Integrate IBM InfoSphere Master Data Management Server using WebSphere Process Server, Part 1: Deploy and execute MDM Server Web services in a distributed environment

Using the WS-Coordination and WS-Atomic Transaction specifications

Stephanie J. Hazlewood
Allan Luo
Martin Oberhofer

January 22, 2009

In this tutorial (the first in this series), get a high-level overview of how to integrate IBM® InfoSphere™ Master Data Management (MDM) Server with another Enterprise Information System (EIS) using IBM WebSphere® Process Server.

View more content in this series

Getting started

In this series

The first tutorial in this series presents a high-level overview of how to integrate IBM InfoSphere MDM Server with another Enterprise Information System (EIS) using WebSphere Process Server. The second tutorial in this series provides a step-by-step process to create and deploy a set of MDM Server Web service calls to execute in an atomic transaction in a distributed environment using the WS-Coordination and WS-Atomic Transaction specifications. The second tutorial builds on this first tutorial to set up a long-running business process involving several service providers in a complex workflow.

Objectives

In this tutorial, learn how to:

- Enable IBM InfoSphere MDM Server to take part in distributed global transactions by making use of the WS-Atomic Transaction and WS-Coordination specifications on WebSphere Process Server
• Resolve the mismatch between client services and data types with IBM InfoSphere MDM Server by mapping business objects and services
• Create and deploy a set of Web service calls that execute in an atomic transaction in a distributed environment

Prerequisites
This tutorial is written for J2EE developers having:

• Experience and skills at an intermediate to advanced level
• Familiarity with MDM Server Web services
• Familiarity with WebSphere Integration Developer
• Some familiarity with WS-Atomic Transaction and WS-Coordination

System requirements
The tutorial is designed to be run in a distributed environment (three separate machines). Each machine should have:

• 2GB memory or more
• 10GB free disk space
• Any operating system on which WebSphere Integration Developer 6.0.2 and IBM Rational® Software Architect 7.0 can be installed

The machines will be running:

1. IBM InfoSphere Master Data Management (MDM) Server 8.0 or later (see Resources for more information about this product). This tutorial may also work with previous versions of the product, though the tutorial scenario has not been thoroughly tested with these. MDM Server will be used to provide enhanced Master Data Management functionality for other information systems.
2. WebSphere Process Server will be used with WebSphere Integration Developer 6.0.2 or later (see Resources for more information about this product). WebSphere Integration Developer is used to develop the integration business module through which MDM server is integrated with other information systems.
3. IBM Rational Software Architect 7.0 or later (see Resources for more information about this product). Rational Software Architect is used for developing the Web client to invoke the business module running on WebSphere Process Server.

Overview
Many established organizations, especially those that have undergone mergers, acquisitions, or both, end up having to manage a wide array of business data, such as customer, financial, and product profiles, across distributed EIS systems. Many of these systems typically run on diverse platforms and technologies. Synchronizing data across EISs and leveraging these existing applications to build new business processes pose a substantial challenge. Using service oriented architecture (SOA) and Web services, new enterprise services may be made available, built from the underlying diverse services. Business process choreography helps you do that. IBM offers solutions to manage both data and services across an enterprise.
IBM InfoSphere Master Data Management (MDM) Server provides the solution for managing and maintaining operational master data for multiple domains, including customer, account, and product across the enterprise. In short, MDM allows for the decoupling of master information from the applications that created it, and MDM Server provides the functionality to pull it all together to provide a single, unified view across business processes and transactional and analytical systems. As an SOA-based based product, MDM Server provides reusable and extendible standards-based services that can be consumed through multiple channels including Web services. Because master data spans across systems, platforms, and enterprise boundaries, it is important that MDM Server can be seamlessly integrated with existing systems at each or any of these levels.

IBM WebSphere Process server is a platform for service-oriented integration. It provides the components and functionality to assemble services from multiple systems, thereby facilitating the creation of composite business services. It is built on a universal invocation model (Service Component Architecture, or SCA) and data representation model (Service Data Object, or SDO). The implementation of both SCA and Business Objects (BOs) (built on SDO technology) is standards-based. That is, interfaces to any SCA service component are described using WSDL (Web Service Definition Language) or Java™ Interfaces, and BOs are described using XML Schema (XSD).

### Business integration fundamentals

The following section provides an overview of some relevant business integration concepts. Figure 1 provides an overview of how MDM services may be integrated with other EIS systems using WebSphere Process Server.

**Figure 1. Overview of MDM Server and EIS integration using WebSphere Process Server**
• Service providers such as IBM MDM Server and other EIS systems have their service definitions (WSDL for Web services, for example), imported to the ESB of WebSphere Process Server.
• Interacting with the ESB, the SOA core and supporting services serve to integrate, or choreograph, these imported services to create new composite services, for example.
• Definitions for these newly created composite service(s) may be exported by WebSphere Process Server, thereby becoming available to service consumers.

Service Component Architecture (SCA components)

It is the Service Component Architecture (SCA) that allows for component abstractions, thus simplifying the process of developing composite business transactions. For example, SCA components may consume MDM Web services and function to orchestrate them with other services from other EIS systems to create new composite services that may serve a particular business need. For instance, if financial profile information for a particular customer is stored in one EIS system and the customer demographic profile has been mastered in MDM Server, a new composite service drawing information from both of these systems will satisfy the business need to retrieve this information together. Using WebSphere Integration Developer, these SCA components may be created and combined.

SCA components consist of:

• Interfaces, which define the inputs, outputs, and faults
• Implementation (business logic)
• References, which allow for the consumption of other component functionality, such as MDM Server Web services, the functionality of other EIS systems, or both

Figure 2 shows an SCA module with a service component consuming MDM Server services, providing new composite services:
Business Objects (BOs) and interface maps

Business objects contain the attributes describing the data stored by a particular application. As depicted Figure 2, BOs are passed between the different systems, that is, the service consumer and service provider. BO maps serve to define the transformation of the application-specific business object (ASBO) of the service consumer into a generic business object (GBO). From the GBO, it is then mapped to the application-specific business object of the service provider. Thus the interface map is used to decouple the representation of the service consumer business objects and the service provider business objects so that they can be changed independently from each other. The only step that needs to be adapted is the mapping between the ASBO and the GBO if either side changes. The interface map provides the binding between the definitions of the operations of the systems and uses the BO maps to transform the parameters of these operations as necessary.

The business process

Described in the document "Web Services Business Process Execution Language (WS-BPEL)" (OASIS, April 2007) (see Resources), business processes provide a means of defining the choreography for the invocation and integration of services, such as those provided by IBM MDM Server, to define and expose new services to other EISs. WS-BPEL is a grammar that describes the order of execution and interactions between business processes and external services.

Transaction management with WS-Coordination and WS-Atomic Transaction (WS-AT)

In the scenario provided in this tutorial, there is an exchange taking place between multiple distributed participants. If any part of the transaction fails, everything done to that point must be
rolled back, and this must be coordinated between the systems involved. Two OASIS standards help meet these goals: WS-Coordination and WS-Atomic Transaction.

The WS-Coordination standard provides the framework for coordinating transactions in a distributed environment. It defines three main participants in this coordination:

- **Coordination context** – a context that gets shared, thereby permitting the coordination
  - The coordination context contains the id for the activation service
  - It defines the coordination type
  - It provides an endpoint reference to the client coordinator's registration service
- **Activation service** – used by the initiator of the coordination to create a coordination context that in turn gets propagated to the distributed participants
- **Registration service** - used by each participant in the transaction to register for the WS-AT protocol

WS-Atomic Transaction (WS-AT) is the OASIS standard that defines the protocol for supporting Web service atomic transactions in a distributed environment. Using this two-phased commit protocol, WS-AT allows for a set of Web service calls to be executed in a single atomic transaction and ensures that each of these service calls complete or roll back as a single unit of work.

See [Resources](#) for additional information about these two standards.

### WS-Coordination example

A very basic example involving the use of MDM Server and WS-AT can be found in the tutorial "[Configure and invoke Web services for WebSphere Customer Center](#)" (developerWorks, October 2007). This example does not involve the use of WebSphere Process Server. For the distributed environment set up in this tutorial, a coordination context will be propagated among the servers, thereby ensuring that a WS-AT transaction is running within the same transaction context. This tutorial demonstrates this context propagation functionality using WebSphere Process Server; the importance of this approach becomes increasingly apparent in the next tutorial when you work with setting up a long-running business process spanning several service providers in a more complex workflow.

One server (client) will run a Web service client, another server will run WebSphere Process Server, and a final server will host IBM Master Data Management Server (Web services).

A (global) transaction will be created that wraps several MDM Web service calls. The WebSphere Process Server will receive the Web service calls from the client and map the interfaces of these calls to MDM Web service interfaces, then invoke the appropriate MDM Web services. The WS-AT coordinator on each server is responsible for coordinating the atomic transaction across the servers and ensuring the global transaction functions as a single unit of work. [Figure 3](#) depicts this setup:
When the client starts the global transaction, it calls the local coordinator to create the context, including activity ID, the coordination type, and an endpoint reference to the client coordinator’s registration service. The created coordination context is sent to WebSphere Process Server module as part of the SOAP request header. The WebSphere Process Server module calls the coordinator by passing the context received from the client as the input. The coordinator of WebSphere Process Server creates the context that contains the same activity ID and coordination type as the received context from the client. WebSphere Process Server module determines the coordination protocols and then registers for a coordination protocol at the coordinator of WebSphere Process Server. The coordinators’ protocol services also exchange the endpoint references for WebSphere Process Server module and the client. Finally, the coordinator of WebSphere Process Server forwards the registration on to the registration server of the client, exchanging endpoint references, and establishes the logical connection between these endpoint references. WebSphere Process Server propagates the transaction context to MDM server with the similar processes.

In this tutorial, you will set up the distributed environment depicted in Figure 3 and then make a call to an MDM Server service (Web service). You will execute this service within the transactional context of this particular topology. Transaction success and failure/rollback will be artificially simulated using a JTA transaction to be set up in a sample client application. This will demonstrate that if there is a failure at any point in the global transaction, all of the WS-AT coordinators will roll back any actions taken to that point.
Implementation

Create WebSphere Process Server module

Follow these steps to create the WebSphere Process Server module:

1. In WebSphere Integration Developer 6.0.2, select Open File > New > Project > BusinessIntegration > Module, then click on Next.
2. Enter a module name: WPSWSATSample, then select Finish.
3. Import the MDM Web service WSDL files, as shown in Figure 4:

   **Figure 4. Imported MDM Services Interfaces**

   ![Figure 4. Imported MDM Services Interfaces](image)

   Import Common.wsdl, PartyBinding.wsdl, PartyPort.wsdl, and PartyService.wsdl from the WPSServerSideSample.zip resource provided in the Downloads section. These are the same WSDL files available with the IBM MDM Server distribution in the Samples.

Import or create the interface required by the client application

Import, as shown in Figure 5, WPSPartyService.wsdl from the WPSServerSideSample.zip resource provided in the Downloads section. The operations defined in WPSPartyService will be called by the client application.

**Figure 5. The imported client-side interface**

![Figure 5. The imported client-side interface](image)
Map the client-side ASBOs to the MDM Web service ASBOs

In most cases, when business objects are included in the module, GBOs are used. Typically, the client-side ASBOs are mapped to GBOs and the GBOs are mapped to the MDM Web service ASBOs. With this approach, if any of the ASBO changes, there is no need to change the business process. Simply re-map the ASBO to the corresponding GBO. In order to focus on the transaction context propagation, this tutorial omits the GBO implementation. Instead, you will map the client-side ASBOs to MDM ASBOs directly. In most cases, this tutorial uses the same ASBOs for both client and server side. The exception to this is the control business object, which provides an example of how to do ASBO-to-ASBO mapping. Figure 6 shows the simple ASBOs mapping, where Control2 is the client side ASBO and Control ASBO is from MDM server:

Figure 6. Mapping the control ASBOs

Create client-side to server-side interface mapping

Figure 7 and Figure 8 show how the client-side WPSPartyService.AddParty operation is mapped to the PartyService.AddParty operation provided by MDM Server:

Figure 7. Map AddParty client service and AddParty MDM Server service
Similarly to the mappings shown in Figure 7 and Figure 8, do the same for the GetParty and SearchParty services.

Assemble the business integration module

In the assembly diagram, drag the newly created WPSPartyService interface with the type of export using Web service binding. For MDM PartyService interface, the type selected should be import and implementation type should be Web Service binding. The newly created interface mapping, should have its type set to component with no implementation type. These preferences are shown in Figure 9:

Figure 9. Assembly diagram mapping of party services

Update the WebSphere Process Server module deployment descriptor

Update the deployment descriptor of the WebSphere Process Server module as follows:

1. Open the deployment descriptor of the generated EJB project for the session bean module.
2. On the Global Transaction tab, check Web Services Atomic Transaction, as shown in Figure 10:

Figure 10. Settings for the Global Transaction tab

3. Select BeanMethod for Local Transaction 2.0, as shown in Figure 10. By default, Boundary is set to ActivitySession for the local transaction settings. In this sample, a global transaction
is started in the client application. A global transaction cannot wrap an ActivitySession. If ActivitySession is selected, the global transaction will be suspended and the transaction context cannot be propagated on to MDM server.

4. Also, under the **Assembly** tab, remove the ActivitySession-related settings as shown in Figure 11:

**Figure 11. The ActivitySession-related settings under Assembly tab**

Build the client

The client application is used to invoke the Web services provided by the WebSphere Process Server module. This application simulates a call that might be made from another EIS or service consumer. **Listing 1** provides an example of an MDM service call being wrapped in a JTA transaction and simulated success/rollback actions. When the JTA transaction is started, a call to the WPSPartyService.AddParty interface on WebSphere Process Server is invoked. (Recall that this is the new composite service that you created using a number of MDM services.) With this invocation, the WS-Coordination context gets propagated from the client-side application to WebSphere Process Server and then to MDM Server. MDM Server then executes the AddParty operation as part of the AddParty service and waits for it to either commit or roll back. In this example, a point of failure is introduced in order to demonstrate the rollback functionality of the global transaction. The client finishes the transaction by sending the user-supplied input parameter for the transaction type (txType), indicating whether or not the transaction should be committed or rolled back after the service has been called.

The code in **Listing 1**, below, is a snippet taken from the Invoker.java class contained in WPSCClient.zip (see **Downloads**). It provides a sample of how the service may be invoked for testing.

**Listing 1. Sample code for testing global transaction**

```java
try {
    // Set up JTA transaction
    InitialContext cxt = new InitialContext();
    userTransaction = (UserTransaction) cxt.lookup("java:comp/UserTransaction");
    // Start the JTA transaction here
    userTransaction.begin();
    // Invoke MDM addParty Services through WPS
    resp = wpSPartyService.addParty(control, person);
    // Simulate the failure point if user choose to roll back
    if (txType.equals("rollback")
```
Run the sample

In order to run the sample application, perform the following steps:

1. Download the client application sample WPSClient.ear and the WebSphere Process Server sample WPSServerSideSample.zip from the Downloads section.
2. Import the WPSClent.ear into a new WebSphere Integration Developer workspace.
3. Import the application from the WPSServerSideSample.zip into the WebSphere Integration Developer workspace:
   1. Select File > Import > Project Interchange.
4. Open the deployment descriptor for the module. Update the transaction settings related to the activity session (see Figure 10 and Figure 11), and deploy the application.
5. The client application name is WPWSAT. Launch wsat.jsp.
6. Enter the following sample values for the fields as shown in Figure 12:
   - **WPS Service EndPoint**: http://localhost:9081/WPSWSATSampelWeb/sca/ WPSPartyServiceExport
   - **WS-AT Test Type**: Rollback
   - **Party Id**: 1001
   - **Name**: aName

**Figure 12. The new service endpoint URL**
7. Select a "WS-AT Test Type" (see Figure 12). If you select **Commit**, the client commits the user transaction and the party is added to database. If you select **Rollback**, the client rolls back the transaction and the party is not added. In the example shown in Figure 12, Rollback has been selected.

The line "Transaction has been" shows whether the transaction has been committed or rolled back after you submit the request. (See Figure 12.)

Using the client application, you can also check to ensure that the party has been added to the MDM Server database. If the WS-AT test type was "Committed", enter the party id for the party that was added and submit the request, as shown in Figure 13. In this example, the party id 1001 is used. "The party added is" line should display the party id of the party added to the database. Otherwise, the party id is not displayed. In this example, the party id does not get displayed, because the WS-AT Test Type selected in Figure 12 was Rollback.

**Figure 13. The GetParty operation**

You can use the TCP/IP monitor to track the transaction context as it gets propagated among the servers.

1. In WebSphere Integration Developer, select **Window > Preferences > Internet > TCP/IP Monitor**.
2. Click on **Add** to create a new TCP/IP monitor.
3. In the local monitoring port field, enter the port by which you want to trace the transaction context. (For this tutorial, the monitor port is set to 65534, as shown in Figure 14.)
4. In the **Edit Monitor** panel, enter the host name, port, and type for the MDM Server Web Services (also shown in Figure 14).
5. Click on **OK** to save the settings, and select **Start** to start the monitor.
6. To show the TCP/IP monitor, select **Window > Show view > TCP/IP monitor**.
7. Update the endpoint to use the port to be monitored. Go to the **Resources** perspective in WebSphere Integration Developer and update this in the PartyServiceImport1.import, found in the WPSWSATSampel project. In this case, the endpoint used is http://localhost:65534/PartyWS_HTTPRouter/Services/PartyPort.

**Figure 14. Configuration of the TCP/IP Monitor**

![Configuration of the TCP/IP Monitor](image)

With the TCP/IP monitor, you can look at SOAP messages exchanged between the systems, as shown in **Listing 2** for the WS-Coordination portion of the SOAP message:

**Listing 2. SOAP message snippet showing WS-Coordination portion**

```xml
<soapenv:Header>
  <wscoor:CoordinationContext soapenv:mustUnderstand = "1">
    <wscoor:Identifier>
      com.ibm.ws.wstx:600011b43225a88000000300000144920bc2747b4a13e1f551c8c7e2c29a72921091
    </wscoor:Identifier>
    <wscoor:Expires>0</wscoor:Expires>
    <wscoor:CoordinationType>
```

Integrate IBM InfoSphere Master Data Management Server using WebSphere Process Server, Part 1: Deploy and execute MDM Server Web services in a distributed environment
Conclusion

This tutorial showed you how IBM InfoSphere Master Data Management Web Services can be deployed and executed in a distributed environment involving other EIS systems using WebSphere Process Server. The Web service calls were wrapped in an atomic transaction (using WS-AT specification), and this tutorial demonstrated transaction context propagation in the distributed environment (WS-Coordination). The next tutorial in this series will build on this first tutorial to set up a long-running business process involving several service providers in a complex workflow.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code deployed on WebSphere Process Server</td>
<td>WPSServerSideSample.zip</td>
<td>35KB</td>
</tr>
<tr>
<td>Sample code for client application</td>
<td>WPSClient.ear</td>
<td>6000KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2009

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)