

Building Extendable Oracle ADF Applications with Embedded Mule ESB

Miroslav Samoilenko

January 2008

ORACLE®
PARTNER NETWORK

Building Extendable Oracle ADF Applications with Embedded Mule ESB

During implementations of packaged software, IT consultants always face the challenge of changing the program logic. Be it an ERP application or a bug tracking solution, each customer insists on its own ways of conducting business and requires that software supports it. In many cases consultants can configure the software to meet customer's requirements. However, there are situations when consultants have to extend the software.

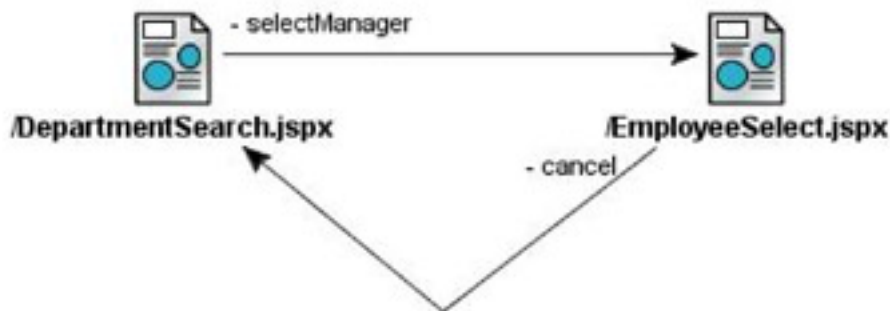
Software extensions are largely divided into two main categories: extensions and customizations. Extensions are formed by custom code which does not alter any of the standard code shipped with the software or which alters the code allowed to be changed, i.e. extension points. Customizations are formed by custom code which alters the standard code shipped with the software.

In this article I discuss a technique of building extension points into an ADF BC4J application using Mule ESB.

Building a Demo ADF BC4J/JSF Application

Oracle 10g database ships with an HR schema. This schema contains data for a simple HR application which we'll be using in our example. The schema contains DEPARTMENT and EMPLOYEE tables. Each employee is assigned to one department and each department has one manager.

We will design an ADF BC4J/JSF application which changes the manager to a department.



The application consists of two pages. The Department Search page allows you to select the department for which we want to change the manager.

Departments

Criteria

DepartmentName
 ManagerName
 LocalAddress

Results

Select and Previous 1-10 of 27 Next 10

Select	DepartmentName	ManagerName	LocalAddress
<input checked="" type="radio"/>	Administration	Lorentz, Diana	2004 Charade Rd, Seattle, Washington, 98199
<input type="radio"/>	Marketing	Hartstein, Michael	147 Spadina Ave, Toronto, Ontario, M5V 2L7
<input type="radio"/>	Purchasing	Raphaely, Den	2004 Charade Rd, Seattle, Washington, 98199
<input type="radio"/>	Human Resources	Mavis, Susan	8204 Arthur St, London, .
<input type="radio"/>	Shipping	Fripp, Adam	2011 Interiors Blvd, South San Francisco, California, 99236
<input type="radio"/>	IT	Hunold, Alexander	2014 Jabberwocky Rd, Southlake, Texas, 26192
<input type="radio"/>	Public Relations	Baer, Hermann	Schwanthalerstr. 7031, Munich, Bavaria, 80925
<input type="radio"/>	Sales	Russell, John	Magdalen Centre, The Oxford Science Park, Oxford, Oxford, OX9 9ZB
<input type="radio"/>	Executive	King, Steven	2004 Charade Rd, Seattle, Washington, 98199
<input type="radio"/>	Finance	Greenberg, Nancy	2004 Charade Rd, Seattle, Washington, 98199

Select and Previous 1-10 of 27 Next 10

Once the department is selected, you navigate to the Employee Select page. Here, you select the new manager, and either commit or rollback the transaction.

Select Employee

Criteria

FullName
 Email
 DepartmentName

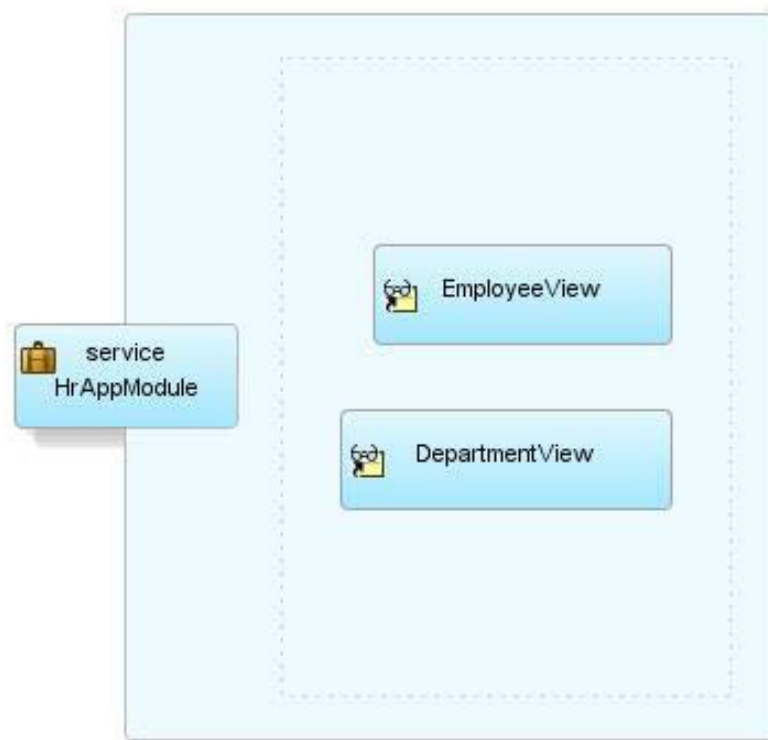
Results

Select and Previous 1-10 of 107 Next 10

Select	FullName	Email	DepartmentName
<input checked="" type="radio"/>	King, Steven	SKING	Executive
<input type="radio"/>	Kochhar, Neena	NKOCHHAR	Executive
<input type="radio"/>	De Haan, Lex	LDEHAAN	Executive
<input type="radio"/>	Hunold, Alexander	AHUNOLD	IT
<input type="radio"/>	Ernst, Bruce	BERNST	IT
<input type="radio"/>	Austin, David	DAUSTIN	IT
<input type="radio"/>	Pataballa, Valli	VPATABAL	IT
<input type="radio"/>	Lorentz, Diana	DLORENTZ	IT
<input type="radio"/>	Greenberg, Nancy	NGREENBE	Finance
<input type="radio"/>	Faviet, Daniel	DFAVIET	Finance

Select and Previous 1-10 of 107 Next 10

The pages are served by one HR service. The service contains two independent views: DepartmentView for the department search page, and EmployeeView for the employee select page. Since we are building the most generic application, we allow any employee become a manager of a department.

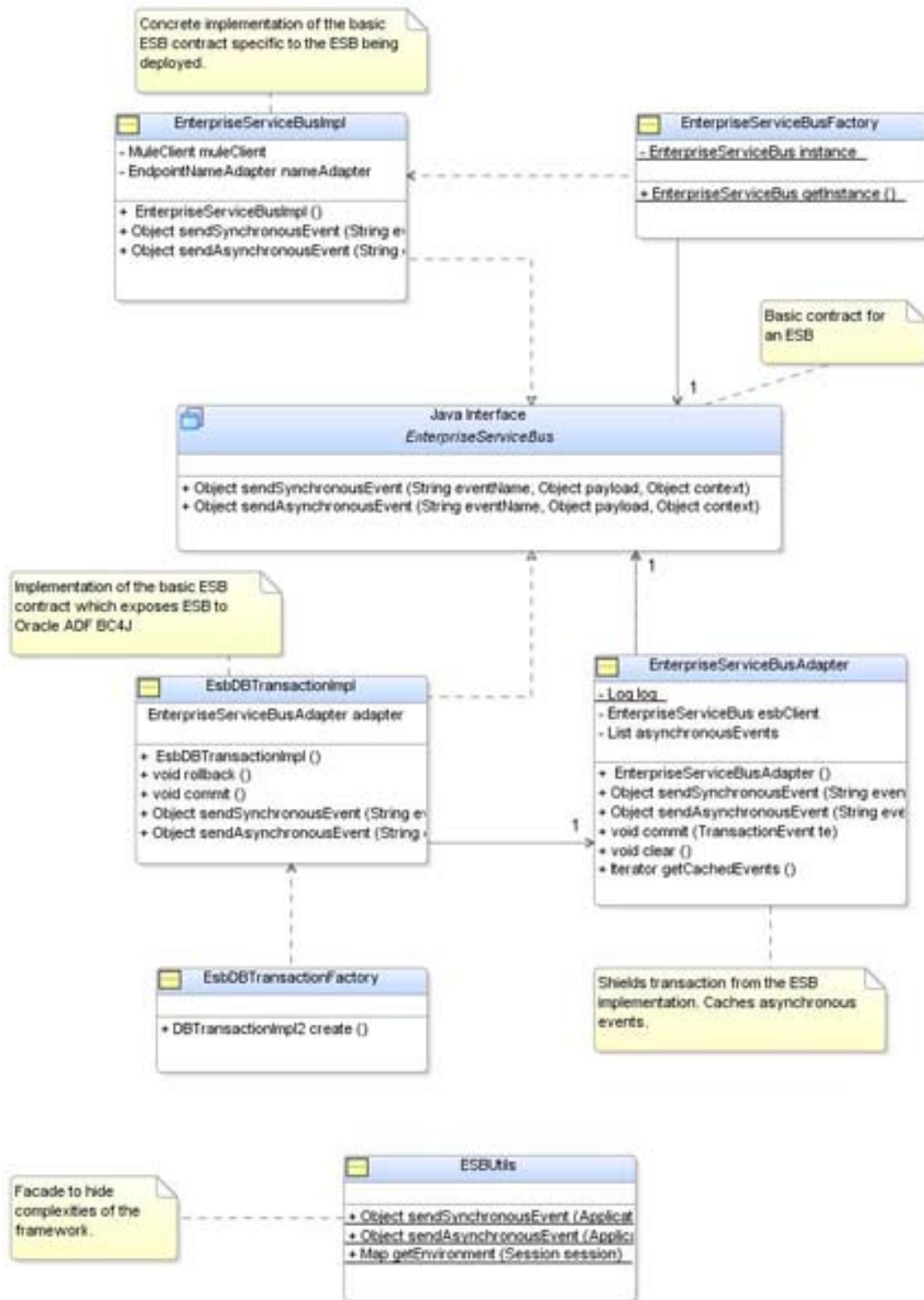


All actions on our pages bind standard operations which are provided by application modules and view objects, except for the action for the button 'Set As Manager'. Action for this button binds to a method in the HR service.

```
public void setDepartmentManager() {  
    Row departmentRow = this.getDepartmentView().getCurrentRow();  
    BigDecimal departmentId = (BigDecimal)departmentRow.getAttribute("DepartmentId");  
    BigDecimal managerId =  
(BigDecimal)getEmployeeView().getCurrentRow().getAttribute("EmployeeId");  
    departmentRow.setAttribute("ManagerId", managerId);  
    getDBTransaction().postChanges();  
    getDBTransaction().commit();  
}
```

How much flexibility does a solution like this give to the consultants implementing this software at a customer site? What if the customer insists on certain business rules about who can become a manager? For example, the same person cannot manage more than two departments? Or, the manager of the department must first be assigned to that department? Or, if the customer does not want to allow to change managers at all, or only a designated HR manager can change those?

Answers for these questions ask for an extension point inside *setDepartmentManager* method. The solution we would like to explore is throwing a synchronous event using an ESB to which consultants can subscribe new handlers. But to do this, we need to introduce an ESB to the application.



The key component of our simplified ESB framework is the interface *EnterpriseServiceBus*. This interface defines the contract which a concrete ESB implements in order to communicate with our application. In this interface we define two methods. *sendSynchronousEvent* is design to raise synchronous events. It accepts three parameters: event name, payload specific to the event, and context of execution. *sendAsynchronousEvent* is design to raise asynchronous events. It accepts the same parameters as its synchronous counterpart.

In our Oracle ADF we implement *EnterpriseServiceBus* interface at the database transaction level by extending the standard class *DBTransactionImpl2*. The implementation delegates execution of the synchronous and asynchronous events to an adapter *EnterpriseServiceBusAdapter*. The adapter executes the synchronous events immediately, while caches the asynchronous events until the database transaction commits or rolls back. Adapter also shields the actual ESB implementation from the application.

At this point we need to decide what ESB implementation we want to use. As the title of this article suggests, we decided to use Mule ESB. Once the ESB implementation is selected, we need to implement our ESB contract specific to the selected implementation. The class *EnterpriseServiceBusImpl* does the job. The following listing shows implementation of *sendSynchronousEvent* method.

```
public Object sendSynchronousEvent(String eventName, Object payload, Object context) throws Exception
```

```
{
    Object result = null;
    try {
        UMOMessage message = null;
        synchronized (muleClient) {
            Map parameters = new HashMap();
            if (context != null) {
                parameters.put("context", context);
            }
            message = muleClient.send(eventName, payload, parameters);
        }
        result = (message == null) ? null : message.getPayload();
    } catch (MalformedURLException ex)
    {
        // if end point is not registered, nothing serious.
        ;
    }
    if (result instanceof Exception) {
        throw (Exception)result;
    }
    return result;
}
```

There are two points which we would like to emphasize. If there is no handler for the passed business event, Mule throws *MalformedURLException*. Since we are shipping our application without any default event handlers for extension points, this exception will always be thrown. We decided to suppress it, since this exception does not inform us of a real problem.

The second point is exceptions thrown by custom handlers. The convention which we adopt is that the custom event handler should never throw an exception. Instead, it returns the exception as the result of execution. *EnterpriseServiceBusImpl* recognizes the result of execution and throws the

exception if necessary.

Complexity of the ESB framework is hidden from the application by façade *ESBUtils*. This class provides two methods for sending synchronous and asynchronous events from an application module. The listing below shows how synchronous events are sent.

```
static public Object sendSynchronousEvent(ApplicationModule applicationModule, String
eventName, Object payload) throws Exception {
    Transaction dbTransaction = applicationModule.getTransaction();
    if (dbTransaction instanceof EnterpriseServiceBus) {
        EnterpriseServiceBus esbClient = (EnterpriseServiceBus)dbTransaction;
        return esbClient.sendSynchronousEvent(eventName, payload, applicationModule);
    }
    return null;
}
```

As you can see from the listing, the context of execution of a synchronous event is the application module which throws it. Since we are using Mule embedded into our Oracle ADF implementation, this approach gives the custom handler access to the service which raised the business event.

Now, we are ready to add the extension point. The method to change the manager to a department now looks as follows:

```
protected void raiseEvent(BigDecimal departmentId, BigDecimal managerId) throws
JboException {
    try {
        Map payload = new HashMap();
        payload.put("DepartmentId", departmentId);
        payload.put("EmployeeId", managerId);
        ESBUtils.sendSynchronousEvent(this, "setDepartmentManager", payload);
    } catch (Exception ex) {
        throw new JboException(ex);
    }
}

public void setDepartmentManager() {
    Row departmentRow = this.getDepartmentView().getCurrentRow();
    BigDecimal departmentId = (BigDecimal)departmentRow.getAttribute("DepartmentId");
    BigDecimal managerId = (BigDecimal)this.getEmployeeView().getCurrentRow().
getAttribute("EmployeeId");
    raiseEvent(departmentId, managerId);
    departmentRow.setAttribute("ManagerId", managerId);
    this.getDBTransaction().postChanges();
    this.getDBTransaction().commit();
}
```

How do we communicate the extension points to our consultants? The more important question, how do we ensure that application patching does not affect extensions built by consultants?

The answer to these questions lies in the separation of Mule configuration files. The application ships with two Mule configuration files. One file contains endpoint definitions and handlers which are necessary for the normal execution of the application. The second Mule configuration file is an empty file. This file is designed for consultants to add their handlers to business events. This second

file is never updated by patches.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration XML V1.0//
EN"
    "http://mule.mulesource.org/dtds/mule-configuration.dtd">
<mule-configuration id="CustomerConfig" version="1.0">
  <mule-environment-properties serverUrl="">
    <threading-profile maxThreadsActive="20" maxThreadsIdle="20"
      maxBufferSize="500" poolExhaustedAction="WAIT"/>
    <pooling-profile exhaustedAction="GROW" maxActive="20" maxIdle="20"
      maxWait="-1" initialisationPolicy="INITIALISE_ALL"/>
    <queue-profile maxOutstandingMessages="5000"/>
  </mule-environment-properties>
  <agents>
    <agent name="JmxAgent" className="org.mule.management.agents.JmxAgent"/>
  </agents>
  <endpoint-identifiers>
  </endpoint-identifiers>
  <model name="CustomerHandlers">
  </model>
</mule-configuration>
```

Building an Extension

Once our application is deployed at a customer site, consultants start its configuration to fit the customer needs. One of the requirements which customer brought to the consultants team is to make sure that the employee is assigned to the department before she or he can become department manager.

Consultants investigated documentation to the application and located a synchronous business event which the application raises before it assigns the new manager to a department.

Consultants build a new service customer.demo.model.services.ExtentionHrAppModule which is designed to be a subservice to the standard HR service. The new service validates that the selected manager is assigned to the same department as the one he or she is about to head. In case of a difference, the service throws an exception.

```
public void extraValidation() {
    ApplicationModule parentModule = this.getRootApplicationModule();
    ViewObject empoyeeView = parentModule.findViewObject("EmployeeView");
    ViewObject departmentView = parentModule.findViewObject("DepartmentView");
    BigDecimal employeeDeptId = (BigDecimal)empoyeeView.getCurrentRow().
getAttribute("DepartmentId");
    BigDecimal departmentId = (BigDecimal)departmentView.getCurrentRow().
getAttribute("DepartmentId");
    if (! departmentId.equals(employeeDeptId)) {
        throw new JboException("Manager must work for the same department which he or
she is managing");
    }
}
```

Next, consultants build a Mule action handler. Since the event is synchronous, the context of the event is the root service. This root service will become the parent service to the new extension. The Mule action handler is presented in the listing below:

```
public class ExtendedDepartmentManagerMuleAction2 implements Callable{

    public Object onCall(UMOEventContext umoEventContext) {
        Object context = umoEventContext.getMessage().getProperty("context");
        if (context instanceof ApplicationModule) {
            try {
                Map payload = (Map)umoEventContext.getMessage().getPayload();
                ExtentionHrAppModuleImpl appModule = (ExtentionHrAppModuleImpl)getCustomEx-
tension((ApplicationModule)context);
                appModule.extraValidation();
            } catch (Exception ex) {
                return ex;
            }
        }
        return null;
    }

    protected ApplicationModule getCustomExtension(ApplicationModule context) {
        ApplicationModule result = context.findApplicationModule("CustomApplicationModul-
e");
        return (result == null) ? context.createApplicationModule("CustomApplicationModule",
"customer.demo.model.services.ExtentionHrAppModule") : result;
    }
}
```

All pieces come together with updated Mule configuration file. The changes to the Mule configura-
tion file is present in the following listing:

```
<endpoint-identifiers>
  <endpoint-identifier name="setDepartmentManager"
    value="vm://danko.demo.model.service.hr.setDepartmentManager"/>
</endpoint-identifiers>
<model name="CustomerHandlers">
  <mule-descriptor name="SetDepartmentManager" inboundEndpoint="setDepartmentMa-
nager"
implementation="customer.demo.model.mule.ExtendedDepartmentManagerMuleAc-
tion2">
    </mule-descriptor>
</model>
```

Since this configuration file is meant to be changed by consultants, it is guaranteed that the exten-
sion will not be overwritten by a patch.

So, now, when you assign a new manager to a department which he or she does not yet work for,
you will be getting an error message like the following example:

✖ Error

- JBO-29000: Unexpected exception caught: oracle.jbo.JboException, msg=JBO-29000: Unexpected exception caught: oracle.jbo.JboException, msg=Manager must work for the same department which he or she is managing
- JBO-29000: Unexpected exception caught: oracle.jbo.JboException, msg=Manager must work for the same department which he or she is managing
- Manager must work for the same department which he or she is managing

Select Employee Cancel

Criteria

FullName

Email

DepartmentName

Results

Select and Previous 1-10 of 107 Next 10

Select	FullName	Email	DepartmentName
<input type="radio"/>	King, Steven	SKING	Executive
<input type="radio"/>	Kochhar, Neena	NKOCHHAR	Executive
<input type="radio"/>	De Haan, Lex	LDEHAAN	Executive
<input type="radio"/>	Hunold, Alexander	AHUNOLD	IT
<input type="radio"/>	Ernst, Bruce	BERNST	IT
<input type="radio"/>	Austin, David	DAUSTIN	IT
<input type="radio"/>	Pataballa, Valli	VPATABAL	IT
<input type="radio"/>	Lorentz, Diana	DLORENTZ	IT
<input checked="" type="radio"/>	Greenberg, Nancy	NGREENBE	Finance

References:

Oracle 10g XE database, <http://www.oracle.com/technology/products/database/xe/index.html>

Oracle JDeveloper 10g, <http://www.oracle.com/technology/tech/java/jsf.html>

Mule ESB 1.4.3, <http://mulesource.org>