Integrating OSB 10gR3 with Oracle BAM 11g
A SOA Technology Article
by Aritza Iratzagorria (aritza.iratzagorriagaray@osakidetza.net) and
Nacho Lafuente (nacho.lafuente.botella@oracle.com)

Updated on Jan 20, 2010
# Table of Contents

1  Introduction .................................................................................................................. 3  
   1.1  Oracle Service Bus (OSB) ....................................................................................... 3  
   1.2  Oracle Business Activity Monitoring (Oracle BAM) ............................................. 5  
   1.3  The need for OSB and Oracle BAM Integration ....................................................... 7  

2  OSB 10gR3 Integration with Oracle BAM ...................................................................... 8  
   2.1  Integration Scenarios .............................................................................................. 8  
   2.2  Architecture ........................................................................................................... 9  
   2.3  OSB BAM Feeder .................................................................................................... 10  
   2.4  Deployment ........................................................................................................... 11  
   2.5  Testing .................................................................................................................. 14  
   2.6  Error Handling ...................................................................................................... 16  
      2.6.1  Handling System Errors ................................................................................. 17  
      2.6.2  Handling Application Errors ......................................................................... 20  

3  Electronic Prescription Example ...................................................................................... 21  
   3.1  Osakidetza and Electronic Prescription ................................................................ 21  
   3.2  Integrating Prescriptions into Oracle BAM ............................................................. 23  
   3.3  Testing the solution .............................................................................................. 37  

4  Key Takeaways and Recommendations ......................................................................... 40
1 Introduction

1.1 Oracle Service Bus (OSB)

Oracle Service Bus (OSB) is a key component of SOA Suite and Event Driven Architecture Suite in the Oracle Fusion Middleware product family.

OSB is a proven, lightweight SOA integration platform designed for connecting, mediating, and managing interactions between heterogeneous services, not just Web services, but also Java and .Net, messaging services and legacy endpoints.

Figure 1: OSB functional view

The functional features of Oracle Service Bus can be categorized into the following layers:

- **Connectivity Layer**: that reliably connects any service by leveraging standards web service transports, traditional messaging protocols and configuration of enterprise-specific custom transports.
  - Traditional Messaging: JMS, MQ, EJB/RMI, Tuxedo, FTP/SFTP, SMTP, File, Native Database Lookup
  - Transport SDK: Enterprise-specific custom transports
  - Interoperability: .NET, Tibco EMS, IBM MQ, IBM WebSphere, Apache Axis, B2B Interchange, iWay 5.5 adapters

- **Security Layer**: a rapid service configuration and integration environment that abstracts policies associated with routing rules, security, and service end-point access.
o Transport Security: SSL/Basic Auth, Custom security credentials


o Console Security: Supports Web Single-Sign-On, Role based access

o Policy-based Security: Leverages WS-Security & WS-Policy, Oracle Web Services Manager

**Mediation Layer:** a meta-data driven feature-rich configuration interface for service discovery and validation capabilities for automatic import and synchronization of services with UDDI registries, allows message flow modelling, transformations, third-party service callouts and a test console.

o Message Flow Modelling: Graphical modelling tools, Content based routing, Mixed modes - Async/Sync, Pub/Sub, Exception management

o Discovery & Validation: Automatic import and sync w/ UDDI Registry or Repository, Validates integrity and reconciles conflicts before deployment

o Transformation: XML and non-XML transformation, Xquery or XSLT

o Service Call-out: Java exit (Plain Old Java Object), Web Services call-out

o Test Console: Tracing & trouble-shooting

**Service Management Layer:** a service management environment that includes dynamic service and policy configuration, centralized usage and performance monitoring, and management of services - not just Web services, but also Java, .Net, messaging services, and legacy end points.

o Endpoint URI Management: API to grow and shrink the pool of Endpoint URI

o Dashboard: Fault and performance metrics, Summaries for aggregated ESB

o Monitoring: Monitor alerts on single node or entire ESB server, SLA Application-based Alerts, Alerts on operation metrics, Alerts from message pipeline

o Reporting: Out of the box reporting provider, Custom reporting providers

o Open interfaces: JMX Monitoring Interfaces, SNMP Alerts

o Operational Customization: One screen for operational customization, Advanced searching capabilities
1.2 Oracle Business Activity Monitoring (Oracle BAM)

Oracle Business Activity Monitoring (Oracle BAM) gives business executives the ability to monitor their business services and processes in the enterprise, to correlate Key Performance Indicators (KPIs) down to the actual business process themselves, and most important, to change business processes quickly or to take corrective action if the business environment changes.

Oracle BAM is a component of both SOA Suite and BPM Suite and it is aimed to provide meaningful and event-driven intelligence for end-users through different features:

- Monitor business processes & services in real-time
  - Key Performance Indicators (KPIs)
  - Service-Level Agreements (SLAs)
- Analyze events as they occur
  - Correlate events & KPIs
  - Identify trends as they emerge
  - Alert users to bottlenecks & solutions
- Act on current conditions
  - Event-driven alerts
  - Real-time dashboards
  - BPEL processes & web services integration

Figure 2 represents a high-level diagram of the Oracle BAM architecture.

Figure 2: Oracle BAM High-Level Architecture

The following list explains each high-level architectural component.
1. Information Sources

- The primary source of data is messages that help Oracle BAM update reports and generate alerts very quickly. Oracle BAM can accept tens of thousands of updates per second into a memory-based persistent cache that is the center of the Oracle BAM architecture.
- Any application can send events using Web services or over any JMS-based messaging protocols.
- JEE applications can integrate using the Oracle BAM JCA Adapter. This is the preferred method for applications deployed to Oracle SOA Suite.
- In addition to integrating real-time information from message queues, you can also integrate historical data or information from any operational database or any data warehouse.

2. Oracle BAM

- The Enterprise Integration Framework connects Oracle BAM to enterprise information sources such as database server, flat files, and XML sources. It is also capable of reading messages from all of the major message queue providers, and then running that data through real-time data integration plans. Real-time integration plans cleanse the data, aggregate it, filter it, correlate it, and perform other necessary actions.
- Oracle BAM Active Data Cache (ADC) is a high-performance, persistent, memory-based storage system, designed to support active data and monitoring. It provides access to current business information for event-based reporting and alerting and supports both data objects and external data objects.
- Oracle BAM Report Cache assembles and formats the data for a live report to be displayed. It applies the report definitions to the data sets retrieved from the ADC for presentation in a browser. It manages information paging for viewing and printing reports. After reports are created, they are stored in the ADC, so that report creation is not repeated each time.
- Oracle BAM Event Engine monitors complex data conditions and implements user-defined rules. The Event Engine continuously monitors the information in the ADC for certain conditions and executes the related actions defined in associated rules. It takes a variety of actions in response to those changes, including notifying the appropriate user with an alert and/or report(s).
- Oracle BAM Report Server (BAM Web Apps & ADF) renders views and reports to be displayed in a client browser, either user dashboards or ADF Pages. Find more information on Oracle Application Development Framework (ADF) technology at the ADF Home.
3. Interactions channels

- Alerts are one of the interaction channels for Oracle BAM. Alerts are launched by a set of specified events and conditions, known as a rule. Alerts can be launched by data changes in a report, or they can send a report to users daily, hourly, or at set intervals. Events in an alert rule can be an amount of time, a specific time, or a change in a specific report. Conditions restrict the alert rule to an event occurring between two specific times or dates. As a result of events and conditions, reports can be sent to users by email.

- Oracle BAM Dashboards are the main source to provide interactive information to users. The dashboards are accessed through the Oracle BAM Active Viewer using a Web browser, and they display active data presentations where data continuously updates, formats, and is displayed. When data changes, the display changes in real-time. Oracle BAM Active Studio is the thin user interface for power user handling the design and development of the dashboards. Dashboards can be shared with other users and rules can be created for determining the scheduling and delivery of the reports.

- ADF components may be leveraged to interact with Oracle BAM 11g active data. This feature unleashes the power of ADF to build up interactive applications that are based on data handled by Oracle BAM and provide the end user with highly tailored interfaces.

1.3 The need for OSB and Oracle BAM Integration

OSB is a common entry point for customers leveraging service oriented architectures. Many customers introduce OSB as the key technology to intermediate among service consumers. Once OSB infrastructure is in place, several benefits are perceived:

- **Inventory** – Services are inventoried so further governance activities may take place.
- **Control** – Services are centrally managed and monitored.
- **Configuration** – Service properties (transport, security, transactions) and service flow are configured without coding.

Business users benefit from OSB technology and assume that services are a commodity.

This is exactly the point of view that a SOA Center of Excellence pursues: services are a commodity, and functional users should design their business process using a top-down approach. Once business users have agreed upon this point, services play a central role in the enterprise business. Services are the business process itself. For example, how many patients were prescribed today, how many orders are in process, or what is the average price/ time for completion.

This paper explores the different alternatives to integrate Oracle Service Bus 10gR3 with Oracle BAM 11g.
## 2 OSB 10gR3 Integration with Oracle BAM

### 2.1 Integration Scenarios

Several OSB/Oracle BAM integration scenarios exist.

- **Using Oracle BAM Adapter** - Oracle BAM Adapter 11g is certified with SOA 11g. It is not certified or supported on OSB 10gR3 at this time. Refer to [Oracle SOA Suite Developer’s Guide](#) for more information on using Oracle BAM Adapter from SOA Suite 11g.

- **Developing a custom OSB Reporting Provider** - OSB defines an extensible Reporting Framework that permits adding custom providers. This OSB/Oracle BAM integration approach consists of developing a custom reporting provider that enables selecting information to be sent out to Oracle BAM. This is accomplished by configuring the reporting action inside an OSB service flow, and Oracle BAM picking up the message using Enterprise Message Sources. Refer to [OSB Operations Guide](#) for further information on Reporting action and Reporting Providers.

- **Leveraging Oracle BAM Web Services** – this scenario uses the Oracle BAM web services that Oracle BAM publishes. The `DataObjectOperationsById` service has operations to insert, delete, update, and upsert existing Oracle BAM data objects, so that information gathered at OSB may be easily integrated into Oracle BAM.

The following table contains a summary of the integration scenarios.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSB Reporting Provider</td>
<td>Very simple to use</td>
<td>Although not complicated, the custom reporting provider must be developed.</td>
</tr>
<tr>
<td></td>
<td>Implicit &amp; Intuitive Mapping</td>
<td>Same message goes to default OSB reporting provider (JMS Reporting Provider) and Oracle BAM Reporting Provider</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JMS Reporting Provider may be undeployed to prevent message duplication, but OSB reporting console functionality becomes unavailable.</td>
</tr>
<tr>
<td>Oracle BAM Web Services</td>
<td>Simple to use</td>
<td>Requires custom OSB assets (services and transformations) to be designed</td>
</tr>
<tr>
<td>Oracle BAM Adapter</td>
<td>Strategic</td>
<td>Not available today on OSB 10gR3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle BAM Adapter 11g is not certified on OSB 10gR3</td>
</tr>
</tbody>
</table>

The rest of this paper focuses on an OSB/Oracle BAM integration solution based on the Oracle BAM Web Services approach.
2.2 Architecture

The selected integration architecture is represented in Figure 3. The OSB/Oracle BAM integration component is called OSB BAM Feeder, and it creates an integration link between client candidates and Oracle BAM. The OSB BAM Feeder is designed as an OSB user project, so it automatically benefits from all existing OSB features. Clients may be internal (other OSB projects) or external (other applications). The OSB BAM Feeder itself may be represented as an integration service that complies with a service contract. This contract defines the service responsibilities, operations, semantics, and other operational values as quality of service or availability.

Figure 3: OSB-BAM Integration Architecture

Figure 4 represents the messaging implications regarding the OSB/Oracle BAM Integration. Each source message to be integrated into Oracle BAM will have its own format (MyPrescription in the example). Oracle BAM defines a basic component called a data object. A data object represents a data entity whose information is to be stored in Oracle BAM. Each new message to be integrated into Oracle BAM must be validated against an underlying XML schema specific for the selected data object (BAMPrescription in the example).

Independently of the integration scenario, client messages must be transformed into Oracle BAM messages at some point. This transformation must always be defined, and may be implemented using different technologies. Currently, Query and XSLT transformations are the methods provided by OSB. The transformation should be packaged along the client service because it depends on the source message.

Figure 4: OSB-BAM Integration Messaging
2.3 OSB BAM Feeder

This integration scenario has been designed and developed from scratch and called OSB BAM Feeder. It leverages the Oracle BAM Web services and specifically, the DataObjectOperationsById service. This Web service has operations (insert, delete, update, and upsert) on existing Oracle BAM data objects, also supporting a batch mode to execute several operations in a single call.

Figure 5 contains a diagram for the OSB BAM Feeder components that made up the integration solution. These OSB artifacts have been designed and configured in order to create a service that can be used to effectively provide OSB/Oracle BAM integration. Following section contain a step-by-step guide to deploy this solution and test it on a SOA Suite installation.

**Figure 5: OSB BAM Feeder components**

OSB BAM Feeder has been designed to meet several requirements:

- **Performance** - Messages are integrated into Oracle BAM asynchronously to avoid client’s performance overhead.
- **Reliability** - Messages delivered in asynchronous mode are transported using JMS/XA that enables persistence (to avoid losing messages in case of server failure) and transactions (to permit Exactly Once delivery).
- **Transport independence** - Inbound transports currently include local access (for internal OSB clients) and HTTP (for external clients).
- **Availability** - The solution is designed to provide the same availability level as the OSB infrastructure does. By leveraging OSB proxy services, JMS transport and error handling techniques, OSB BAM Feeder will be available at least if one OSB instance is alive.
2.4 Deployment

Follow these instructions to deploy OSB BAM Feeder to your environment. Some OSB and Oracle WebLogic Server background knowledge is assumed. If you need further information please refer to Oracle WebLogic Server 10.3 Documentation or OSB 10gR3 Documentation. Scripts delivered with this sample only work for Microsoft Windows environments.

1. Unzip OSBBamFeeder.zip to your local environment. You should have the following contents.
   a. bam - Oracle BAM resources (data objects, dashboard)
   b. osb - OSB resources (BAM Feeder project and Mortgage Broker testing services)
   c. testing - XML testing payloads
   d. wls - Oracle WebLogic Server applications & JMS resources
   e. setEnv.cmd - Script to set required environment variables

2. Adjust the environment variables defined in setEnv.cmd to reflect your local environment.
   a. OSB_HOME - OSB 10gR3 installation home, needed to locate OSB libraries and tools.
   b. WLS_HOME - Leave default value (Oracle WebLogic Server packed with OSB),
   c. SOASUITE_HOME - SOA Suite installation home, needed to locate Oracle BAM libraries and tools.
   d. JAVA_HOME - Path to JDK 1.6 (default path points to JRockit bundled with OSB).

3. Create or get access to an OSB Domain. The rest of these steps assume that the OSB instance is named AdminServer and listening in localhost at port 7021.

4. Adjust the Oracle WebLogic Server variables defined in wls/serverEnv.properties to reflect your deployment environment.
   a. wls_url - Administration URL (default is t3://localhost:7021)
   b. wls_user - Administration user
   c. wls_password - Administration password
   d. target - Administration instance name (default AdminServer)

5. Deploy the Mortgage Broker sample application to Oracle WebLogic Server.
   a. Execute the wls/deployApplications.cmd script.
b. Four Web service applications and one Web application should be deployed.

c. Open the Oracle WebLogic Server Console at http://localhost:7021/console, and verify that these new deployments are active.

d. Access http://localhost:7021/examplesWebApp/index.jsp when the deployment is finished to double-check.

6. Deploy JMS resources to Oracle WebLogic Server.

   a. Execute the wls/deployJms.cmd script.

   b. This script creates several JMS resources:

      i. A File Store to persist JMS messages

      ii. A JMS Server called OSBBAMJMSServer. The server persists messages on a file store.

      iii. A new JMS Module called OSBBAMJMSModule. This module contains the JMS resources required by OSB BAM Feeder to work properly.


   d. If JMS resources are to be deployed on a cluster:

      i. Create one JMSServer for each instance in the cluster. Enable message persistence.

      ii. The Connection Factory should be targeted to the cluster

      iii. The queues should be created as distributed queues

      iv. The subdeployment should be targeted to all JMS Servers belonging to the cluster.

7. Deploy BAM Feeder project to OSB

   a. Execute osb/deployOsb.cmd script.

   b. Double-check the imported project by using OSB Console at http://localhost:7021/sbconsole. You should see the projects Oracle/BamFeeder and Oracle/BamMortgageBroker imported.
8. Now it’s time to deploy the Oracle BAM resources. Create or get access to an Oracle BAM domain. The rest of the steps assume that an Oracle BAM instance is listening in localhost at port 9001. For more information on creating an Oracle BAM domain refer to the Oracle SOA Suite Installation Guide.

a. Execute the `bam/import_all.cmd` script. This script imports a sample data object definition and a dashboard.


c. Log in and select **Architect**.
   
   i. Select **Data Objects** from the upper-left drop-down list.
   
   ii. Navigate to the **OSB** folder and select the **Loan** data object. You should see an object layout like the following.

![Data Objects Layout](image)

   d. Close the window and select **Active Viewer** from the Oracle BAM Console.
   
   i. Click the **Select Report** button.
   
   ii. Select the **Shared Reports/OSB/OSB Loan Gateway 1 Report**.
   
   iii. You should see a dashboard similar to the following (differences may appear due to different data).
2.5 Testing

The sample Mortgage Broker application will be used to verify that everything is deployed and configured as expected. The Mortgage Broker application is shipped with OSB to demonstrate basic OSB service configuration. The OSB BAM Feeder deploys a modified version of the Mortgage Broker application that features Oracle BAM Integration.

1. Open the Oracle BAM Console available at http://localhost:9001/OracleBAM/
   a. Select Active Viewer.
   b. Select the Shared Reports/OSB/OSB Loan Gateway 1 Report.
   c. Arrange this window in one side/corner of the screen.

   a. Select the Routing a Loan Application example and click Run the example.
b. Now you have a Loan Request form that submits information over OSB services. The OSB service has been intercepted to extract information from the loan service and send it out to Oracle BAM using the OSB BAM Feeder component. Whenever a new loan request is submitted the Oracle BAM dashboard will reflect that information in real time.

**Oracle Service Bus Examples**

**Oracle Service Bus: Routing a Loan Application**

Adjust the *requested loan rate* to change the Oracle Service Bus routing behavior. If the requested loan rate is less than 5% the Oracle Service Bus proxy service routes this loan application to the ManagerApprovalService business service. If the requested rate is greater than or equal to 5% the loan application is routed to the NormalLoanApprovalService business service.

<table>
<thead>
<tr>
<th>Enter the applicant last name</th>
<th>Smith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter the applicant identification number</td>
<td>123456789</td>
</tr>
<tr>
<td>Enter the requested loan amount</td>
<td>900000</td>
</tr>
<tr>
<td><strong>Enter the requested loan rate</strong></td>
<td>5.3</td>
</tr>
<tr>
<td>&lt;5% routes to the ManagerApprovalService</td>
<td></td>
</tr>
<tr>
<td>Enter the requested loan duration in years</td>
<td>30</td>
</tr>
<tr>
<td>Add a note</td>
<td>loan request for me!</td>
</tr>
</tbody>
</table>

**Submit Loan Application**

b. Test several loan rates (that is, 3.0 and 5.3) to see how the Oracle BAM dashboard reacts to integrated information.

You can also directly test some of the services that OSB BAM Feeder provides. The folder *testing/* provides different payload messages that may be used as input.

1. Open the OSB Console available at [http://localhost:7021/sbconsole](http://localhost:7021/sbconsole)
2. Select the Project Explorer menu and navigate to *Oracle/BamFeeder/ProxyServices*. Launch *Test Console* for the *BamInternalFeederPS* proxy service.
3. Copy and paste payload testing/BamFeeder-Insert.txt into the Test Console. Unselect the Direct Call checkbox so that the request goes through the JMS request queue.

4. If everything runs as expected, you should get no response, and the OSB log file won’t display anything. The OSB BAM Feeder is designed for a production environment, so no logging is generated if everything is running correctly. However, the Oracle BAM dashboard should reflect your last request.

2.6 Error Handling

The OSB BAM Feeder was designed to meet production environment quality. One of the most important issues for Operations & Administration is error handling. Once the OSB BAM Feeder goes into production, it should be tailored to permit monitoring, alerts, and troubleshooting.

The OSB BAM Feeder benefits from generic OSB system administration best practices and features. Further information on these topics may be accessed at the OSB Operations Guide.
The goal of efficient error handling is to silently work under normal operation, and provide a specific procedure for each unexpected error situation.

The main entry points for OSB BAM Feeder are the `BamAsyncFeederHttpPS` and `BamAsyncFeederLocalPS` proxy services. These services follow the “Fire&Forget” design pattern (no response is ever returned) and are asynchronous (request payload is queued up).

Errors will be notified later at the `BamInternalFeederPS` proxy service. This service is responsible for de-queuing messages and processing the payload to create a valid Oracle BAM request.

OSB BAM Feeder differentiates system and application errors.

- **System Errors** – those related to communication (that is, Oracle BAM Server is down or not reachable) or internal (that is, some JMS resource is unavailable). System errors are notified to OSB administrators and message is retried using JMS error handling parameters. If the retry limit is reached, the message is forwarded to an error queue and persisted. Whenever the error condition is solved (that is, Oracle BAM Server is started again after maintenance is done), the administrator may decide to re-queue the messages to allow normal processing.

- **Application Errors** – those related to the client application, mainly due to invalid payload or other error reported at the Oracle BAM side. An application error is identified because the Oracle BAM Web service used to send out messages is returning a SOAP fault. Application errors are notified, and the message is discarded (an invalid payload is not expected to change over time).

### 2.6.1 Handling System Errors

This example shows how OSB BAM Feeder handles system errors if Oracle BAM Server is down for maintenance.

This test must be executed with the OSB instance up and running, and the Oracle BAM instance not available.

2. Select Project Explorer menu and navigate to `Oracle/BamFeeder/ProxyServices`. Launch Test Console for the `BamInternalHttpFeederPS` proxy service.
3. Copy and paste payload `testing/BamFeeder-Insert.txt` into the Test Console and click Execute.
4. The service response is empty because is not designed to return any response. Remember that OSB BAM Feeder is asynchronous to maximize performance at runtime.
5. Whenever this request is de-queued and executed it will raise a system error because Oracle BAM is not running. The JMS redelivery error configuration will handle the different retries. Every request is de-queued inside the context of a transaction. If routing is not possible, the transaction is aborted and the message is not processed until the
redelivery limit is reached. When this limit is reached, the configured redelivery policy states that it should be forwarded to an error queue.

a. Every time the transaction is aborted, the OSB log file will reflect the following stack trace.

<17-nov-2009 19H47' CET> <Info> <EJB> <BEA-010213> <Message-Driven EJB: RequestEJB-2669023879117828128-bf3182a.124fc2335c-7fd6's transaction was rolled back. The transaction details are: Xid=BEA1-01768203F36FFB121A37(29935953),Status=Rolled back. [Reason=weblogic.transaction.internal.AppSetRollbackOnlyException],numRepliesOwedMe=0,numRepliesOwedOthers=0,seconds since begin=0,seconds left=60,XAServerResourceInfo[WLStore_osbDomain_FileStore]=(ServerResourceInfo[WLStore_osbDomain_FileStore]=(state=rolledback,assigned=AdminServer),xar=WLSstore_osbDomain_FileStore1367845,registered = false),JInfo[osbDomain=AdminServer]=(state=rolledback),OwnerTransactionManager=ServerTM[ServerCoordinatorDescriptor=(CoordinatorURL=AdminServer+192.168.200.2,NonXAResources={}),CoordinatorURL=AdminServer+192.168.240.1:7021+osbDomain+t3], XAResources={cgDataSource, wlsbjmsrpDataSource, WLStore_osbDomain_WseeFileStore, weblogic.deployment.jms.WrappedXAResource_weblogic_transaction_internal_TransactionManagerImplJMSXAResource, WLStore_osbDomain_FileStore, WLStore_osbDomain_WLS_AdminServer},NonXAResources={})>,

b. The following image shows the JMS transport configuration for the BamInternalFeederPS proxy service. These parameters define how error handling proceeds.

c. The OSB exception handling procedure also includes a Log Action to further describe the system error.

<17-nov-2009 19H47' CET> <Warning> <ALSB Logging> <BEA-000000> < [null, null, null, ERROR] System error routing to BAM (retried): Fault: BEA-380002None of the URIs are online for Business service BamDOOByIdBSRoute to BAM servicerequest-pipeline>

d. Finally, the OSB exception handling procedure raises a Pipeline Alert to notify system administrators about the system error. The Operations dashboard, at Pipeline Alerts tab will show two alerts (because one additional retry has been configured).

e. The Operations dashboard may also be used to monitor services with problems. A simple search to locate services with offline URIs will detect the Oracle BAM BamDOOById Business Service, as this service is not currently reachable and OSB has decided to take its URI offline to improve performance.
6. System errors are configured to always persist the message. Whenever the error condition is cleared (that is, Oracle BAM Server is up again after maintenance) the stored messages may be retried again. All messages that have reached redelivery limit are stored at the OSB BAM Feeder JMS error queue.


   b. Select Services > Messaging > JMS Modules > OSBBAMJMSModule.

   c. Select the jms.OSBBAMFeederErrorQueue queue.

   d. Select the Monitoring tab.

   e. Select the checkbox by OSBBAMJMSModule!jms.OSBBAMFeederErrorQueue queue, and click Show Messages.

   f. All messages stored at the error queue will be shown.

   g. To resume processing those messages, select the Move > Move All option. Follow the steps to select OSBBAMJMSServer as the destination JMS Server and OSBBAMJMSModule!jms.OSBBAMFeederQueue as the destination queue. This procedure could be easily automated using WebLogic Script Tool (WLST) if needed.

7. After messages are moved to the main processing queue, the process starts over again with no previous memory. If the system error condition still exists, it will fail over the exception handling procedure.
2.6.2 Handling Application Errors

This example shows how OSB BAM Feeder handles applications errors if an invalid payload is sent out.

This test must be executed with both OSB and Oracle BAM instances up and running.

1. Open OSB Console available at http://localhost:7021/sbconsole
2. Select Project Explorer menu and navigate to Oracle/BamFeeder/ProxyServices. Launch Test Console for the BamInternalHttpFeederPS proxy service.
3. Copy and paste payload testing/BamFeeder-Insert.txt into the Test Console.
4. Modify the request to indicate an invalid Oracle BAM data object identifier. Change /OSB/Loan to any other value so that the request looks like the following.

```xml
<bam:request xmlns:bam="http://xmlns.oracle.com/bam">
  <bam:operation>insert</bam:operation>
  <bam:objectType>/OSB/Loan/BlublaBla</bam:objectType>
  <bam:payload>
    <_Loan>
      <_SSN>002</_SSN>
      <_Name>Nacho</_Name>
      <_Rate>12</_Rate>
      <_Amount>14000</_Amount>
      <_NumOfYear>2</_NumOfYear>
      <_Notes>hola caracola</_Notes>
    </_Loan>
  </bam:payload>
</bam:request>
```
5. The service response is empty because it is not designed to return any response. Remember that the OSB BAM Feeder is asynchronous to maximize performance at runtime.
6. Whenever this request is de-queued and executed it will be routed to Oracle BAM Web service. This Web service is based on standard SOAP envelopes, so any error with be returned using a SOAP fault. As the indicated Oracle BAM data object is invalid, the Oracle BAM Web service will reply with an error. The OSB BAM Feeder error handling procedure will detect this SOAP fault and manage the application error. Application errors are due to malformed or invalid payloads, and the offending request message is discarded after the notification.

   a. The transaction that de-queued the message is rolled back, and the message is discarded. The log file displays the following error information.

```xml
<17-nov-2009 22H56' CET> <Info> <EJB> <BEA-010213> <Message-Driven EJB: RequestEJB-2669023879117828128-bf3182a.12f2c2335c6.-7fd6`'s transaction was rolled back. The transaction details are: Xid=BEA1-0B178203F3EFB121A37(1185418),Status=Rolled back. [Reason=weblogic.transaction.internal.AppSetRollbackOnlyException],numRepliesOwedMe=0,numRepliesOwedOthers=0,seconds since begin=0,seconds left=0,XAServerResourceInfoWLStore_osbDomain_FileStore=(ServerResourceInfoWLStore_osbDomain_FileStore)=(state=rolledback,assigned=AdminServer),xar=WLStore_osbDomain_FileStore1367845,re-Registered =false),SCInfo=osbDomain+AdminServer=(state=rolledback),OwnerTransactionManager=ServerTM|ServerCoordinatorDescriptor=(CoordinatorURL=AdminServer+192.168.2
```

b. The OSB exception handling procedure also includes a Log Action to further describe the application error.

```
```

c. Finally, the OSB exception handling procedure raises a Pipeline Alert to notify system administrators about the application error, clearly stating that this message has been discarded. The Operations dashboard, at Pipeline Alerts tab will show the alert.

7. Application errors should seldom be reported at production environments. These errors indicate a configuration problem, either on the OSB side (the client is sending out malformed/invalid payload) or on the Oracle BAM side (data object configuration, or any other internal server issue).

## 3 Electronic Prescription Example

### 3.1 Osakidetza and Electronic Prescription

The Basque Country (Basque: Euskadi, Spanish: País Vasco) is an Autonomous Region in Northern Spain with a total area of 7,234 sq km (2,793.1 sq mi) and a population of 2.155.546 (2008).

Osakidetza is the organization that manages the Public Health Service for Basque Country and depends on the Health Department (Basque Country Government).

Osakidetza manages 29 autonomous health services:

- 7 health regions
- 11 general hospitals
- 9 specialized hospitals
- Emergencies
Tissue & Blood

Electronic Prescription will enable prescribers, such as General Practitioner or specialized doctors, to send prescriptions electronically to a dispenser (such as a pharmacy) of the patient’s choice. This makes the prescribing and dispensing process safer and more convenient for patients and staff. The Health Department has engaged a project to roll out electronic prescriptions to all health services in Euskadi.

Osakidetza IT manages several Health Information Systems (HIS) to support the different health services. A Service Oriented Architecture (SOA) approach was selected to evolve these systems. The Electronic Prescription was designed and deployed on top of the new SOA Reference Architecture. This SOA Reference Architecture, among other products and technologies, includes Oracle Service Bus (OSB) and Oracle BAM.

Figure 6 represents the Electronic Prescription high-level architecture. Physicians use the Electronic Prescription HIS Application to prescribe whatever is needed to a patient. This HIS application eventually consumes the Prescription Notification Service that is published at OSB. The main goal of this service is intermediating between the consumer (HIS application) and the provider (Health Department Prescription Processing Service), but the OSB BAM Feeder solution is also being leveraged to publish prescription information to Oracle BAM.

Inside Oracle BAM, a new Prescription data object was defined to store that information, and a Prescription Dashboard was designed to enable business users to monitor the business process, and eventually act on certain conditions (that is, if some hospital suddenly stops prescribing).

Figure 6: Electronic Prescription Architecture

Figure 6: Electronic Prescription Architecture
3.2 Integrating Prescriptions into Oracle BAM

This section contains a detailed step-by-step guide to describe how the prescription information is integrated into Oracle BAM.

1. The Prescription Notification Service is published at OSB.
   a. This service is assumed to be predefined, so no additional instructions will be given.
   b. This service contains a proxy service (RecetasEnvioPS), a business service (RecetasEnvioBS) and an XML Schema (RecetasEnvioXSD) that defines the messaging format.

2. Design the Oracle BAM Prescripcion data object.
   a. A data object is required in Oracle BAM to store information related to prescriptions. This data object depends on the business itself and will be defined by business users.
   b. Detailed instructions on how to create an Oracle BAM data object may be found at Oracle Fusion Middleware SOA Suite Developer's Guide.
   c. The following image represents the /Osakidetza/Asistencial/Comun/Prescripcion data object that was defined and is used throughout this example.

3. Retrieve the Oracle BAM data object XML schema.
   a. Oracle BAM data objects are defined by business users and have their own format. This format is defined by an XML schema and is needed to automate the transformation from the source message format in OSB to target message format in Oracle BAM.
b. For the prescription service, a transformation is required to convert the original message format (defined by the RecetasEnvioXSD schema) to the Oracle BAM /Osakidetza/Asistencial/Comun/Prescripcion data object.

c. Oracle JDeveloper 11g is used to automate this transformation. You should have at least Oracle JDeveloper version 11.1.1.1 with the Oracle SOA Composite editor installed.
   - Download Oracle JDeveloper from the JDeveloper Home Page.
   - Install Oracle JDeveloper.
   - Run Oracle JDeveloper and select Help > Check for Updates.
   - Select the Oracle SOA Composite Editor and finish the update process.

d. Select File > New and create a new SOA application. This application will only be used to create the required transformation assets. When finished, these assets will be imported into Oracle Service Bus and this SOA application is no longer needed.
e. Click `composite.xml` to open the SOA Composite Editor. The selected composite is empty.

f. Create a connection to Oracle BAM Server to view the data objects that are available to enable Oracle JDeveloper to automatically generate appropriate assets.

   - Right-click **Connections** in the Application Resources navigator and choose **New Connection > BAM**.

   - In Step 1 (Name page), select **Application Resources**, accept the default name `BAMServerConnection1`, and click **Next**.
- Enter the host name of your server into **BAM Web Host** and **BAM Server Host**. Enter the HTTP port for Oracle BAM in the **HTTP Port** field.

- On the Test Connection page, click **Test Connection**. Success is displayed in the **Status** window when you achieve a successful connection.

- Click **Finish**.

- In the Application Navigator, **Application Resources** pane, the new Oracle BAM Server connection is listed in the **Connections** section under the **BAM** node.
g. Oracle BAM Adapter will be used to retrieve the XML schema that defines the Pracle BAM Prescripción data object.

- Drag and drop an Oracle BAM Service Adapter into the SOA Composite.

- This will automatically open the configuration wizard for Oracle BAM Adapter.

- Give the service any name and click **Next**.
- Use the browse button to select the **Prescripción** data object under the correct patch and click **OK**.

- Click **Next** until the dialog is finished.

h. The XML schema that defines the Prescription data object is now stored in the **xsd** project folder.

4. Create the transformation from the OSB source message format to the target Oracle BAM message format.

a. This transformation may be created using either XQuery or XSLT, both are supported transformation technologies in OSB.
b. If you feel comfortable hand-coding XQuery/XSLT skip to the next step.

c. You may use some tooling or manually edit the transformation. For the rest of this example, Oracle Workshop for WebLogic 10gR3 will be used. This IDE currently supports the graphical edition of OSB 10gR3 assets.

d. Click **Oracle Workshop for WebLogic** 10gR3 to open the IDE. This IDE is packaged with the Oracle Service Bus 10gR3 installation, and it contains a graphical XQuery editor that will facilitate the transformation editing and testing. More information on this product may be found at the [Workshop Home](#).

e. You could reuse an existing OSB project to create the required transformation. Developing OSB projects with Workshop is out of the scope of this paper – you can find additional information at the [OSB Plug-ins for Workshop Page](#).

f. Click **File > New** and create a new OSB Project. Click **Next** until wizard is finished.

![New](image)

**Select a wizard**

Create an Oracle Service Bus Project.

**Wizards:**

```
Oracle Service Bus
Alert Destination
Business Service
Custom Resource
JNDI Provider
MFL
Oracle Service Bus Configuration Project
Oracle Service Bus Project
Proxy Server
Proxy Service
Service Account
Service Key Provider
SMTP Server
```

Click **Oracle Service Bus Configuration Project** to create a new OSB Project.

Click **Next** to finish creating the new OSB Project.

![New Project](image)

g. Drag and drop the source and target XML schemas into the OSB Project. Press **F2** to rename the files if required.
h. Select **File > New > XQuery Transformation** to create the required transformation. Give a name to the transformation.

i. Select a source format type. This type should match the message payload at the OSB service side. Click **Add** to select the type.
j. Select the target message type. This type should match the Oracle BAM Prescripción data object type. Click **Add** to select the type.

![New XQuery Transformation](image)

k. Click **Finish** to close the wizard and open the XQuery editor. The editor should be used to create the transformation that converts the information in OSB format to Oracle BAM format. The transformation depends on the business itself and should be defined by business users. The following image shows a sample transformation that copies some information.
I. You can test the transformation using the **Test** tab. Copy a sample request payload and execute the transformation to retrieve an XML payload in Oracle BAM format.

5. Introduce some changes in the XQuery transformation
   a. Proceed directly to this step and ignore last one if you feel comfortable editing in XQuery.
   b. Some manual changes are required to the recently created XQuery transformation. These changes are needed to create a payload that matches the OSB BAM Feeder request. The Oracle BAM request contains additional information like the operation to be executed (insert, update, upsert, batch, remove) and the Oracle BAM data object identifier.
c. Oracle BAM data objects are easily represented in XML by placing an underscore prefix before the identifier. Using this coding, a data object called Foo transforms into <_Foo> XML, and a data object column Identificator transforms into <_Identificator> XML.

d. The following example code contains the original XQuery transformation as created in the last step.

```
declare namespace ns1 = "http://xmlns.osakidetza.net/services/RecetaElectronica";
declare namespace ns0 = "http://xmlns.oracle.com/bam";
declare namespace xf = "http://tempuri.org/OSB%20Project%201/RecetasToBAMPrescripcion/";
declare function xf:RecetasToBAMPrescripcion($recetas1 as element(ns1:Recetas))
  as element(ns0:_PrescripcionCollection) {
    <ns0:_PrescripcionCollection>
      for $Receta in $recetas1/ns1:Receta
        return
          <ns0:_Prescripcion>
            <ns0:_Centro>{ data($Receta/ns1:CODCENPRES) }</ns0:_Centro>
            <ns0:_Fecha>{ data($Receta/ns1:FECHAPRES) }</ns0:_Fecha>
            <ns0:_IdProfesional>{ data($Receta/ns1:CODPRONALTIT) }</ns0:_IdProfesional>
            <ns0:_TotalRecetas>{ data($recetas1/ns1:Receta[1]/ns1:NUMRECETAS) }</ns0:_TotalRecetas>
            <ns0:_IdReceta>{ data($recetas1/ns1:Receta[1]/ns1:CODREC) }</ns0:_IdReceta>
            <ns0:_IdProducto>{ concat($recetas1/ns1:Receta[1]/ns1:CODUAP , $recetas1/ns1:Receta[1]/ns1:CODFOR , $recetas1/ns1:Receta[1]/ns1:CODPF) }</ns0:_IdProducto>
          </ns0:_Prescripcion>
    </ns0:_PrescripcionCollection>
}
declare variable $recetas1 as element(ns1:Recetas) external;
xf:RecetasToBAMPrescripcion($recetas1)
```

```
 plead function xf:RecetasToBAMPrescripcion($recetas1 as element(ns1:Recetas))
  as element(ns0:request) {
    <ns0:request>
      for $Receta in $recetas1/ns1:Receta
        return
          <ns0:Recetas>
            <ns0:_Centro>{ data($Receta/ns1:CODCENPRES) }</ns0:_Centro>
            <ns0:_Fecha>{ data($Receta/ns1:FECHAPRES) }</ns0:_Fecha>
            <ns0:_IdProfesional>{ data($Receta/ns1:CODPRONALTIT) }</ns0:_IdProfesional>
            <ns0:_TotalRecetas>{ data($recetas1/ns1:Receta[1]/ns1:NUMRECETAS) }</ns0:_TotalRecetas>
            <ns0:_IdReceta>{ data($recetas1/ns1:Receta[1]/ns1:CODREC) }</ns0:_IdReceta>
            <ns0:_IdProducto>{ concat($recetas1/ns1:Receta[1]/ns1:CODUAP , $recetas1/ns1:Receta[1]/ns1:CODFOR , $recetas1/ns1:Receta[1]/ns1:CODPF) }</ns0:_IdProducto>
          </ns0:Recetas>
    </ns0:request>
}
declare variable $recetas1 as element(ns1:Recetas) external;
xf:RecetasToBAMPrescripcion($recetas1)
```

e. This example code contains the XQuery transformation with the changes required to build a valid OSB BAM Feeder request. The code is commented to explain the reasoning behind the change. The OSB BAM Feeder distribution contains a set of request payload that may be used to better understand how a request should be built.
f. If these changes are not performed on the XQuery transformation (or any other transformation you've created), the resulting payload will be malformed.

6. Import required resources into OSB.
   a. Once every required resource has been created, they should be imported into OSB to be used during runtime.
   b. Log into the OSB Console available at http://localhost:7021/sbconsole. Create a change session and select the project that contains the Prescription Notification Service.
   c. Some assets are required before modifying this service to use the OSB BAM Feeder. Import the Oracle BAM XML schema and XQuery transformation into the project.
      - Select **Create Resource > XML Schema** and locate the Oracle BAM XML schema for the Prescripcion data object.
      - Select **Create Resource > XQuery Transformation** and locate the XQuery transformation created in the last step.
   d. Click **Activate** to commit the changes in this session.
7. Update OSB service to publish Oracle BAM messages.
   a. Now it’s time to close the loop and publish the transformed payload to Oracle BAM using the OSB BAM Feeder component.
   b. Log into the OSB Console available at http://localhost:7021/sbconsole. Create a change session and select the project that contains the Prescription Notification Service.
   c. Edit the proxy service flow. The message should only be transformed and published to Oracle BAM under certain conditions – these conditions depend on the business and should be defined by business users. This example shows how the message is transformed and published using the Routing node.
   d. Edit the route node and add a Publish action to leverage the OSB BAM Feeder component. Click Add an action > Communication > Publish. Select Oracle/BamFeeder/ProxyServices/BamAsyncFeederLocalPS as the service to publish the message.

   ![Route to RecetasEnviado](image)

   e. The message to be published to OSB BAM Feeder needs some transformation. Otherwise, Oracle BAM Server receives a payload in OSB service format and it is rejected.
   f. Inside the Request Actions for the Publish Action, select Add an Action > Message Processing > Replace. This action is used to
replace the body with a transformed payload that matches the
Oracle BAM data object format.

- Set "." as selected XPath.
- Set `body` as the variable.
- Choose Replace node contents.

g. Click Expression to select the XQuery transformation.
   - Click XQuery Resources in the upper menu.
   - Click Browse to select the XQuery resource. Select the
     XQuery resource previously imported that transforms
     from OSB source format to Oracle BAM target format.
   - Use the Variable Structures menu options to navigate
     the request and select the Xpath expression that binds
     the XQuery input variable.
   - Click Validate and Save.
h. Click **Save All** and **Activate** to commit the changes in this session.

8. The Prescription Notification OSB Service is now intercepted and configured to publish messages to Oracle BAM using the OSB BAM Feeder. The next section describes how to test the entire solution.

### 3.3 Testing the solution

This section explains how to test the entire solution. At this point, the Oracle BAM data object has been created, and the OSB service leverages the OSB BAM Feeder to publish messages to Oracle BAM.

1. Monitor the Oracle BAM data object.
   b. Log in and select **Architect**.
   c. Select **Data Objects** from the upper-left combo box. Navigate to the Prescripcion data object and click **Contents**. You should see an empty data object if no information has been published yet.
   d. Keep this window open to monitor how this data object receives published messages from OSB.
2. Test the OSB BAM Feeder integration.
   a. The Prescription Notification service at OSB has been intercepted to publish messages to Oracle BAM using the OSB BAM Feeder component.
   b. Open the OSB Test Console for this service to execute a sample request. Edit the request or load a preconfigured request payload and click **Execute**.
c. Get back to the Oracle BAM Architect window. Click Contents to refresh the view. If the OSB BAM Feeder integration is working, and the message payload is being transformed as expected, you should now see the published items.

d. To further refine this example, an Oracle BAM Dashboard is created so that prescriptions may be monitored by business users.
To find more information about Oracle BAM dashboards, refer to the Oracle BAM 11g User's Guide.

4 Key Takeaways and Recommendations

Oracle Service Bus is one of the most valued products to solve integration issues during the initial steps of a SOA initiative. OSB is specially designed to provide solutions to deal with point-to-point integration, tight coupling and enterprise application integration.

Effective monitoring becomes a key requirement once the OSB infrastructure is in place and usage is widespread. OSB provides functionality to monitor service’s health using metrics and alerts; however, this monitoring is mostly oriented toward technical users.

Oracle Business Activity Monitoring is the preferred product to give business users the functional monitoring on their business services and processes in the enterprise. Oracle BAM provides real-time monitoring to correlate Key Performance Indicators (KPIs) that are conceptually nearer to the business domain.

Integrating OSB with Oracle BAM enables business users to leverage a huge amount of information that is being intermediated at OSB services. This information may be used fully exploit Oracle BAM capabilities and monitor business process, analyze events as they occur and act on certain conditions.

There are different technical alternatives to integrate OSB with Oracle BAM. Specific to the Oracle BAM 11g version, the preferred integration uses the Oracle BAM JCA adapter when this configuration is supported/available. Other integration scenarios, like OSB 10gR3 and 3rd party products, should leverage Oracle BAM Web services.

This paper explores the technical details regarding the OSB/Oracle BAM integration for OSB 10gR3 and Oracle BAM 11g using the Oracle BAM Web services.
services. A custom OSB configuration is designed and analyzed so that using Oracle BAM Web services from within OSB becomes very simple.

**About the Authors**

Aritza Iratzagorria is the Lead Architect for Centralized IT at Osakidetza (Basque Country Health Service). He has been working at the middleware space for 9 years in different roles – operations, system administration and corporate architecture. He is currently focused on the SOA strategy deployment at Osakidetza and has actively sponsored the SOA Center of Excellence for Osakidetza IT Departament.

aritza.iratzagorriagaray@osakidetza.net

Nacho Lafuente is a Consulting Technical Manager at Oracle SOA Consulting. He has been related to technology solutions in the middleware stack over the last 10 years and has delivered many integration solutions to customers. He is the SOA Client Architect for Osakidetza and helps this customer as a lead member of the Osakidetza’s SOA Center of Excellence.

nacho.lafuente.botella@oracle.com