On demand business process life cycle

Create key performance indicators

Naveen Sachdeva  Joshy Joseph

Learn how to receive the Common Base Events (CBE) using Event Access and Event Distribution services provided by Common Event Infrastructure (CEI) that comes with IBM® WebSphere® Business Integration Server Foundation V5.1.1. Learn, also, how to use an object model for event-to-KPI (Key Performance Indicators) mapping to create KPIs from events, as demonstrated in this article.

Introduction

In this article, we show how to receive events using Event Access and Event Distribution services as provided by CEI and to create a business process monitoring application.

Create KPIs

In order to create the KPIs from the events, first create an object model to represent the event-to-KPI mapping. You can use this object model to create and capture the KPIs. Then create the appropriate EJBs to retrieve the events and create the KPIs using the object model. In this section, we will show the object model for OTMPS and methods for retrieving events.

OTMPS Metric Model

As shown in Figure 1, the created object model captures the required metrics for the KPI specifications. The model contains three layers: events, metrics, and collections.

- The event layer represents the events on which the metrics have a dependency. These events encapsulate the corresponding CBEs.
- The metric layer represents the process metrics of interest. Process metrics use the events in the event layer to compute the metric. For example, ThroughPut metric uses the OTMPS start, stop, and orders data events to compute the process throughput.
- The collection layer represents process metric collections, which calculate the required KPIs. For example, ThroughPutCollection is used to compute the average throughput. The
collection classes are singleton classes, which are populated by the EJBs using the events retrieved from the event server and are later accessed by the front end through a session facade to display the KPIs.

Figure 1. Metric model for OTMPS

Listing 1 and Listing 2 show the code for the ThroughPut and ThroughPutCollection classes respectively.

Listing 1. Throughput Metric class

```java
public class ThroughPut extends Metric {
    private OTMPSStartEvent startEvent;
    private OrdersDataEvent ordersEvent;
    private OTMPSStopEvent stopEvent;
    private int ordersProcessed;
    private long processingTime;
    private double throughPut;

    public ThroughPut(OTMPSStartEvent _startEvent, OrdersDataEvent _ordersEvent, OTMPSStopEvent _stopEvent) {
        startEvent = _startEvent;
        ordersEvent = _ordersEvent;
        stopEvent = _stopEvent;

        processingTime = stopEvent.getCreationTime() - startEvent.getCreationTime();
        if (stopEvent.isCompletionSuccessful()) {
            ordersProcessed = ordersEvent.getNumberOfOrders();
        }
        throughPut = ordersProcessed / processingTime;
    }

    // getters
```
Listing 2. ThroughPutCollection class

```java
public abstract class MetricCollection {
    protected Vector metrics;
    protected MetricCollection() {
        metrics = new Vector();
    }
    public void addMetric(Metric metric) {
        metrics.add(metric);
    }
}
public class ThroughPutCollection extends MetricCollection {
    public static ThroughPutCollection eINSATNCE = new ThroughPutCollection();
    public double getAverageThroughPut() {
        double totalNumberOfOrdersProcessed = 0;
        double totalProcessingTime = 0;
        for (int i = metrics.size(); i > 0; i--) {
            totalNumberOfOrdersProcessed =
                totalNumberOfOrdersProcessed +
                ((ThroughPut)metrics.get(i-1)).getOrdersProcessed();
            totalProcessingTime = totalProcessingTime +
                ((ThroughPut)metrics.get(i-1)).getProcessingTime();
        }
        return (totalNumberOfOrdersProcessed==0?0:totalNumberOfOrdersProcessed/totalProcessingTime);
    }
}
```

Retrieve Events

In order to create the metrics, you need the events emitted by OTMPS. CEI provides two interfaces to receive events: Event Access interface and Event Distribution interface. Event Access interface allows the event consumers to query the historical events synchronously. Event Distribution interface is a Java Messaging Service (JMS) interface that allows the event consumers to receive events asynchronously, as they are emitted. For more information on developing an event consumer, see `Resources`.

We decided to obtain the historical events using the Event Access interface and retrieve the new events using the Event Distribution interface. In this section, we illustrate how to query the historical events using the XPath and how to define the event groups and use them with the Event Access interface. We will also show how to implement the message-driven bean (MDB) to receive the events from the Event Distribution Interface.

Event Selectors and Event Groups

An event selector is a regular expression that defines a set of events based on the property data (attributes or sub elements) of these events. For example, an event selector might specify all of the events from a particular host with a severity that is greater than 30 (warning). Because the CBE specification is based on XML, event selectors are written using a subset of the XPath syntax. For details on XPath restrictions and writing event selectors, see `Resources`.

An event group defines a logical collection of events by associating an event selector with a unique name. Event groups can be used to query events. Event groups can also be associated with a JMS topic for asynchronous event distribution.
Event Access interface

Events can be queried using the Event Access interface, which is implemented by the EventAccess session bean. An instance of EventAccess bean can be created by using the corresponding home interface, as shown in Listing 3.

Listing 3. Create an instance of EventAccess bean

// use home interface to create remote event access bean
InitialContext context = new InitialContext();
Object eventAccessHomeObj = context.lookup("ejb/com/ibm/events/access/EventAccess");
EventAccessHome eventAccessHome = (EventAccessHome)
    PortableRemoteObject.narrow(eventAccessHomeObj,
        EventAccessHome.class);
EventAccess = (EventAccess) eventAccessHome.create();

The Event Access interface supports the following ways for querying events:

- By global instance identifier. Get the event using its globalInstanceId event property that can be used as a primary key.
  ```java
  CommonBaseEvent event =
  eventAccess.queryEventByGlobalInstanceId(eventId);
  ```

- By event group. Retrieve events that belong to a specified event group and (optionally) meet a criterion that is specified using an event selector. See Event Selectors and Event Groups for more information.
  ```java
  The ascendingOrder parameter is used to order events in either ascending or descending order, according to creation time. The maxEvent parameter is used to specify the maximum number of events to be returned from the event query.
  ```

- By association type. Retrieve the associated events by associationType for a given event. An event may have other related or associated events. An associated type attribute defines the relationship type, such as: contains, cleared, causedBy, multiPart, or correlated.

Event Distribution interface

If the event distribution service is enabled (default) and an event matches an event group, it is distributed to the queues configured for the event group. By default, CEI defines the "CommonEventInfrastructure_AllEventsTopic" with JNDI name "jms/cei/notification/AllEventsTopic" and associates it with the default event group "All events".

An event consumer can implement a message-driven bean (MDB) and associate it with the appropriate JMS destination to receive events. As shown in Listing 4, CEI also provides a notification helper that the MDB can use to convert the received JMS message into an instance of CBE.
Listing 4. Use notification helper to convert the JMS message into CBE

```java
// Get notification helper factory from JNDI
InitialContext context = new InitialContext();
Object notificationHelperFactoryObject =
    context.lookup("com/ibm/events/NotificationHelperFactory");
NotificationHelperFactory nhFactory = (NotificationHelperFactory)
    PortableRemoteObject.narrow(notificationHelperFactoryObject,
        NotificationHelperFactory.class);

// Create notification helper
NotificationHelper notificationHelper =
    nhFactory.getNotificationHelper();

// Currently CREATE_EVENT_NOTIFICATION_TYPE is the only message type
int msgType = notificationHelper.getNotificationType(msg);
if(msgType == NotificationHelper.CREATE_EVENT_NOTIFICATION_TYPE)
{
    CommonBaseEvent event = notificationHelper.getCreatedEvent(msg);
    if (event != null) {
        // Process the event
        ....
    }
}
```

Steps for retrieving events

1. Create event groups: Create event groups using WebSphere Administration Console and associate them with the appropriate JMS destinations, if required.
2. Query historical events: Write a session bean or servlet to query the historical events by event groups using Event Access interface.
3. Receive new events: Write a message-driven bean to receive new events using Event Distribution interface and merge them with the historical events.

Step 1: Create event groups

In WebSphere Business Integration Server Foundation V5.1.1, event group profiles can be created using the administrative console. In the admin console, select Resources > Common Event Infrastructure Provider > Event Group Profile List > Event groups list > Event Group Profiles > New. The event group profiles can also be created using the "Integration Test Environment" WebSphere, Version 5.1 server in WebSphere Studio Application Developer Integration Edition V5.1.1, as shown in Figure 2. But make sure that you select Enable the Common Event Infrastructure while creating the server, as shown in Figure 3. This will automatically install CEI with default configuration on the server.
Figure 2. Create event groups using WebSphere Administration Console
Figure 3. Enable CEI while creating the "Integration Test Environment" server

Note: In WebSphere Studio Application Developer Integration Edition V5.1.1, CEI "All Events Topic Connection Factory" (CommonEventInfrastructure_AllEventsTopicCF) is created with Port set to DIRECT. Make sure you change it to QUEUED, as shown in Figure 4; otherwise, the event notification and event creation will consequently fail.
WebSphere Business Integration Server Foundation V5.1.1 stores the event group definitions in the `<WAS_INSTALL_DIR>/config/cells/<cell_name>/resource-cei.xml` file. In the case of WebSphere Studio Application Developer Integration Edition V5.1.1, this file can be found in the `<WORKSPACE_DIR>/Servers/ <server_config_name>.wsc/cells` directory.

For OTMPS, we created the following event groups:

- **ProcessStartEventGroup**: `CommonBaseEvent[extensionName='WPC:ProcessinstanceEvent' and situation[@categoryName='StartSituation']]`
- **ProcessStopEventGroup**: `CommonBaseEvent[extensionName='WPC:ProcessinstanceEvent' and situation[@categoryName='StopSituation']]`
- **OrdersDataEventGroup**: `CommonBaseEvent[extendedDataElements[@name='WPCEventCode' and @type='string' and values='21090'] and extendedDataElements[@name='elementName' and @type='string' and values='orders']]`
- **InvalidOrdersEventGroup**: `CommonBaseEvent[extendedDataElements[@name='WPCEventCode' and @type='string' and values='21090'] and extendedDataElements[@name='elementName' and @type='string' and values='invalidOrders']]`

**Step 2: Query historical events**

You can query the historical events during the application startup, so that the historical events are available before you receive any new events. For that purpose, create a WebSphere startup bean and query the historical events in the start() method. (For more information on creating a WebSphere startup bean, refer to the "WebSphere Business Integration Server Foundation V5.1..."
Handbook," listed under in Resources.) First obtain the process start events, then use them to query the related events, and create the corresponding metrics. For example, in order to query the start events for the OrderProcessor process, use the StartEventGroup and further qualify it with an event selector to get OrderProcess start events only. Then use these events to get the completion, orders data, and invalid orders data events by using the corresponding event group and qualifying it with the process context, obtained from the process start events. Based on the OTMPS metric model (refer to Figure 1), create the ThroughPut, Outcome, and NumberOfInvalidOrders metrics using the appropriate events and add them to the corresponding singleton collection classes. Listing 5 shows the different event groups and event selectors that we used. Listing 6 shows the complete code for the startup() method.

### Listing 5. Event selectors for OTMPS

```java
class EventSelectors {
    static String START_EVENT_GROUP = "ProcessStartEventGroup";
    static String STOP_EVENT_GROUP = "ProcessStopEventGroup";
    static String ORDERS_DATA_EVENT_GROUP = "OrdersDataEventGroup";
    static String INVALID_ORDERS_DATA_EVENT_GROUP = "InvalidOrdersDataEventGroup";
    static String MILESTONEMANAGER_EVENT_SELECTOR =
        "CommonBaseEvent[extendedDataElements[@name = 'processTemplateName' and
        @type='string' and values = 'OrderProcessSystem']]";
    static String EXCEPTIONHANDLER_EVENT_SELECTOR =
        "CommonBaseEvent[extendedDataElements[@name = 'processTemplateName' and
        @type='string' and values = 'ExceptionHandler']]";
    static MessageFormat contextEventSelector = new MessageFormat
        ("CommonBaseEvent\[contextDataElements[@name="ECSCurrentID" and
        @type="ECSID" and @contextValue="{0}"]\]"");
    static MessageFormat validationSubprocessEventSelector = new
        MessageFormat ("CommonBaseEvent\[contextDataElements[@name="\"ReleaseMileStone\"\n        and @contextValue="{0}\"]\]");
}
```

### Step 3: Receive new events

In order to receive new events, associate the StopEventGroup with a JMS destination and write a message-driven bean to receive the new stop events. We chose to receive the stop events because we only wanted to calculate the performance metrics if the corresponding process completes successfully and is not cancelled or terminated. Moreover, by receiving the stop events, we could avoid keeping the events in memory and waiting for related events. Listing 7 shows how we handled the new stop events. We used the stop event to get the corresponding start event and then obtained the related events, if needed.

### Listing 7. Receiving new events

```java
if (cbe != null) {
    // get the context id from the start event
    String contextId = Event.getContextId(cbe);
    //complete the context event selector
    Object [] args = {contextId};
    String contextEventSelector =
        EventSelectors.contextEventSelector.format(args);
```
Display KPIs

In order to expose the collection classes, you can create a session facade and access it using a servlet, which forwards the results to a JSP. Figure 5 shows the interaction diagram for KPI monitoring. When the OTMPSMetrics servlet is invoked, it finds the MetricsFacade session bean and calls the get<KPI>() function on it (for example, getAverageThroughput), for each KPI. MetricsFacade in turn calls the corresponding singleton collection class to get the KPI. After getting all the KPIs, the OTMPSMetrics servlet forwards the information to the DisplayOTMPSMetrics JSP, which presents the information in the browser.

Figure 5. Display KPIs
You can also integrate the KPIs with the WebSphere Portal Server using the WebPage portlet, which can be configured to invoke the OTMPSMetric servlet. Figure 6 shows the OTMPS KPIs in a portal view.

**Figure 6. Portal View**

![OTMPS Metrics](image)

**Conclusion**

In this article, we showed how to create customized Business Process Monitoring applications using CEI. We illustrated how we created an event-to-KPIs object model mapping and showed how to use such a model to create KPIs from events. We also showed how to use the Event Access and Event Distribution interfaces together to effectively receive events, how to correlate events to create the KPIs, and how to display them in a portal.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample EAR file for BPM app</td>
<td>ws-odbp8code.zip</td>
<td>309 KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2005  
**Trademarks**  
(www.ibm.com/developerworks/ibm/trademarks/)