Understanding how WebSphere MQ Workflow and WebSphere Application Server Enterprise Process Choreographer can work together

Kurt Lind
Friedemann Schwenkreis

June 11, 2003

Business processes that span both MQ Workflow and Process Choreographer offer the advantages of both products and allow a smooth and staged migration from MQ Workflow to Process Choreographer. This paper gives an overview of MQ Workflow and Process Choreographer and then describes interoperability scenarios for both products. It is intended for application developers and business solution architects.

1. Introduction

Why is interoperability between WebSphere MQ Workflow and WebSphere Application Server Enterprise Process Choreographer needed? On the surface, both products are designed to provide a flexible infrastructure for executing business processes and they appear to have the same functionality. However, both products have their own strengths and thus combining them can be very useful.

Business processes that span both MQ Workflow and Process Choreographer offer the advantages of both products and allow a smooth and staged migration from MQ Workflow to Process Choreographer.

For example, if part of a business process interacts frequently with Java applications, Enterprise Java Beans (EJBs) or Web services, then you can model this part as a Process Choreographer subprocess in an MQ Workflow business process. This has the advantage of better build-time support in WebSphere Studio Application Developer Integration Edition for modeling these interactions, transactional support for EJB invocations, and transactional support for Web-service invocations using EJB or JMS bindings. You can also reduce the number of transactions needed by a business process if you model several activities as part of a non-interruptible Process Choreographer process.

Modeling Process Choreographer processes as subprocesses in an MQ Workflow master process, means that you can use the MQ Workflow monitoring tools with Process Choreographer.
To reuse your existing MQ Workflow processes in Process Choreographer, you can model them as subprocesses inside new Process Choreographer processes. This way, you can avoid migrating these processes to the Process Choreographer base.

This paper gives an overview of MQ Workflow and Process Choreographer and then describes interoperability scenarios for both products. It is intended for application developers and business solution architects.

2. Overview of MQ Workflow

MQ Workflow consists of several server components that communicate via WebSphere MQ queues. The most important server component for interoperability is the execution server. The execution server is responsible for starting and navigating business processes. Furthermore, it is the client of any service that is used on behalf of the execution of a business process.

Information about business processes, the infrastructure, and participating people (staff) is made known to the execution server by populating the Runtime database with the corresponding information. This is done by importing a process-definition-language (FDL) file using a tool that translates the FDL definitions into the entities of the Runtime database. Normally, MQ Workflow Buildtime is used to generate FDL files. However, because the FDL description is public, FDL files can also be generated in other ways.

2.1 MQ Workflow interfaces

The communication interface with the execution server is messages that are sent to a WebSphere MQ queue. The execution server can process the following message formats:

- **SDDS**
  An internal (proprietary) format of MQ Workflow messages that is generated by the client API functions.
- **XML**
  A subset of the SDDS messages in XML format. The XML messages are sent as the payload of an MQ message. The MQ message can contain additional headers, for example, a JMS header. Therefore, if you use the JMS implementation of WebSphere MQ, you can call the execution server from a JMS client.

The execution server accepts XML messages for the invocation of certain functions and it uses XML to invoke user-defined program execution servers (UPES). Thus, XML messages are well suited for interoperability with non-MQ Workflow components.

3. Overview of Process Choreographer

Process Choreographer is a business-process engine that allows for the efficient execution of business processes. The types of business processes can vary greatly, ranging from Web services navigation to business transaction support. Business processes can be automatic, recoverable processes, or processes that require human interaction. With Process Choreographer,
you can combine business process technology with any other service offered by the open J2EE architecture.

3.1 Process Choreographer architecture

Process Choreographer adds a business process container (also known as a business process engine) to the WebSphere Application Server. This allows the seamless management and efficient execution of all kinds of processes in the application server. Business processes are deployed as part of enterprise archive (EAR) files, and are configured and controlled using the WebSphere Application Server Administrative Console.

Figure 1 shows an overview of the Process Choreographer architecture.

Figure 1. Process Choreographer architecture

The process engine is the main component of Process Choreographer. It is responsible for the execution of business processes and includes the navigator, the factory, the work item manager, and the plug-ins for various domains:

- **Navigator.** Manages the state transitions for process instances and their activities. It also manages the plug-ins and invokes them whenever it needs to delegate a task.
- **Factory.** Manages the state of the objects that represent the business process. It allows data to be stored either transiently in memory as required by non-interruptible processes, or persistently in a database as required by interruptible processes for durability.
- **Work item manager.** Manages people assignments and associated queries.
- **Plug-ins.** The navigator relies on plug-ins for functionality not provided by the business process state machines. The plug-ins act as bridges to other components and systems.

The Process Choreographer functionality is externalized as an API that is available in a synchronous RPC-style EJB rendering and in an asynchronous JMS-style rendering.

The Web client has a browser-based graphical user interface based on Java Server Pages (JSP) technology. It offers the user a set of worklists containing work items on which the user can
perform queries, view further details, or perform actions. The Web client interacts with the process engine via the API.

In addition to the components illustrated in Figure 1, there are the deployment component and the administrative console. The deployment component installs EAR files that contain processes, while the administrative console allows you to control the business process container resources.

For further information on the Process Choreographer architecture, see WebSphere Application Server Enterprise Process Choreographer - Concepts and Architecture white paper.

### 3.2 Process Choreographer interfaces

Process Choreographer provides two main interfaces:

- The Process Choreographer API gives clients access to the engine. The Process Choreographer base functionality is externalized through an API. The API is available as a synchronous RPC-style API that is accessible via EJBs, and an asynchronous messaging-style API that is accessible via JMS using message driven beans (MDBs). A process-specific EJB and optionally an MDB are also generated for each business process that provide direct, type-safe access to it. They are called facades (following the "facade" design pattern).
- Plug-ins allow you to extend the engine base functionality with additional features. The process engine uses plug-ins to delegate tasks to the external world. The functionality delegated to plug-ins includes the handling of process data, the invocation of elemental operations (activities), people support, and logging changes in an audit trail.

This paper describes only those interfaces that are used for interoperability in more detail: the asynchronous API via facade MDBs and the Web Services Definition Language (WSDL) invocation plug-in via the Web Services Invocation Framework (WSIF).

The static asynchronous MDB API supports the message types call (synchronous process invocation), initiate (asynchronous process invocation), sendEvent, and forceTerminate.

A facade MDB allows you to send JMS messages in a custom format defined at build time to a dedicated queue. Receipt of the message by the MDB causes an implicit call of the associated process. On completion of the process, the response is sent back to the reply-to queue specified in the request.

For example, the generated facade MDB takes incoming messages of type PurchaseOrder and calls a new instance of the process template associated with it, passing the input message. On completion of the process, it returns the result of the process execution, which is of type DeliveryData, to the reply-to queue.

Each elemental activity in Process Choreographer has one or more implementations associated with it, which describe the operation invoked at run time on behalf of the activity. The invocation
plug-in manages the associated implementations, that is, it selects the right implementation, provides its input data, invokes the implementation, and handles its output data and possible errors.

The invocation plug-in architecture supports both synchronous (RPC-style), and asynchronous (JMS-style) invocation of operations.

Asynchronous invocations are used for invoking MQ Workflow processes. The call to the plug-in receives the operation's input, and then the plug-in issues the subprocess request as an asynchronous message. The plug-in returns the correlation ID to the process engine, which sets the state of the invoked activity to running. The engine can then continue navigation in parallel process branches. When the reply message for this request is received in the process engine's input queue, the reply is passed to another method in the invocation plug-in that retrieves the invocation results and returns them to the process engine. The engine sets the state of the activity to finished and continues navigation.

The WSDL invocation plug-in provides the default invocation mechanism for Process Choreographer. It uses WSIF to implement service invocations described by WSDL documents. WSIF uses the extensibility of WSDL to support various bindings such as, the Simple Object Access Protocol (SOAP), EJB, and JMS. The interoperability described in this paper uses the JMS binding.

The JMS binding lets you invoke asynchronous operations using JMS messages. The operation parameters are marshaled into a JMS message and sent to the operation input queue. After the request is processed, the operation response is returned as another JMS message, which gets unmarshalled and made available to the process.

The interoperability scenario uses JMS message of type `TextMessage`. That is, MQ Workflow XML messages are sent as text strings to the MQ Workflow execution server, which represents the remote service being invoked. The execution server returns the reply messages to the internal MDB of the Process Choreographer engine.

### 4. How interoperability works

Interoperability between MQ Workflow and Process Choreographer works in both directions. According to the WebSphere Studio programming paradigm, it is strongly recommended that you define the MQ Workflow processes and activities before modeling the Process Choreographer processes. This ensures that the message definitions in the MQ Workflow and Process Choreographer process models are equivalent.

#### 4.1 Invoking a Process Choreographer process from an MQ Workflow

#### 4.1.1 Process modeling

It is assumed that a Process Choreographer process that is to be invoked from MQ Workflow has been designed with this in mind. Therefore, the scenario starts by defining a process in MQ Workflow. One of the activities (`InteropAct`) will invoke a Process Choreographer process. This
activity is defined as an automated unattended activity with a UPES definition that points to a WebSphere MQ queue. In addition, you must define a program that defines the input and output data structures and the name of the invoked Process Choreographer process.

Figure 2 shows how to model a business process for interoperability.

![Figure 2. Process modeling in WQ Workflow for interoperability with Process Choreographer](image)

During a transformation step, the definitions of the activity are used to generate a WSDL file that defines the input and output message structures of the Process Choreographer business process. The WSDL file must be imported in WebSphere Studio Application Developer Integration Edition to generate the corresponding objects that represent the messages.

After the Process Choreographer process has been created, a facade MDB has to be defined. The important property of the facade MDB is the ListenerPort that must point to the queue that was used in the UPES definition for the MQ Workflow process.

**Execution**

Figure 3 shows how the Process Choreographer process is started at run time from MQ Workflow. When the MQ Workflow process is executed and the activity InteropActivity is started, the execution server sends an XML message in the MQ Workflow ActivityImplInvoke " message format to the listener queue of the Process Choreographer facade MDB.
The facade MDB reads the JMS message, extracts the payload, and transforms it into the format of the Process Choreographer message object. Then the process is invoked with the Java message object as the input parameter.

When the Process Choreographer process is finished, the output message is passed to the facade MDB which in turn converts it into a JMS message with the MQ Workflow ActivityImplInvokeResponse XML message format.

Process Choreographer sends this response message to the input queue of the MQ Workflow execution server. The MQ Workflow engine continues navigating through the business process. Because all messages in queues are written under the protection of a transaction, the complete execution is protected by a chain of transactions.

### 4.2 Invoking an MQ Workflow process from a Process Choreographer

#### 4.2.1 Process modeling

To make an MQ Workflow process available to Process Choreographer, you must export the process instance from MQ Workflow Buildtime as shown in Figure 2. The export step generates a WSDL definition that contains the message formats and a JMS binding to call the MQ Workflow engine. When you define the process in Process Choreographer, you can import the generated WSDL into WebSphere Studio Application Developer Integration Edition to define an activity.

#### Execution

Figure 4 shows how the MQ Workflow process is started at run time from Process Choreographer. When the Process Choreographer process is executed and the activity InteropActivity is started, the WSDL invocation plug-in transforms the Process Choreographer message object into an XML
message with the MQ Workflow `ProcessTemplateExecute` message format. Subsequently, the XML message is sent to the input queue of the MQ Workflow execution server.

![Diagram showing the invocation of an MQ Workflow process from a Process Choreographer process](image)

**Figure 4: Invocation of an MQ Workflow process from a Process Choreographer process**

The MQ Workflow execution server executes the process and returns an XML message in the MQ Workflow `ProcessTemplateExecuteResponse` format to the reply queue that was specified in the request message (the Process Choreographer engine's internal MDB queue).

Process Choreographer receives the response and the WSDL invocation plug-in transforms it into the Process Choreographer message object. The navigation of the Process Choreographer process continues.

The whole execution is protected by a chain of transactions.

© Copyright IBM Corporation 2003
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)