Dynamic message aggregation in WebSphere Message Broker

Sandipan Chakraborty
Amit Upadhyaya

June 05, 2013

This article shows you how to manipulate the default behaviour of the WebSphere Message Broker Aggregation node to pass a new control message from a Fan-in flow to a Fan-out flow in order to achieve dynamic message splitting and aggregation.

Introduction

IBM® WebSphere® Message Broker has built-in Aggregation Control, Aggregation Request, and Aggregation Reply nodes to facilitate Fan-in/Fan-out message aggregation. While built-in nodes provide static Fan-in/Fan-out aggregation, you need to know before runtime the number of destination applications participating in the aggregation, and you need to provide the address of each destination to the MQOutput node before the Aggregation Request node. However, in some integration scenarios, the number of requests to destination applications is only determined at runtime, and therefore conventional Message Broker Aggregation nodes will not work.

Conventional design using Aggregation nodes

Figure 1

![Diagram of conventional design using Aggregation nodes]
Problem statement

The design challenge is to split a single incoming request from a consumer application into multiple output requests to the same or different provider applications. The number of requests to the back end can vary from 1 to N depending on the payload. The new design must be flexible enough to route, transform for 1 to N different target systems, and then aggregate the response. Here are the design goals:

- **Simple** -- It must be simple enough to understand and adopt, while still demonstrating the solution.
- **Extensible** -- It must be extensible and customizable for specific customer requirements.
- **Memory efficient** -- It must cache only the required data.
- **Dynamic** -- It must allow segregation and aggregation for a variable number of messages determined at runtime.

Dynamic message aggregation using Aggregation nodes

In dynamic aggregation, the message flow takes control of the default aggregation mechanism (static Fan-in/Fan-out) implemented in the broker to facilitate dynamic aggregation (Fan-in/Fan-out). Here are the design changes to facilitate dynamic aggregation (Fan-in/Fan-out):

- Logic to split and dispatch multiple request messages to the Aggregation Control node
- Only one Aggregate Request node in a Fan-out flow
- A Compute node connected to the control terminal of the Aggregate Control node to capture and cache the individual control message of the request messages
- Configuration of Fan-out message flows to capture individual control messages for each sub-request message
- Configuration of Fan-out message flows to create a single control message from cached individual control messages

In the example below, for the sake of simplicity, the target system is same for all requests.

Prerequisites

Before starting the development, complete the following steps:

1. Set the system environment variable MQSI_AGGR_COMPAT_MODE with the value ON to send a control message. For more information see Using control messages in aggregation flows in the WebSphere Message Broker information center.
Dynamic message aggregation in WebSphere Message Broker

Developing the solution

1. Create a new application named DynamicAggregationAppln.
2. Create a new message flow named DynamicAggregationMsgFlow:

   Figure 4

3. Click on the created message flow, place an MQ Input node on the message flow canvas, and name it SAMPLE,MSG.IN. On the Properties tab, provide the following details:
4. Place a Compute node on the message flow canvas and name it CopyMsgToEnv. Connect the Out terminal of the MQ Input node to In terminal of the Compute node. Then click on the Compute node and insert the following code:

```java
DECLARE msgBodyRef REFERENCE TO InputBody;
CREATE FIELD Environment.Orig.MsgBody FROM msgBodyRef;
DECLARE retRef REFERENCE TO Environment.Orig.XML.Test;
DECLARE iterRef REFERENCE TO Environment.Orig.XML.Test;
DECLARE icnt INTEGER 0;
WHILE (icnt < 2) DO
    PROPAGATE TO TERMINAL 'out1' FINALIZE NONE DELETE NONE;
    SET icnt = icnt + 1;
END WHILE;
RETURN TRUE;
```

5. Place an Aggregate Control node on the message flow canvas. Connect the Out1 terminal of the Compute node to In terminal of Aggregate Control node. On the Properties tab of the Aggregate Control node enter the Aggregate Name DynAggr:

**Figure 6**

6. Place a Compute node on the message flow canvas and name it CombineAggrCnt. Connect the Control terminal of the Aggregate Control node to the In terminal of the Compute Node. Click on the Compute node and insert the following code:
IF (FIELDNAME(Environment.AggrFolder) IS NULL) THEN
  CREATE FIELD Environment.AggrFolder;
  CREATE LASTCHILD OF Environment.AggrFolder DOMAIN 'XML';
END IF;

DECLARE cntRef REFERENCE TO InputRoot.XML.ComIbmAggregateControlNode;
IF (LASTMOVE(cntRef)) THEN
  CREATE LASTCHILD OF Environment.AggrFolder.XML NAME 'AggrReq';
  DECLARE AggrRef REFERENCE TO Environment.AggrFolder.XML.AggrReq[<1];
  CREATE LASTCHILD OF AggrRef NAME 'AggrControlNode';
  DECLARE AggrCntlRef REFERENCE TO Environment.AggrFolder.XML.AggrControlNode FROM cntRef;
  -- Get the Data from Env for AggrReq Node
  CREATE LASTCHILD OF AggrRef NAME 'AggrReqNode';
  DECLARE envRef REFERENCE TO Environment.ComIbmAggregateRequestNode;
  CREATE LASTCHILD OF AggrRef.AggrReqNode FROM envRef;
  DELETE FIELD envRef;
END IF;

7. Place a Compute node on the message flow canvas and name it CreateSingleReq. Connect the Out terminal of the Aggregate Control node to In terminal of the Compute node. Click on the Compute node and insert the following code:

CALL CopyEntireMessage();
DECLARE fin REFERENCE TO OutputRoot.*[<1];
IF (FIELDNAME(fin)='XML') THEN
  DELETE FIELD fin;
END IF;
DECLARE Body REFERENCE To OutputRoot;
CREATE Lastchild of OutputRoot AS Body DOMAIN 'XML';
DECLARE ref REFERENCE TO Environment.Orig.XML.Test;
IF (Exists(ref.*[])) THEN
  MOVE ref LASTCHILD TYPE Name NAME 'Sale';
  CREATE LASTCHILD OF OutputRoot.XML FROM ref;
  DELETE FIELD ref;
END IF;
RETURN TRUE;

8. Place an MQ Output node on the message flow canvas and name it AGGR.REQ.OUT. Connect the Out terminal of the CreateSingleReq Compute node to the In terminal of the MQ Output node. On the Properties tab of the node, provide the following details:

Figure 7
9. Place an Aggregate Request node on the message flow canvas. Connect the Out terminal of the AGGR.REQ.OUT MQ Output node to In terminal of the Aggregate Request node. On the Properties tab of node, provide the following details:

**Figure 8**

10. Place a Compute node on the message flow canvas and name it ComIbmAggrReq. Connect the Out terminal of CopyMsgToEnv to the In terminal of the ComIbmAggrReq Compute node. Click on the Compute node and insert the following code:

```sql
SET OutputRoot.Properties.ReplyProtocol = 'MQ';
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Encoding = InputRoot.Properties.Encoding;
SET OutputRoot.MQMD.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
CREATE LASTCHILD OF OutputRoot DOMAIN ('XMLNSC');
-- Read Values from Env
DECLARE EnvRef REFERENCE TO Environment.AggrFolder.XML;
IF LASTMOVE(EnvRef) THEN
    CREATE LASTCHILD OF OutputRoot.XML FROM EnvRef.AggrReq[1].
        AggrControlNode.ComIbmAggregateControlNode;
    DELETE FIELD OutputRoot.XML.ComIbmAggregateControlNode.replies;
ENDIF;
DECLARE cnt INTEGER CARDINALITY (EnvRef.AggrReq[1]);
DECLARE cntlRef REFERENCE TO OutputRoot.XML.ComIbmAggregateControlNode;
SET cntlRef.count = cnt;
CREATE LASTCHILD OF cntlRef NAME 'replies';
DECLARE iterCnt INTEGER 1;
MOVE EnvRef FIRSTCHILD TYPE Name NAME 'AggrReq';
CREATE FIRSTCHILD OF Environment NAME 'ComIbmAggregateRequestNode';
DECLARE comIbmRef REFERENCE TO Environment.ComIbmAggregateRequestNode;
CREATE LASTCHILD OF comIbmRef NAME FIELDVALUE(EnvRef.
    AggrControlNode.ComIbmAggregateControlNode.aggregateName);
DECLARE aggrRef REFERENCE TO comIbmRef.*[<1];
WHILE (iterCnt <= cnt) DO
    DECLARE reqNodeRef REFERENCE TO EnvRef.AggrReqNode.ComIbmAggregateRequestNode.*[<1].*[<1];
    CREATE LASTCHILD OF cntlRef.replies NAME (FIELDNAME(reqNodeRef)||CAST(iterCnt AS CHAR));
    CREATE LASTCHILD OF aggrRef NAME (FIELDNAME(reqNodeRef)||CAST(iterCnt AS CHAR));
    CREATE LASTCHILD OF cntlRef.replies.*[<1] IDENTITY (XML.Attribute)replyIdentifier VALUE reqNodeRef.replyIdentifier;
    CREATE LASTCHILD OF aggrRef.*[<1] IDENTITY (XML.Attribute)replyIdentifier VALUE reqNodeRef.replyIdentifier;
    CREATE LASTCHILD OF cntlRef.replies.*[<1] IDENTITY (XML.Attribute)replyProtocol VALUE reqNodeRef.replyProtocol;
    CREATE LASTCHILD OF aggrRef.*[<1] IDENTITY (XML.Attribute)replyProtocol VALUE reqNodeRef.replyProtocol;
    MOVE EnvRef NEXTSIBLING ;
    SET iterCnt = iterCnt + 1;
END WHILE ;
-- clear up ENVironment
DELETE FIELD Environment.Orig ;
DELETE FIELD Environment.AggrFolder ;
DECLARE blbHoder BLOB;
DECLARE options INTEGER BITOR(FolderBitStream, ValidateContent, ValidateValue);
```

Dynamic message aggregation in WebSphere Message Broker
11. Place an MQ Output node on the message flow canvas and name it AGGR.CTRL.MSG. On the **Properties** tab of the node, provide the following details:

**Figure 9**

12. Place an MQ Input node on the message flow canvas and name it AGGR.CTRL.MSG.IN. On the **Properties** tab of the node, provide the following details:

**Figure 10**

13. Place an MQ Input node on the message flow canvas and name it AGGR.RESP.IN. On the **Properties** tab of the node, provide the following details:

**Figure 11**

14. Place an Aggregate Reply node on the message flow canvas. Connect the Out terminals of the AGGR.CTRL.MSG.IN and AGGR.RESP.IN MQ Input nodes to the Control and In terminals of the Aggregate Reply node. On the **Properties** tab of the node, provide the following details:
15. Place a Compute Node on the message flow canvas. Connect the Out terminal of the Aggregate Reply node to In terminal of Compute node. Click on the Compute node and insert the following code:

```
SET OutputRoot.Properties = InputRoot.Properties;
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XML NAME 'ComIbmAggregateReplyBody';
DECLARE next INTEGER 1;
DECLARE repliesIn REFERENCE TO InputRoot.ComIbmAggregateReplyBody.[next];
DECLARE icnt INTEGER 0;
SET icnt = CARDINALITY(InputRoot.ComIbmAggregateReplyBody.[]);
DECLARE repliesOut REFERENCE TO OutputRoot.XML.ComIbmAggregateReplyBody;
WHILE next <= icnt DO -- 4-way aggregation
    CREATE LASTCHILD OF repliesOut NAME FIELDNAME(repliesIn);
    SET repliesOut.[next].ReplyIdentifier =
        CAST(repliesIn.Properties.ReplyIdentifier AS CHAR);
    MOVE repliesIn NEXTSIBLING;
    SET next = next + 1;
END WHILE;
```

16. Place an MQ Output node on the message flow canvas and name it SAMPLE.MSG.OUT. On the Properties tab, provide the following details:

**Figure 13**

```
MQ Output Node Properties - SAMPLE.MSG.OUT
```

17. After you have completed the above listed steps, here is what the message flow should look like:
How it works at runtime

1. The sample XML has multiple Sale elements -- two of them are used for aggregation and are shown below.
2. CreateSingleReq loops through incoming Sale elements, propagates the highlighted Sale elements for aggregation, and drops them into the AGGR.REQ.OUT queue. Each propagated request is consumed by the Aggregate Request node to keep the details of the aggregation fan-out requests:

Figure 15
3. After all of the messages are propagated to the destination, the flow proceeds from the Control terminal of the Aggregation Control node, and control messages for each request are recorded. As highlighted below, each control message contains a brokerUUID, replyGroupId, and replyIdentifier:

**Figure 17**
4. After all of the messages are propagated to the destination and control messages for each request message are recorded, message flow processing continues from the Out1 terminal of the CopyMsgToEnv Compute node, the combined aggregate message is recorded, and the dropped into the AGGR.CTRL.MSG queue:

**Figure 19**

5. The AGGR.MSG.CTRL.IN node picks the control message from the AGGR.CTRL.MSG queue and triggers the Aggregate Reply node to accumulate the response (Fan-in). When the aggregation works as expected, the control message should always reach the Aggregate Reply node before individual responses (Fan-in) are received.
Figure 20

MQ Input1 receives individual aggregate reply message from the TESTAGGRRES.IN queue and triggers the Aggregate Reply node, which matches the correlation-id for each reply and the replyIdentifier of the control message in order to aggregate all replies. The number of replies to be aggregated is identified by the count. Figure 21 shows that the correlation id of each reply matches the replyIdentifier (MQMD MsgId) for each aggregate request:

Figure 21

Advantages of dynamic message aggregation

- Simple design and easy to configure.
- Easy to implement using built-in WebSphere Message Broker nodes.
- Allows segregation and aggregation for a variable number of messages.

Disadvantages of dynamic message aggregation

- Fan-out and corresponding Fan-in flows are always in a single execution group.
- The message flow processes messages in sequence, and therefore response time is the cumulative sum of the responses of the individual subrequests.
• The message flow thread halts until all subrequest messages are processed and routed to their target applications.

Conclusion

The message flow received one request from the consumer application, split it into a variable number of requests, and propagated them to downstream processes as new requests through a single MQ Output node and Aggregation node. Then the Fan-in flow captured the individual control messages to create a final control message, and the request message flow propagated that final control message to an aggregate Reply node of the Fan-in flow. The Fan-in flow then got the responses of the individual requests, built a composite reply, and propagated it to the original consumer application. In summary, this article showed you how to manipulate the default behaviour of the Aggregation node to pass a new control message from a Fan-in flow to a Fan-out flow in order to achieve dynamic message splitting and aggregation.
Related topics

- **WebSphere Message Broker resources**
  - **WebSphere Message Broker V8 information center**
    A single Web portal to all WebSphere MQ V7 documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere Message Broker V8.
  - **WebSphere Message Broker product page**
    Product descriptions, product news, training information, support information, and more.
  - **Download free trial version of WebSphere Message Broker**
    WebSphere Message Broker is an ESB built for universal connectivity and transformation in heterogeneous IT environments. It distributes information and data generated by business events in real time to people, applications, and devices throughout your extended enterprise and beyond.
  - **WebSphere Message Broker documentation library**
    WebSphere Message Broker specifications and manuals.
  - **Knowledge collection: What's new in WebSphere Message Broker V8**
    A list of resources that explain additions, enhancements, and improvements to WebSphere Message Broker V8 for distributed platforms and z/OS.
  - **WebSphere Message Broker forum**
    Get answers to technical questions and share your expertise with other WebSphere Message Broker users.
  - **WebSphere Message Broker support page**
    A searchable database of support problems and their solutions, plus downloads, fixes, and problem tracking.
  - **IBM Training course: WebSphere Message Broker V8 Development**
    This course from IBM Training shows you how to use the components of the WebSphere Message Broker development and runtime environments to develop and troubleshoot message flows that use ESQL, Java, and PHP to transform messages.
  - **Using control messages in aggregation flows**
    This topic in the WebSphere Message Broker information center shows you how to use control messages in aggregation flows, and explains why the default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored.
  - **Generic message retry and requeue with WebSphere Message Broker**
    This developerWorks article by Stephen Cox shows you how to use WebSphere Message Broker to implement delayed message reprocessing, which is a requirement in many WebSphere Message Broker implementations.

- **WebSphere resources**
  - **developerWorks WebSphere**
    Technical information and resources for developers who use WebSphere products. developerWorks WebSphere provides product downloads, how-to information, support resources, and a free technical library of more than 2000 technical articles, tutorials, best practices, IBM Redbooks, and online product manuals. Whether you're a beginner, an
expert, or somewhere in between, you'll find what you need to build enterprise-scale solutions using the open-standards-based WebSphere software platform.

- **developerWorks WebSphere application integration developer resources**
  How-to articles, downloads, tutorials, education, product info, and other resources to help you build WebSphere application integration and business integration solutions.

- **Most popular WebSphere trial downloads**
  No-charge trial downloads for key WebSphere products.

- **WebSphere forums**
  Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.

- **WebSphere demos**
  Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.

- **WebSphere-related articles on developerWorks**
  Over 3000 edited and categorized articles on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.

- **developerWorks WebSphere weekly newsletter**
  The developerWorks newsletter gives you the latest articles and information only on those topics that interest you. In addition to WebSphere, you can select from Java, Linux, Open source, Rational, SOA, Web services, and other topics. Subscribe now and design your custom mailing.

- **WebSphere-related books from IBM Press**
  Convenient online ordering through Barnes & Noble.

- **WebSphere-related events**
  Conferences, trade shows, Webcasts, and other events around the world of interest to WebSphere developers.

- **developerWorks resources**
  - **Trial downloads for IBM software products**
    No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.
  - **developerWorks business process management developer resources**
    BPM how-to articles, downloads, tutorials, education, product info, and other resources to help you model, assemble, deploy, and manage business processes.
  - **developerWorks blogs**
    Join a conversation with developerWorks users and authors, and IBM editors and developers.
  - **developerWorks tech briefings**
    Free technical sessions by IBM experts to accelerate your learning curve and help you succeed in your most challenging software projects. Sessions range from one-hour virtual briefings to half-day and full-day live sessions in cities worldwide.
  - **developerWorks podcasts**
    Listen to interesting and offbeat interviews and discussions with software innovators.
  - **developerWorks on Twitter**
    Check out recent Twitter messages and URLs.
  - **IBM Education Assistant**
A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)