bostech™ corporation

ChainBuilder® ESB Transaction Support White Paper

Bostech Corporation
April 2008

Table of Contents

About This Document

This white paper provides a technical overview of transaction support in Bostech ChainBuilder® ESB, a comprehensive Service Oriented Architecture development and runtime environment based on the Java Business Integration (JBI) standard. This document describes the concept of transactions in ChainBuilder ESB, transaction support in JDBC, JMS, Transformer and ETL components followed by a case study of how transaction support is applied to a real-world example.

The audience for this document includes potential customers who will be evaluating the ChainBuilder ESB product for their application integration, data integration or Service Oriented Architecture (SOA) development.

Introduction

Transaction support is an important requirement for an enterprise service bus. A transaction is a set of operations executed as a single unit. When a transaction has the characteristic of either being completed in its entirety or not at all, it is considered indivisible or atomic. An atomic transaction type guarantees transaction integrity such that any partial updates are rolled back automatically in the event of a failure during the transaction update. Transaction integrity is especially critical in the financial industry or any application where money exchanges hands. Loss of transaction integrity can have detrimental effects to company business and customer satisfaction.

ChainBuilder ESB is implemented in 100% Java. Java and J2EE provide extensive support of transactions. Two types of transactions (local transactions and global transactions) are supported in J2EE. A third type of transaction, compensating transactions, is used to ensure transaction integrity when an enterprise service bus interacts with non-Java applications or systems.

Local Transactions

A local transaction represents a unit of work on a single connection to a data source managed by a resource manager. Relational DBMS and JMS servers are examples of resource managers.

Each resource manager offers a specific API to support local transactions. In JDBC, a connection call of `conn.setAutoCommit(false)` will disable transaction auto commit. Developers can use commit or rollback methods to control the transaction. In JMS, developers can use the JMS API to get a transacted session and call similar commit or rollback methods for the transaction objects.

Local transaction is supported in both Java Standard Edition and J2EE.

Global Transactions

When an application updates multiple data sources, a transaction manager is needed to coordinate the updates to multiple resource managers. This type of transaction is called a global transaction.

J2EE defines the Java Transaction API (JTA) to support global transaction. A Java application uses JNDI to get a JTA `UserTransaction` object and calls the BEGIN, COMMIT and ROLLBACK on the transaction object. The JTA UserTransaction can be used to implement a global transaction between JDBC, EJB or JMS.

Compensating Transactions

For many legacy systems or even web services, there is no specific standard technology available to participate in a global transaction using JTA. Fortunately, this type of application or system often supports compensating transactions. In the compensating transaction model, an early business activity can be semantically reversed by executing a compensating transaction. For example, when a trade order is placed into a stock trading system, the order is effectively removed from the system when a delete order is received and posted.

Compensating transactions allow legacy systems to be included in a global transaction to ensure transaction integrity across the multiple enterprise systems.

Transaction Support in ChainBuilder ESB

ChainBuilder ESB provides extensive transaction support to ensure transaction integrity.

When ChainBuilder ESB is deployed into a Java Standard Edition environment, Java local transactions are used in the components. If ChainBuilder ESB is deployed in a J2EE environment, such as IBM Websphere Application Server (WAS) or JBoss App Server, a global transaction can be used for transactions across multiple data sources using JDBC, JMS or EJB.

ETL Component Transaction Support

The ETL component can be used to perform database queries and updates without writing SQL code. The ETL component is implemented using the JDBC API.

The ETL consumer provides the following transaction settings:

- **Single with AutoCommit on** - The JDBC driver's AutoCommit is set to true. The transaction will not be rolled back even if there is an error.
- **Batch with AutoCommit off** – The JDBC driver's AutoCommit is set to false. A batch of database updates will be committed or rolled back depending on if an error occurs at the provider endpoint.

The ETL provider supports the following transaction settings based on message metadata:

- **Etl.Provider.StartTransaction:** If set to true, the component starts a new transaction for the message exchange.
- **Etl.Provider.EndTransaction:** If set to true, it will mark the end of a transaction. If no errors occurred since the start transaction message was received, then the transaction will be committed. If at least one error occurred, then the transaction will be rolled back.
- **Etl.Provider.CommitTransaction:** If set to true, the component commits the current transaction.
- **Etl.Provider.RollbackTransaction:** If set to true, the component rolls back the current transaction.

JDBC Component Transaction Support

The JDBC component uses an XML-based message interface to execute SQL statements against a relational DBMS using the JDBC API.

The following are the transaction request messages supporting JDBC transactions:

- **BEGIN** - Starts a new transaction for the session.
- **COMMIT** - Commits pending operations for the session.
- **ROLLBACK** - Rolls back the pending operations for the session.
- **END** - Indicates the end of a transaction for the session. If all operations since the BEGIN transaction statement were executed successfully, then the transaction is committed. If one or more operations failed since the BEGIN, then it is rolled back.

The following are examples of XML request messages that can be processed by the JDBC component. The example illustrates a single transaction which updates two fields (SALARY and QUANTITY) in two database tables (PERSON and ORDER).

```
<jdbc_request xmlns="http://cbesb.bostechcorp.com/jdbc/1.0">
   <transaction>BEGIN<transaction>
</jdbc_request>

<jdbc_request xmlns="http://cbesb.bostechcorp.com/jdbc/1.0"
   sessionId="2399823459">
   <execute>
      <statement>UPDATE Person SET Salary = ?
WHERE LastName = ?
</statement>
      <vars>
         <var mode="IN" datatype="INTEGER">25000</var>
         <var mode="IN" datatype="VARCHAR">Smith</var>
      </vars>
   </execute>
</jdbc_request>

<jdbc_request xmlns="http://cbesb.bostechcorp.com/jdbc/1.0"
   sessionId="2399823459">
   <execute>
      <statement>UPDATE Order SET Quantity = ?
WHERE orderId = ?
</statement>
      <vars>
         <var mode="IN" datatype="INTEGER">15</var>
         <var mode="IN" datatype="VARCHAR">Smith01</var>
      </vars>
   </execute>
</jdbc_request>

<jdbc_request xmlns="http://cbesb.bostechcorp.com/jdbc/1.0"
   sessionId="2399823459">
   <transaction>END<transaction>
</jdbc_request>
```

JMS Component Transaction Support

In the JMS component, the JMS consumer mode has a transaction property setting of "Transactional".

When the Transactional property is set to "yes", the JMS component creates a JMS transacted session. The JMS component reads a message from a JMS queue and creates and sends a JBI message exchange to the destination endpoint. The JMS transaction is committed when the message exchange is returned. If an error occurs at the destination endpoint, the JMS transaction is rolled back.

Transformer Component Transaction Support

The Transformer component provides the JDBC operation for database queries and updates, by a transaction setting, "AutoCommit". When the "AutoCommit" is set to false, the transaction will be committed when all map operations complete successfully, otherwise, the transaction will be rolled back. When the "AutoCommit" is set to true, the transaction will be committed regardless of detected errors.

Transaction Coordinator Component

The Transaction Coordinator component is a custom component to coordinate transactions among multiple service components. It is responsible for sending the BEGIN transaction, END transaction, COMMIT or ROLLBACK messages to the components that support transactions (e.g., JDBC and ETL). The Transaction Coordinator is also responsible for invoking a compensating transaction to an external system when supported. If the Transaction Coordinator fails to invoke a compensating transaction, the message is saved into the error database. An alert may be triggered for human intervention, assisting with extreme cases such as a fatal network error or system error has occurred.

The Transaction Coordinator is an implementation-specific component.

Case Study

Business Case

This business case is when transaction integrity needs to be ensured:

- The external *OrderWebApp* sends an order request to the enterprise service bus
- The enterprise service bus invokes multiple JDBC updates to the *OrderDB* database
- The enterprise service bus invokes a web service call to the back-end *OrderSys* system.

If the invocation to the back-end *OrderSys* fails, the updates to the *OrderDB* must be rolled back to ensure transaction integrity

ChainBuilder ESB Solution

The following diagram shows the solution implemented by ChainBuilder ESB with a description of the flow below the diagram.
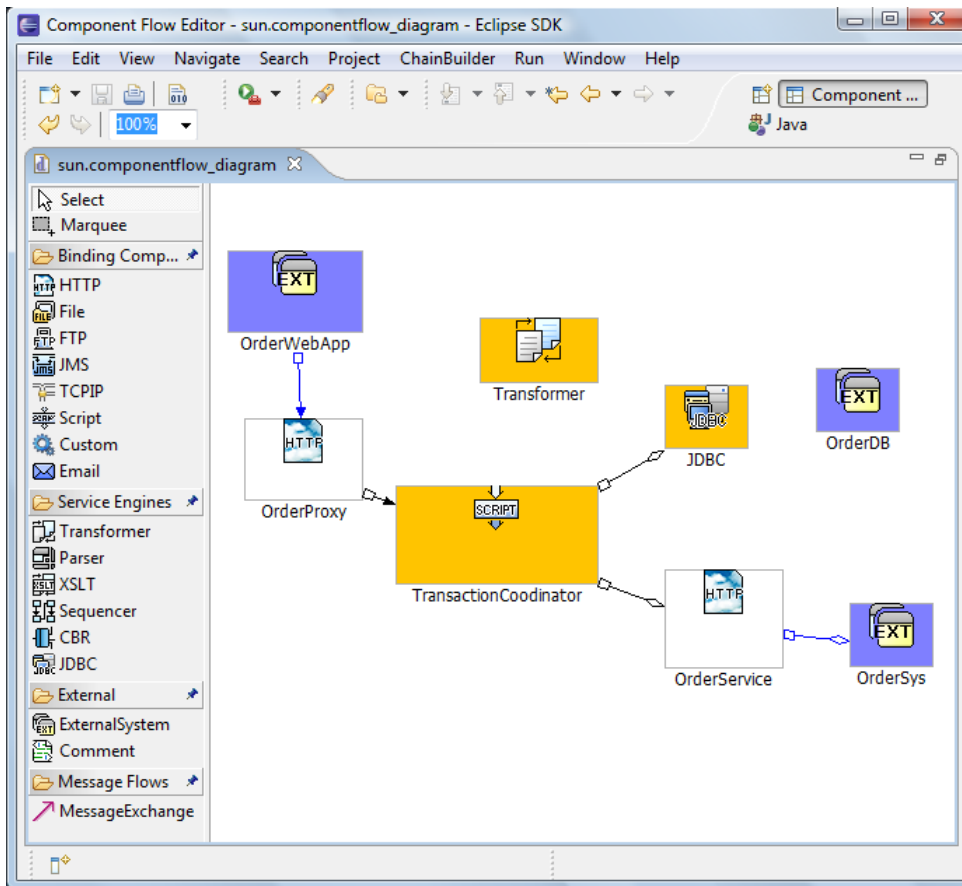
ChainBuilder ESB processes the incoming request and ensures the transaction integrity between database updates and the back-end web service invocation, as illustrated in the following steps:

1. The *OrderProxy* web service server receives an order from the *OrderWebApp.*
2. The Order message is passed to the *TransactionCoordinator* custom component.
3. The *TransactionCoordinator* component creates a JDBC request message that contains the begin transaction and SQL updates. It passes the JDBC request to the *JDBC* component. The *JDBC* component executes the SQL updates to the *OrderDB* database and returns the status.
4. The *TransactionCoordinator* component creates a new Order message based on the original Order message received from *OrderWebApp*. The new Order message is passed to the web service *OrderService* component to invoke the back-end *OrderSys* system and returns the status.
5. The *TransactionCoodinatior* component checks the status from the *OrderService*. If the status is a success, the *TransactionCoordinator* component commits the transacation by createing a JDBC request message that contains a JDBC commit and sending it to the *JDBC* component. Otherwise, the *TransactionCoodinator* component rolls back the transaction by creating a JBDC request with a JDBC rollback and sending it to the JDBC component.
6. The *TransactionCoodinator* component creates a result for *OrderProxy* and returns it back the calling *OrderWebApp*.

For more Information

To learn more about the ChainBuilder ESB, please send your inquiry to Bostech Corporation at info@bostechcorp.com, or visit Bostech's Contact Us page at: http://www.bostechcorp.com/ContactUs.