InnoDB default charset=latin1;

```

```
insert into items values ("0354938443", 12.99);
insert into items values ("58934289734232", 11.99);
insert into items values ("490834324R2355", 10.99);
```

```
create table if not exists customers (
  customer_number varchar(50) not null,
  name varchar(50) not null,
  available_credit decimal not null,
  primary key (customer_number)
) engine=InnoDB default charset=latin1;
```

```
insert into customers values ("44933875", "good credit man", 800.0);
insert into customers values ("47339543", "bad credit man", 100.0);
```

```
create table if not exists orders (
  order_number integer not null auto_increment,
  uuid varchar(50) not null,
  status varchar(10) not null, -- one of placed or pending (means waiting for authorisation)
  primary key (order_number)
) engine=InnoDB default charset=latin1;
```

```
insert into orders values (null, "1c1b50d9-a387-4813-9d78-233aff43daa4", "placed");
insert into orders values (null, "1c1b50d9-a387-4813-9d78-2b5aae8e15f8", "pending");
```

```
create table if not exists products_in_order (
  order_number integer not null,
  item_number varchar(50) not null,
  quantity integer not null,
  foreign key (order_number) references orders(order_number),
  foreign key (item_number) references items(item_number)
) engine=InnoDB default charset=latin1;
```

```
insert into products_in_order values (1, "0354938443", 2);
insert into products_in_order values (2, "490834324R2355", 120);
insert into products_in_order values (2, "58934289734232", 334);
```

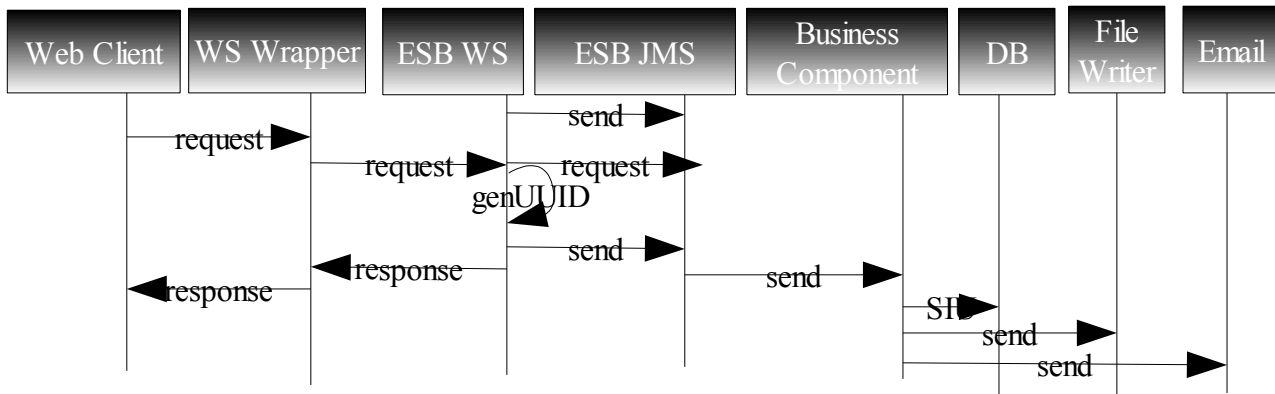
This DDL shows that indeed the product information is contained in the DB. However, the web site that is implemented as the starting point for executing the business process uses hard coded values because of the narrow time constraints of the project.

1.3. LogicBlaze FUSE Evaluation

The following describes the evaluation of the LogicBlaze FUSE product.

Implementation

The ideal flow could not be implemented because of restrictions in this ESB product. However, the following business process was implemented which follows the required business process quite accurately.



The flow above shows the business process that was implemented. A slight modification was required because the product did not appear to allow the generation of a WS entry point into the ESB from a WSDL document. As such, a WS wrapper was created to provide the required WSDL interface into the ESB. The wrapper simply implemented the WSDL, and called through to the ESB WS which was a simple WS that accepted a string and returned a string. The WS in the ESB generated the UUID as required, then passed the information on for processing while returning the UUID in the original WSDL response string.

The business process requirements do not specifically mention that an asynchronous process is required, and initially none was to be implemented. However, the ESB WS component taken from the examples in LogicBlaze did not allow the data from the call to be sent to another component in the ESB directly. So a manual JMS API connection was made to send the data to a JMS end point that had been specified in the ESB. From there, the Business Component subscribed to the queue, and upon receiving data it would complete the population of the Pivot Structure (matching the XSD), and depending upon the order value and the customer credit limit would either write the outgoing order file, or send an email. Another slight modification was made here. Ideally, the email and file components would have been out-of-the-box components supplied by the product and implemented as suggested in their documentation. However, the first problem was that although dynamic routing of an outbound message is apparently possible with ServiceMix, the author could not determine how to do this in the time allowed for this PoC. To get around this, the proposed design allowed the business component to send the email but for the outbound message to be processed by another component that writes the file to the disk. The second problem was that the subscription of this file writing component to the business component could not be established, even though it was copied from an example that was tested and at that time worked. It seems to relate to either the originating JMS starting point of the flow, or the fact that perhaps ServiceMix does not allow more than two components in a flow (perhaps in turn related to some transaction attributes of flows which the author did not understand or know about). The author went to the extent of debugging into ServiceMix, which suggested that although the business component was processing the data, that the JMS component still owned the routing of it, and hence the subscriber of the business component was not in the correct context. However, the author validated the configuration and could not determine where the problem lay. The work around was to add the file writing component to the business component, and to manually publish the data to a file, by calling the API of the file writing component.

While the implementation did not copy exactly the suggested implementation, it did provide an implementation of the business process that mimicked the required processing exactly. An end to end scenario was implemented in 5 days, the allowed time.

Problems

There were a number of problems encountered during this PoC, in addition to those listed above that required modifications to the required business process.

The first was a complete lack of documentation. Not only was there a shortage, but what was supplied did not provide enough details. Furthermore, there were discrepancies between the ServiceMix examples documentation and the FUSE examples documentation. It seems that LogicBlaze has taken a release of ServiceMix and modified it and packaged it, and that this deliverable differs slightly from the ServiceMix implementation. Additionally, the version they used is not a stable release of Service Mix (it claims to be version 3.0, but that version is not in stable release yet), and so they have potentially used a nightly build that is no longer compatible with the documentation on the ServiceMix website. The only documentation that is provided by LogicBlaze is for FUSE, and as such there is no documentation for any of the connectors. Another problem of this incompatibility is that it is not possible to get the correct version of the source code of this open source product! Nightly build history is not maintained on the ServiceMix web site and only named releases are available. So debugging into ServiceMix is effectively only 90% possible. Occasionally classes are encountered where the source code does not match what the JVM expects (the debugger goes to blank lines or wrong methods because the debug information in the classes is out of sync with the source code).

A further problem was in the JMS queue that sat between the WS and the business component. Randomly (but more than 50% of the time), the subscriber failed to get the message from the queue, complaining that the connection had been closed. As well as this problem, the message was then lost. This is extremely bad news! JMS is supposed to be reliable and provide guaranteed messaging. Although the queue was not persistent to a database or other genuinely persistent media, it was persistent to the in memory Derby database. This means that the message should not be removed from the queue until it is committed to the business component. No further configuration details could be found that might make this behavior change.

When using the Spring API to send email, it failed to set the list of recipients correctly, even through the Spring documentation was followed exactly. The method *setTo()* did not set the *to* header in the email, and so when it was sent to the mail server, it was rejected with a 503 error code. To get around this, the author manually set the recipient header in the email, and then it worked fine. The root cause of this problem was not determined.

No documentation could be found to show how to set up dynamic or content based routing of messages, although the marketing documents did suggest it was possible. This type of routing is important in real world scenarios because depending upon the business logic, different binding components might need to process data. In our example, dynamic routing would have been used to send email or write a file from the business component. Without it, it meant that the business component needed to contain the binding component for sending one of the outputs, since without dynamic routing, only one output can be sent from a component. Furthermore, the business method in the interface of business components in ServiceMix only allows for one input and one output message. In the authors experience, this appears to be a bit limiting, where the author has used other products that allow multiple inputs and outputs. However, with clever designs this is not as great an issue as it first appears. The only question is whether the product really does offer dynamic routing, and if so, how to implement it.

Creating the subscription of the file writing component to the business component did not work. Although another example with just a file polling component, transforming component and file writing component worked fine, integrating the file writer into this flow did not work. When

debugging into ServiceMix it appeared as though the JMS component still owned the routing context, and it correctly thought that the message had to be routed to the business component. However the author was debugging after the business components code had been executed, and as such expected that ServiceMix would have then checked for the next subscription in the data flow, but no such additional subscription was found by ServiceMix. It was as though ServiceMix could only deal with one subscription, and because we had already subscribed to the JMS component, the data could not be sent further than its subscriber (the business component). No solution was found, so the file writer had to be brought into the business component. This required some manual manipulation of the component. Effectively, it was added as a new Spring Bean into the business component. However, its context was then not initialized correctly and it threw errors after writing the file, complaining that it could not complete the transaction. However, it did not then delete the file as one might expect, suggesting that the file binding components do not offer a two phase commit or XA compliant transaction ability. This could also concern an architect using the product...

Just above, it was mentioned about Spring Beans. Spring is a framework (see reference 6) that creates a runtime environment that is configured through XML. The framework takes care of initializing the components from that configuration. It does this via the powerful Java Beans framework specified by Sun. For example, a configuration might say that an email component should be made available. The configuration would specify the connection details, as well as the class to initialize with those details. The class would provide accessor methods so that the framework can do the initialization. Then, this component would be referenced by a business logic component, that would itself contain a reference to this email component (bean) which the framework would set after the initialization.

Spring is used by ServiceMix as a quicker way to get started and to implement because it is more simple and more flexible than the plain JBI standard. However, the marketing documentation of an ESB complying with the JBI specification is that it reduces the vendor lock in that traditional EAI platforms forced. The problem is, that unless all other JBI compliant ESBs also allow Spring configuration of components, ServiceMix implementation using Spring, force vendor lock in! This is a problem for architects and project managers to consider if using FUSE / ServiceMix, as it also affects the maintainability in terms of getting resources who have the JBI skills as well as Spring skills.

There are no development GUIs provided with ServiceMix, so everything is developed by writing XML configuration manually. Eclipse was used as the IDE as it has a good Java editor, helps with building the environment, and also has a good XML editor. The effect of this lack of development GUIs affects the productivity of the developer. Firstly, development is slower, because the coder has to manually write the XML files. Secondly, the accuracy of the configuration is reduced. The configuration is partially checked against an XSD schema at deployment time, but that does not actually check the configuration completely. It just ensures the syntax of the configuration. Errors like invalid service names are still possible, and these are not discovered until runtime. So the effect is that to test the configuration, the developer has to deploy and run the components.

There is a deployment and management console in the form of a web application, however it is quite basic – it allows you to see what components are up and running, and lets you view the queues and topics in ActiveMQ. If messages have not been consumed, you can even view them in this console, which can be useful for debugging. However, this console was a bit flaky and contained bugs. For example, to deploy a new component, you could select the JAR file containing it, but the button to deploy did nothing – the author had to hit the return key to submit the form. Things like this make the console painful to use and are demoralizing. Additionally, it takes a patient and experienced developer to figure out the workarounds for such problems.

There were times during the development and testing of components where data /messages were effectively lost from the ESB. Sometimes that was because the author did not include good error handling in the components. Sometimes it happened when errors occurred outside of the scope where the developers components code was running, meaning that it was not possible to add error handling at this point (see earlier, in the discussion about JMS errors). These problems lead to the conclusion that ServiceMix does not include any standard error handling patterns, and any error handling needs to be implemented manually by the developer. It is recommended that a suitable error handling framework be defined by the architect of any project using ServiceMix, and that its use and adherence be monitored during code reviews. Some form of externally configurable error handling framework with the ability to send emails, write errors to a database or store messages on dead letter queues is desirable in the authors experience.

The next problem the author experienced was a lack of transaction support in the framework. If for example a binding component fails to send a message out, then it is desirable that it the message is rolled back and that the ESB try to process the data again later. The problem is that it seems that ServiceMix does not have such functionality. Another good example is that in the business component, if the DB fails and goes down, then the business component should rollback and start again later. But this does not happen. The error can be caught within the business component, and for example, an error notification (email) can be sent containing the problem, the original message and how to fix it. But it is not possible to then just rollback that message within the ESB to be automatically processed later. For idempotent data, this scenario is not so important, but where data can only be processed once (for example it only contains the differences between transactions and not the absolute values), or where the sequence of data is imperative, for example because of foreign key constraints in the target system, such rollback functionality is extremely desirable. Products like SeeBeyond eGate, SAP XI and Microsoft Biztalk all offer SOA / ESB frameworks, and all have the option that if the business logic fails, the data is rolled back without being lost. This seems to be something in ServiceMix that is missing, and depending upon data requirements could be quite significant in the product selection.

The last problem that the author found was that ServiceMix components seem to contain Marshal objects which one assumes allow the incoming data to be marshalled and unmarshalled into predefined objects so that they can be easily manipulated at development time. However, no example of this could be found. Instead, the PoC used JAXB (Java Architecture for XML Binding) to generate the classes for the data structure (pivot structure) from the XSD. These classes were then used to manually marshal and unmarshal the XML as it was passed through the ESB.

Implementation and Build

To understand what was implemented and how to do a build, follow these instructions.

There are several deployments. The first is the serviceMix deployment. This consists of a number of components. These are all located in "*open source ESB\logicblaze\FUSE-1.1\FUSE-1.1_ant\demos\poc*". There is an Eclipse project file here. Create a new project in Eclipse and specify this directory, and it will automatically read that file.

The directories are as follows:

httpserviceunit_jar – not used. Contained an example of the HTTP connector.

jdbcserviceunit_jar – contains the business component.

poc_assembly – used to assemble all the other components when building a service assembly (deployment unit)

pollserviceunit_jar – not used. Contained an example for polling a databases at a given interval.

Resources – contains DDL for creating the database and doing an initial load. Also contains a ServiceMix JNDI configuration that includes the database URL. This has already been deployed to ServiceMix by simply copying it over the top of the original one.

soapbindingserviceunit_jar – used for the WS entry point into the ESB

soapengineserviceunit_jar – used for the WS entry point into the ESB

The *.jardesc files are Eclipse export files, used to generate the JAR files into the poc_assembly directory with the Eclipse GUI. In Eclipse, right click on one, and select “Create JAR” to build the JAR file. Sadly there was no time to make this nice with an Ant build script...

The PoC requires jdbcserviceunit.jar, soapbindingserviceunit.jar and soapengineserviceunit.jar only. Once these are build into the poc_assembly directory, use Winzip or the jar tool to create a JAR of this directory. This is what is deployed to ServiceMix. The deployment is done through the management console found at <http://localhost:8081/serviceMix-console>. The deployment is done half way down the screen. Ensure that if the mysql-mart deployed assembly already exists (this is the name of the PoC!), that it is stopped, shutdown and uninstalled before re-installing, otherwise the FUSE server needs to be restarted because it gives strange errors.

To do modifications to the code, look in the jdbcserviceunit_jar folder. It contains a Spring style serviceMix.xml file that specifies the components, as well as containing the actual source code and classes. The easiest way to get to grips with this business logic is to attach a remote debugger to port 30033 and set a breakpoint in the business components class. See below for how to start off the dataflow.

The second deployment is the WS wrapper, located at “*open source ESB\website_frontend\apache-tomcat-5.5.17\webapps\axis\OrdersWebService.jws*”. This JWS file is a standard Axis 1.x Java Web Service. It is basically a Java class which the servlet container maps to its extension. The servlet container than handles the change from a web service request into calling the method on the Java class. To deploy this, nothing needs to be done, because it is automatically deployed when the server starts up.

To build the JAXB project, go to “*open source ESB\pivotStructureJAXB*” and run the Ant build script with the default task. Note that the pivot.jar output needs to be deployed to all classpaths where it is used, in addition to the XSD file it maps to.

To build the web service clients, as used by the web site front end, and by the web service wrapper, just run the batch scripts that are contained in the “*open source ESB\website_frontend*” folder. Note that any client using these classes needs to have access to them on their classpath. To determine where the source is generated, examine the contents of the batch file. The '-o' flag is the output directory.

Instructions

To run a demonstration, follow these instructions.

Open the platformCommandLine shortcut 4 times.

In the first type

```
cd mysql-4.1.13a-win32\bin
```

```
mysqld.exe --console
```


This starts the MySQL database that contains the orders, customers and items information.

In the second, type

```
cd mysql-4.1.13a-win32\bin
```

```
mysql.exe -u root -p
```

This starts the MySQL client, as user 'root' specifying that you will enter the password. It prompts you for the password. Type

```
password
```

Type

```
use esbproject_poc
```

to select the database.

Find the file at “open source ESB\logicblaze\FUSE-1.1\FUSE-1.1_ant\demos\poc\resources\dbcreator.sql”, open it and copy its contents. In the command line that contains the MySQL client, right click to paste the file contents. This will wipe the database and install all the tables fresh.

Go to the third command line and type

```
cd website_frontend\apache-tomcat-5.5.17\bin
```

```
catalina.bat jpda run
```

This starts up the web server, Tomcat. Tomcat contains two web applications. The first is the generic website GUI which mimics the site where orders are placed. It can be found at http://localhost:8080/website_frontend/

The second web application is the web service wrapper, hosted at <http://localhost:8080/axis/OrdersWebService.jws?wsdl> where you can see it implements the WSDL as defined in the requirements, earlier in this document.

Once Tomcat is running, you can attach a remote debugger to it on port 30031, if you find that kind of thing useful.

Finally, go to the fourth command line and type

```
cd logicblaze\FUSE-1.1\FUSE-1.1_ant\bin
```

```
fuse.bat
```

This starts the FUSE server. Once it is running, you can attach a remote debugger to it on port 30033, if you find that kind of thing useful.

FUSE is publishing its web service at <http://localhost:8192/Service/main.wsdl>, where you can see the WSDL. This WSDL is not as required in the above requirements, and hence the need for the web service wrapper in this design.

Now, to start a demo, go MySQL client and type

```
show tables;
```

This gives a list of the tables. They are:

customers – a table containing customer credit information

items – a table providing item pricing information

orders – a table providing order status information

products_in_order – a table providing a list of the quantity of each product in each order

Type

```
select * from customers;
```

And make a note of the credit of each customer.

Then type

```
select * from orders;
```

and note there are two existing orders. The status of placed means that it has been sent to the ERP system (via an XML output). The status of pending means that it is waiting to be authorized or rejected by a manager.

Now, go to “*open source ESB\mailserver*” and double click Ability Server_2.34.exe to start the mail server. It should already be preconfigured with a user:

```
username: boss
```

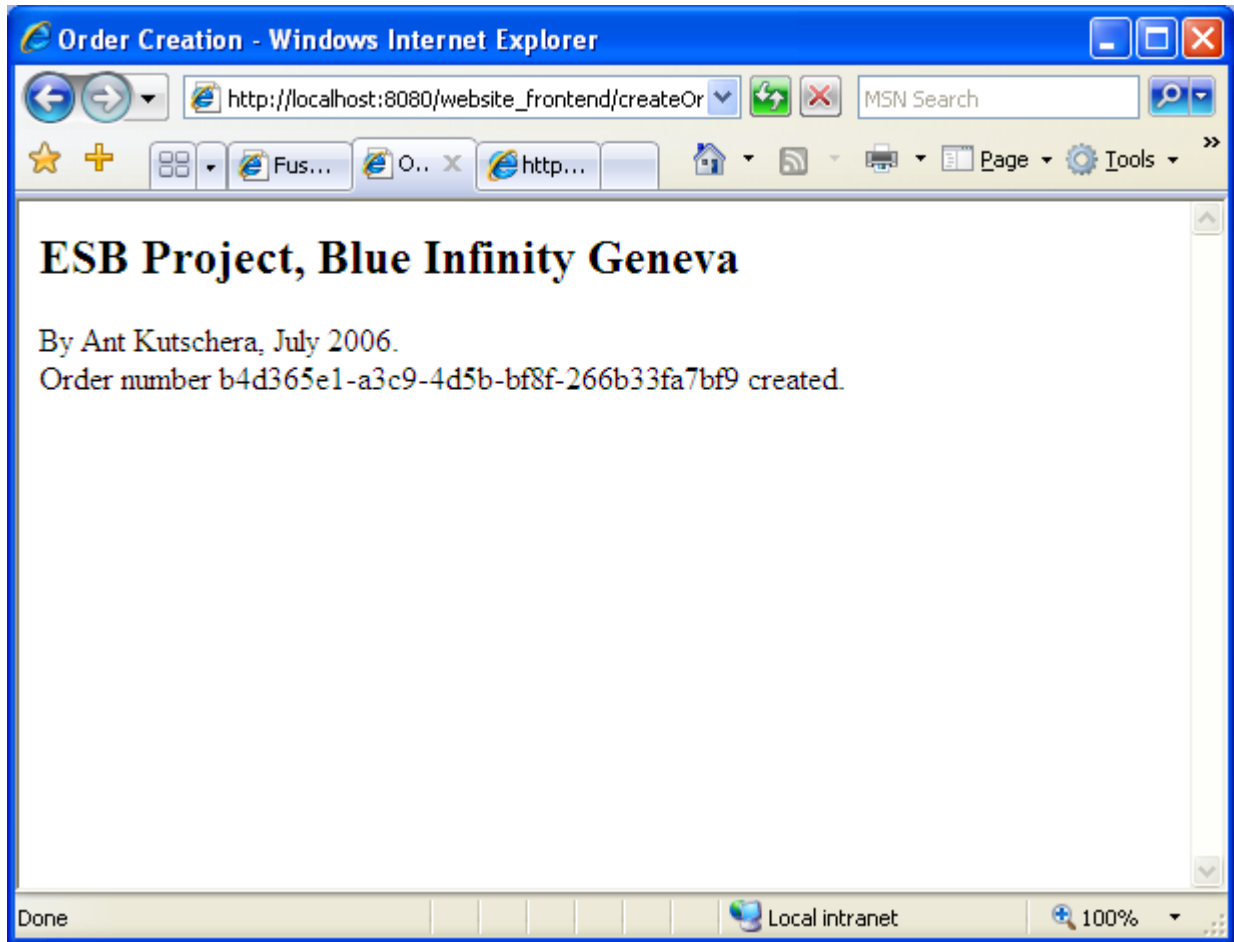
```
password: boss
```

and that user should own the catch-all-account meaning that all email sent to the mail server goes to this user.

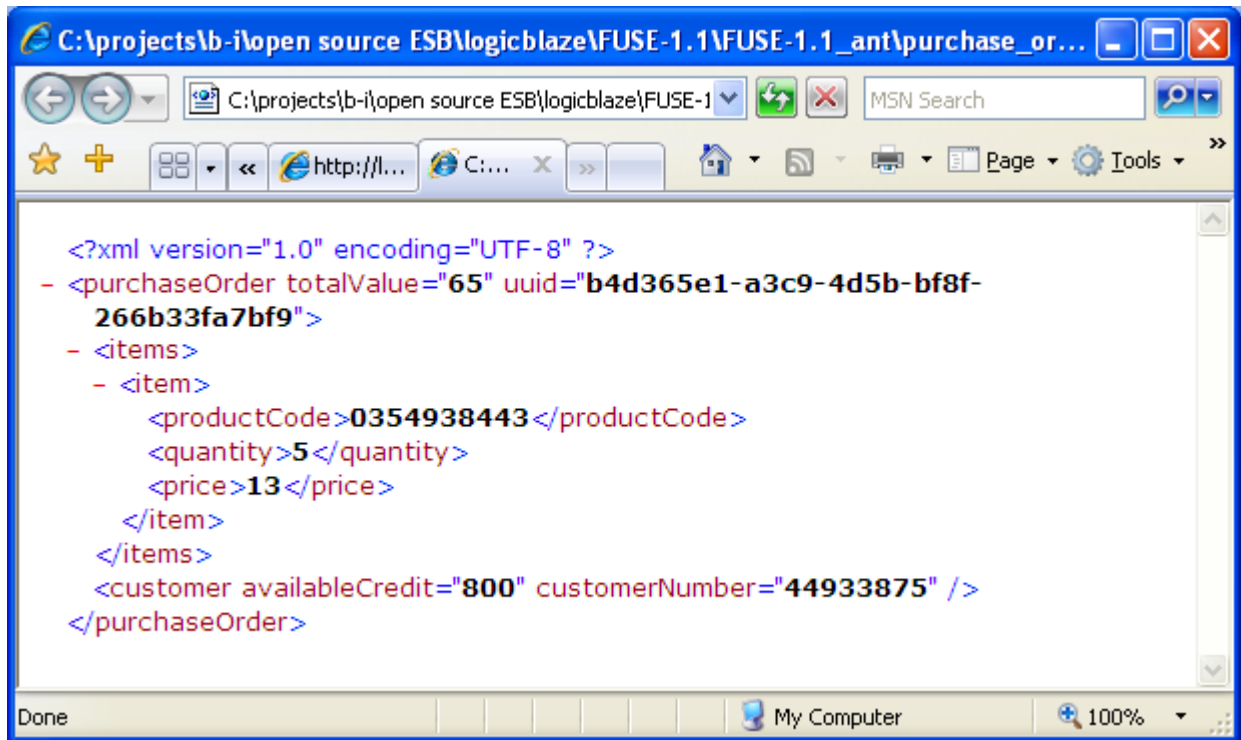
So, to start the demo, go to http://localhost:8080/website_frontend and create an order as follows:



Submit the form, and you will get a response with a new order number that is randomly generated – it is the UUID. For example,



As this customer has good credit, and it was a low order value, we expect to see it as an XML file in the output directory, at *“open source ESB\logicblaze\FUSE-1.1\FUSE-1.1_ant\purchase_orders_to_process”*. It is in a file with the name *“po_XXX.xml”* where XXX is the UUID as displayed on the browser response. Ours looks like this when opened in Internet Explorer:



The screenshot shows a web browser window with the following XML content displayed:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <purchaseOrder totalValue="65" uuid="b4d365e1-a3c9-4d5b-bf8f-266b33fa7bf9">
- <items>
- <item>
  <productCode>0354938443</productCode>
  <quantity>5</quantity>
  <price>13</price>
</item>
</items>
<customer availableCredit="800" customerNumber="44933875" />
</purchaseOrder>
```

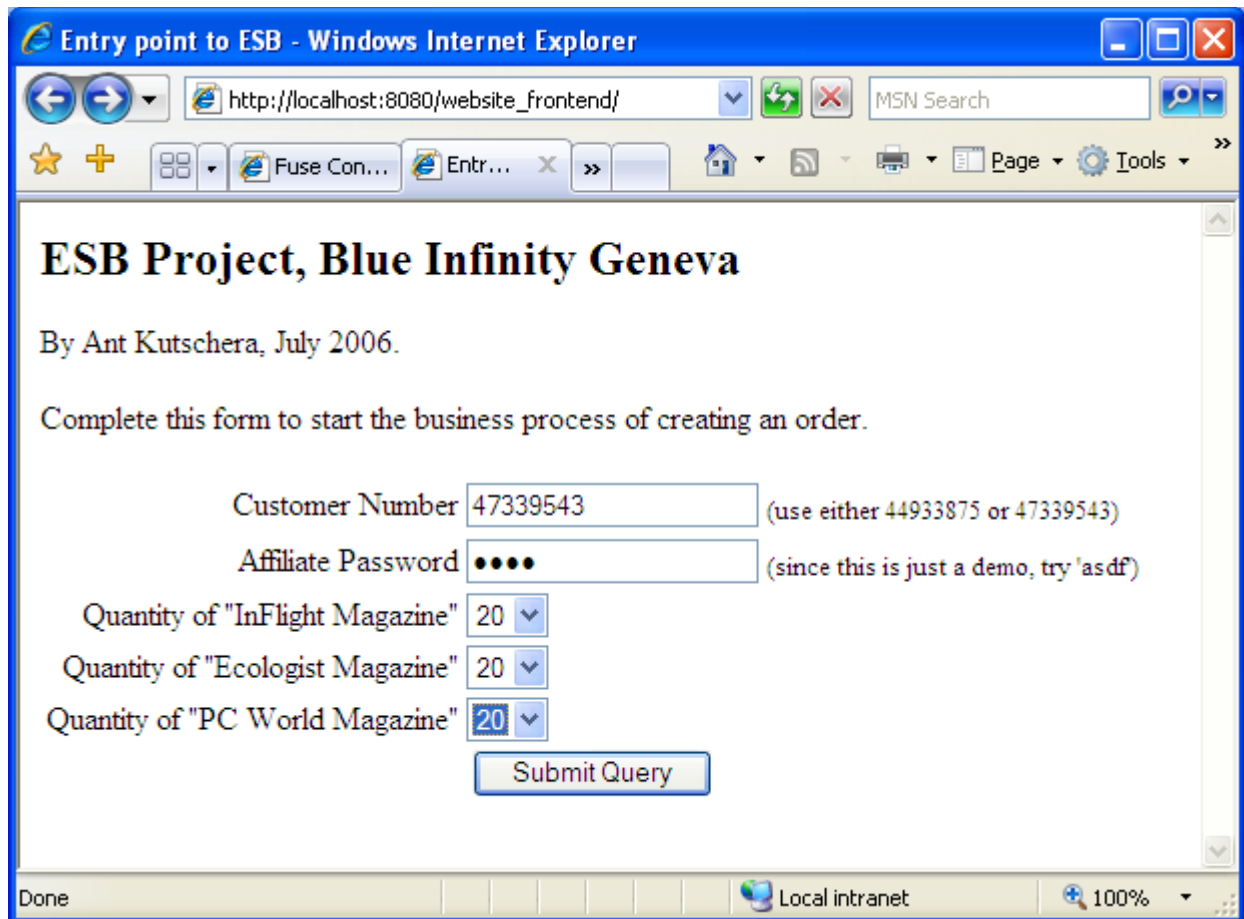
Next, use the MySQL client and type

```
select * from customers;
```

```
select * from orders;
```

You will see that the customers credit has reduced, and that a new order has been created with a status of 'placed', meaning that the XML file was written.

Now, to test the scenario where the order is large and needs to be authorized. Go back to the order screen in the browser, and create an order like this:



Submit the form, and again, you see a new UUID for the second order.

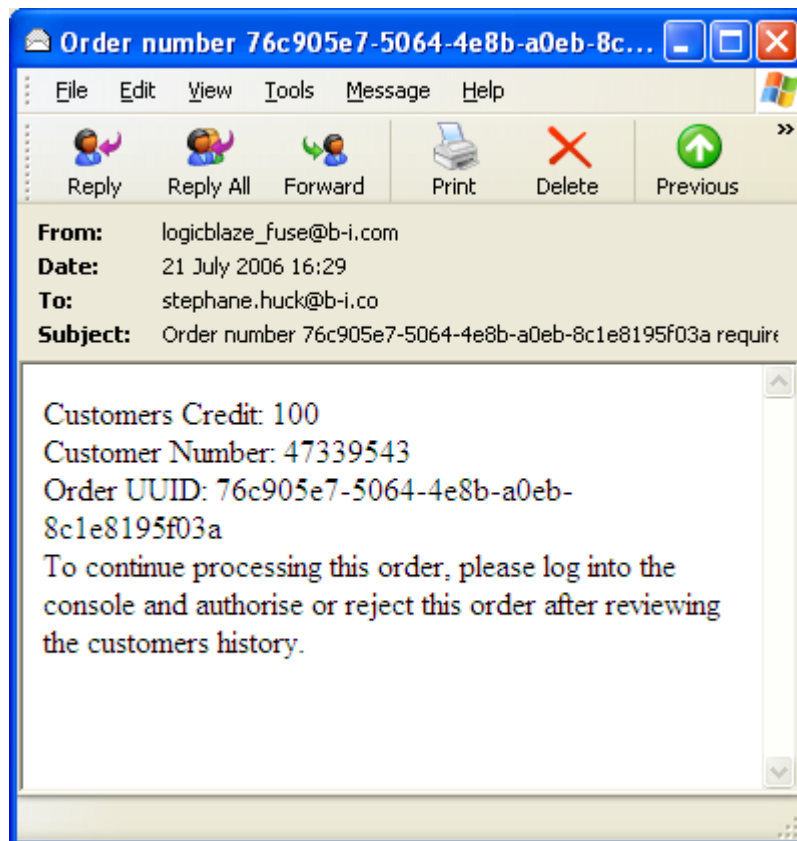
Now, go to the MySQL client and type

```
select * from orders;
```

and you will see that there is another order, but that it has a status of 'pending'. This is because it needs to be authorized, because it is of a high value, and the customer has not got enough credit.

Checking the output folder will show that it has not been created as an XML file.

The last check is to go to your favorite mail browser and to configure an account for the 'boss' user (as shown above). This results in one new email, as follows:



So, both use cases have passed successfully.

1.4. Product II Evaluation

To be completed.

1.5. Product III Evaluation

To be completed.

1.6. Product Evaluation Results

To be completed.

1.7. Product Evaluation Discussion

To be completed.

1. References

- 1) *What is an ESB?*, JBoss.org, <http://labs.jboss.com/portal/jbossesb/resources/WhatIsAnESB.html>
- 2) *Top Ten Reasons to use a True ESB*, CapeClear, <http://www.capeclear.com/download/whitepapers/TrueESB.pdf>
- 3) *Service-Centric versus Message-Centric ESBs*, CapeClear, http://www.capeclear.com/download/whitepapers/The_Service_Centric_ESB.pdf
- 4) *Using JBI for SOI*, Java.net, <https://open-esb.dev.java.net/public/whitepapers/JBIforSOI.pdf>
- 5) *JSR208: Java Business Integration*, Sun, <http://www.jcp.org/en/jsr/detail?id=208>
- 6) *The Spring Framework*, Spring, <http://www.springframework.org>