

Message Correlation

Indeed, activity timed out and appropriate message was generated and sent. We could exercise timeout on the third activity but this would require us to modify bpASNandDNsubmitter so it submits just the ASN and not the DN. We will skip this.

This is one of the possible implementations of the Scatter Gather Pattern. Another implementation, using Asynchronous Subprocess, will be presented in Chapter on Architecting Scalability and Resilience, section dealing with Subprocesses.

11.9.9 Message Relationship Patterns Summary

Message Relationship Patterns discussed in the preceding sections represent the basic relationships that exist between messages in messaging solutions. Other, more complex patterns can be broken down into these basic patterns. All of the Message Relationship Patterns, whose examples were implemented using eInsight, rely on the eInsight Engine's ability to identify key data values in messages and use that key data to associate messages with business processes and, in particular, business process instances that are collecting related messages. To be effectively employed in integration solutions all implementations must have a notion of when to stop waiting for related messages and to processes messages that are already collected. Common indicators are timeout expiration, collection of a pre-defined number of messages or arrival of a sentinel message that indicates end of collection. The basic examples presented above are common to all implementation of a type and can be used as the basis for extended implementations. Custom logic will need to reflect processing requirements of assembled collections of messages, for example selections of a lowest price or a highest bid, rather than how the collection is performed or when collection ends and processing begins.

11.10 eGate Correlation with Dynamic Selectors

eInsight-based Correlation takes advantage of the eInsight Engine's correlation facilities. eInsight Engine ensures that processing components receive only messages they require. Without eInsight, correlation must be implemented differently.

As a general proposition, a correlating component would receive all messages, determine which are of interest and discard or return to sender these that are not. It would then pass the messages of interest to the processing component or would process them itself. The major issue with this approach is that the correlating component receives all messages even if only a small proportion of them are of interest to it. This is an overhead in terms of resources required to receive all messages and increased latency whilst the

Message Correlation

component determines whether it is required to process each message and return it to the sender, or not. The component could rapidly become a bottleneck. Re-submitting messages to the sender could result in the same messages being continually re-delivered to the correlating component only to be continually returned to the sender. This issue could be addressed by having the correlating component subscribe to a JMS Topic so that it receives copies of all messages and can safely discard these that are not of interest. That addresses one issue but introduces another – that of additional resources required for copies of all messages. Ideally one would like the correlating component to receive only messages that are of interest to it.

To further refine the problem one must mention that the correlating component is only interested in messages that are to be correlated by it. That means not just specific kinds of messages, or messages from specific sources but messages with specific Correlation Identifier values.

If one were to develop a solution that correlates messages without the benefit of eInsight Engine's correlation facilities the solution itself would have to implement the necessary correlation logic.

Message correlation processing consists of collecting and storing batches of related messages until batch completion criteria are met, and submission of completed batches to the processing components for processing. With eInsight-based correlation all of the logic necessary to determine if a message is related to any other message(s) is performed by the eInsight Engine. All of the logic necessary to collect, store, determine completion criteria state and process the batch of messages is implemented by eInsight Business Processes.

The fundamental piece of functionality would be that which determines if a current message is related to any other message already known or isolating and making available the pieces of data necessary for some component to make that determination.

Another fundamental piece of functionality necessary for correlation would be the storage of batches of related messages, as they are being assembled, prior to invocation of the completed batch processing component.

Yet another fundamental piece would be the functionality that determines batch completion and delivers related messages for processing.

Message Correlation

The final piece of functionality would be that which processes batches of related messages.

Short of developing a sophisticated and, most likely large, Java Collaboration that implements all of the required functionality, there is no real way to develop a generic Java Collaboration Definition-based Correlation Processor. Not all is lost, however. We will implement some of the Message relationship Patterns, discussed above, using just the JMS Message Server and Java Collaboratoin Definitions.

Bear in mind that it still is much easier to implement correlations using eInsight than it is to do so without it.

11.10.1 Items-Trailer Correlation

Let's re-implement the Items-Trailer Message Relationship Pattern without the use of eInsight and eInsight Correlation functionality.

Recaping the scenario - We are required to collect related messages until a "special" trailer message, indicating that a batch of messages is finished, arrives. When it does we are to process all messages collected so far and release them for further processing by the next component.

Let us assume that each ordinary message carries a Purchase Order Number, an Item Number and a Quantity to be delivered. The trailer message carries the same Purchase Order Number, a dummy ItemNumber and a count of items for cross-check. The messages are to be assembled into a Purchase Order with the Purchase Order Number and a repeating group of Item number and Quantity. Ordinary messages will come from a JMS Destination qItemsIn. The Trailer message will come from another JMS Destination qTrailerIn. This is a contrived example. With the items-trailer one would perhaps be better off using a single queue and a trailer message with a sentinel value in one of the fields, perhaps an ItemNumber value of "TRAILER" or similar.

Let us assume that the ordinary messages use pipe-delimited structure, as in PONumber|ItemNumber|ItemQuantity, and the Purchase Order message is an XML message with the same structure as used previously.

We copy the User-defined OTD, udtitems, and the Purchase order XML Schema, SOAPRequestReply_PO, from the eInsight Items Trailer implementation project.

Message Correlation

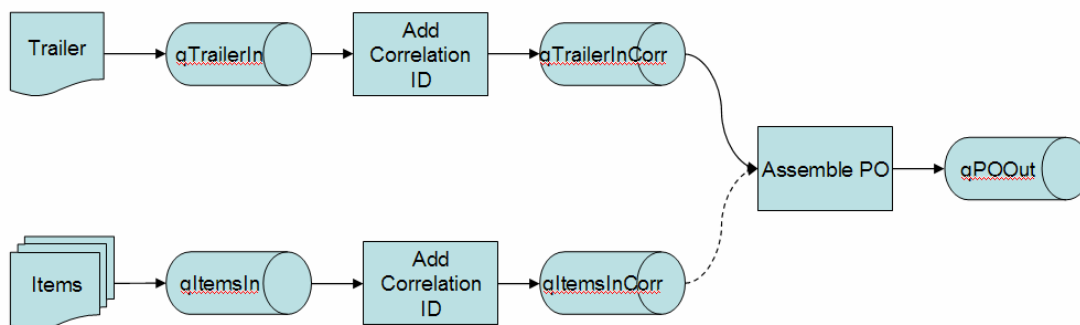
We have the User Defined OTD for the Items messages and the XSD-based OTD for the Purchase Order message that will aggregate related Items. The User-defined Items OTD will also carry a trailer message that will indicate end of run of items and cause completion of correlation processing for the run of items.

With just the eGate, and the JMS Message Server, we need to figure a way of collecting related items and triggering processing of related items once the trailer is received.

One way to implement storage of items is to use a JMS Destination with no current receiver/subscriber – all messages sent there will remain until expired or until explicitly received by some component, whichever is the sooner. One way to implement collection of related items, such that they can be retrieved as a collection of related messages, is to use JMS Correlation ID property to store the Correlation ID of related items, and to use the JMS Selector mechanism to retrieve items related by the common Correlation ID. One way to trigger processing of related items is to create a Java Collaboration that will be triggered by a trailer message containing the Correlation ID to use, and that will retrieve and process related items from the JMS Queue using dynamically constructed selector expression that contains the Correlation ID.

Of the required infrastructure only dynamic selectors are not available out of the box. The reader should have reviewed material presented in “12.5.7 Selectors (MCz)” before looking at the example.

The implementation schematic, below, shows the major user-developed components involved in the solution.



Any items or trailer messages, submitted to qItemsIn and qTrailerIn, will be passed to the “Add Correlation ID” collaboration where the Purchase Order value will be extracted and assigned to the JMS Header Property Correlation ID. The Assemble PO collaboration will be invoked by the arrival of the trailer message. Using the Correlation ID in the trailer message it will construct a

Message Correlation

dynamic selector expression, create a selective receiver, receive from qItemsInCorr, all messages with the matching Correlation ID, combine them and finally send the combined message to qPOOut.

Here the JMS Message server is the means of storing messages whilst they are being correlated and retrieving related messages.

The jcdAddCorrelationID Correlation is identical to that presented in 11.9.5 Header Counted Items Correlation, and is reproduced below.

```
public void receive
    (com.stc.connectors.jms.Message input
    ,com.stc.connectors.jms.JMS W_toJMS )
    throws Throwable
{
    String sPONumber = input.getTextMessage().substring
        ( 0, input.getTextMessage().indexOf( "|" ) );
    com.stc.connectors.jms.Message jmsOut = W_toJMS.createTextMessage();
    jmsOut.getMessageProperties().setCorrelationID( sPONumber );
    jmsOut.setTextMessage( input.getTextMessage() );
    W_toJMS.send( jmsOut );
}
```

As before, the collaboration takes advantage of the delimited message structure to extract the leading part, up to but not including the initial pipe delimiter, the Purchase Order Number, to use as the value of the Correlation ID.

The jcdAssemblePO Correlation, see below, uses the dynamic JMS Selectors technique, documented in 12.5.7 Selectors (MCz), to receive all messages related to the trailer message by the Correlation ID value. It assembles them into a single Purchase Order message and sends the message on its way.

```
package __BookMessageCorrelationEGateCorrelations980396720;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.Queue;
import javax.jms.QueueReceiver;

public class jcdAssemblePO {

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    static final long lTimeout = 5 * 1000;
    static final String cJMS_HOST = "localhost";
    static final String cJMS_PORT = "20007";
    static final String cRECEIVE_FROM = "qItemsInCorr";
    static final String cPROVIDER_URL =
        "stcms://" + cJMS_HOST + ":" + cJMS_PORT;
```

Message Correlation

```
static final String cCONNECTION_FACTORY =
    "connectionfactories/queueconnectionfactory";
static final String cINITIAL_CONTEXT_FACTORY =
    "com.stc.jms.jndispi.InitialContextFactory";
static final boolean cTRANSACTIONED = true;

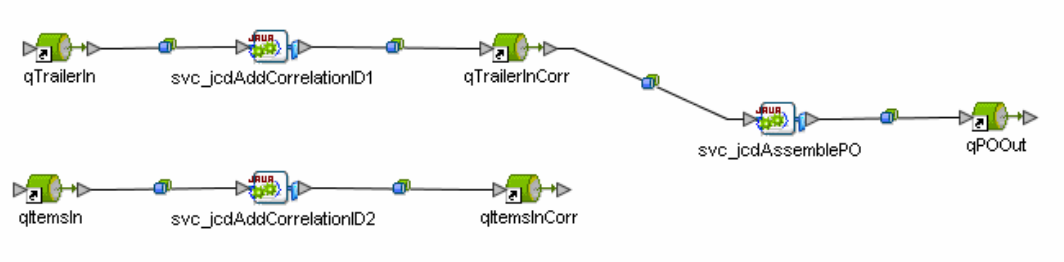
public void receive
    (com.stc.connectors.jms.Message input
    ,udl.udtItems236415793.Udtitems v_udtItems
    ,stcgen.fcxotd.http___MCZ01_14000__SOAPRequestReply.PODocument v_PO
    ,com.stc.connectors.jms.JMS W_toJMS )
    throws Throwable
{
    // construct dynamic selector expression
    ;
    String sCorrelationID =
        input.getMessageProperties().getCorrelationID();
    ;
    String sSelector = "JMSCorrelationID = '" + sCorrelationID + "'";
    ;
    // get a receiver for the specific Queue object
    ;
    QueueConnection myConnection = null;
    QueueSession mySession = null;
    QueueReceiver myReceiver = null;
    ;
    try {
        java.util.Hashtable env = new java.util.Hashtable();
        env.put
            ( Context.INITIAL_CONTEXT_FACTORY, cINITIAL_CONTEXT_FACTORY );
        env.put( Context.PROVIDER_URL, cPROVIDER_URL );
        InitialContext jndiContext = new InitialContext( env );
        QueueConnectionFactory QCFactory =
            (QueueConnectionFactory) jndiContext.lookup
                ( cCONNECTION_FACTORY );
        myConnection = QCFactory.createQueueConnection();
        Queue myQueue = (Queue) jndiContext.lookup
            ( "queues/" + cRECEIVE_FROM );
        mySession = myConnection.createQueueSession
            ( cTRANSACTIONED, Session.AUTO_ACKNOWLEDGE );
        myReceiver = mySession.createReceiver( myQueue, sSelector );
        myConnection.start();
    } catch ( Exception e ) {
        e.printStackTrace();
        throw new Exception
            ( "\n====>>> Exception from jndi processing", e );
    }
    ;
    // ===== end of dynamic selector preliminaries
    ;
    ;
    ;
    // is there at least one message in the correlation queue?
    ;
    javax.jms.TextMessage m = null;
    m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
    if ( m == null ) {
        logger.debug( "\n====>>> There were no messages to receive" );
        mySession.rollback();
        return;
    }
    ;
    v_PO.getPO().setPONumber( sCorrelationID );
    v_PO.getPO().setPODate( "" + new java.util.Date() );
    ;
    // process all related message in the correlation queue
    ;
    int i = -1;
```

Message Correlation

```
while ( m != null) {
    i++;
    ;
    // populate item entry
    ;
    v_udtItems.unmarshalFromString( m.getText() );
    v_PO.getPO().getItems( i ).setItemNumber
        ( v_udtItems.getItemnumber() );
    v_PO.getPO().getItems( i ).setItemQuantity
        ( new java.math.BigInteger
            ( v_udtItems.getItemquantity() ) );
    ;
    // get next related message, if any
    ;
    m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
}
;
// create a message, set its correlation id and payload and send out
;
com.stc.connectors.jms.Message msgOut = W_toJMS.createTextMessage();
msgOut.getMessageProperties().setCorrelationID( sCorrelationID );
msgOut.setTextMessage( v_PO.marshallToString() );
W_toJMS.send( msgOut );
;
// ===== wind down =====
;
mySession.commit();
mySession.close();
myConnection.stop();
myConnection.close();
;
logger.debug( "\n====>>> sent message: " + msgOut.getTextMessage() );
}
}
```

Note the collaboration receives all messages using the selective receiver and constructs the PO message. This is the business part of the correlation infrastructure – the rules that govern what is to be done with the related messages. In this case the PO message is assembled. In other cases item costs could be summed up and a summary message could be sent out. In still other cases each message would be inspected to choose the highest bid, the lowest price or whatever business requirement is being met by message correlation implementation.

Let's create the Connectivity Map, build and deploy.



Feed qItemsIn the following messages in the order given:

PO12345 | IT001 | 2

Message Correlation

```

PO12346 | IT001 | 1
PO12346 | IT002 | 2
PO12345 | IT002 | 3
PO12345 | IT003 | 4
PO12346 | IT005 | 7
PO12345 | IT001 | 8
    
```

Observe 7 messages queued in qItemsInCorr, each with the Correlation ID property set to either PO12345 or PO12346.

Queue Name	Min Sequence Number	Max Sequence Number	Available Count	Number of Receivers	Last Published Date/Time
qItemsInCorr	0	6	7	0	10/17 4:18:29 PM EST

Sequence Number	Message ID	Status	Message Size	Delivery Mode	Priority	Sent On
0	ID:a66b1:10e53b29cd2:175c:819ebc0d:10e54eaa626:30ccdd6210e53b251c67d61	unread	363	PERSISTENT	4	Tue Oct 17 16:17:16 EST 2006
1	ID:db8c3:10e53b29cd3:175c:819ebc0d:10e54eaa626:30ccdd6210e53b251c67d5f	unread	363	PERSISTENT	4	Tue Oct 17 16:17:32 EST 2006
2	ID:dce9e:10e53b29cd4:175c:819ebc0d:10e54eb1ae2:30ccdd6210e53b251c67d5d	unread	363	PERSISTENT	4	Tue Oct 17 16:17:43 EST 2006
3	ID:d3c87:10e53b29cd3:175c:819ebc0d:10e54eb5857:30ccdd6210e53b251c67d5f	unread	363	PERSISTENT	4	Tue Oct 17 16:17:59 EST 2006
4	ID:2586f:10e53b29cd4:175c:819ebc0d:10e54eb1ae2:30ccdd6210e53b251c67d5d	unread	363	PERSISTENT	4	Tue Oct 17 16:17:59 EST 2006
5	ID:43e0e:10e53b29cd4:175c:819ebc0d:10e54eb1ae2:30ccdd6210e53b251c67d5d	unread	363	PERSISTENT	4	Tue Oct 17 16:17:59 EST 2006
6	ID:7c9c4:10e53b29cd4:175c:819ebc0d:10e54eb1ae2:30ccdd6210e53b251c67d5d	unread	363	PERSISTENT	4	Tue Oct 17 16:17:59 EST 2006

Property Name	Value
Type	Text
Destination Name	qItemsInCorr
Message ID	ID:dce9e:10e53b29cd4:175c:819ebc0d:10e54eb1ae2:30ccdd6210e53b251c67d5d
Expiration Time	NEVER
Delivery Mode	PERSISTENT
Message EnqueueTime	Tue Oct 17 16:17:43 EST 2006
Message Size	363
Priority	4
Correlation ID	PO12346

Feed qTrailerIn with the following messages:

```

PO12345 | TRAILER | 0
PO12346 | TRAILER | 0
    
```

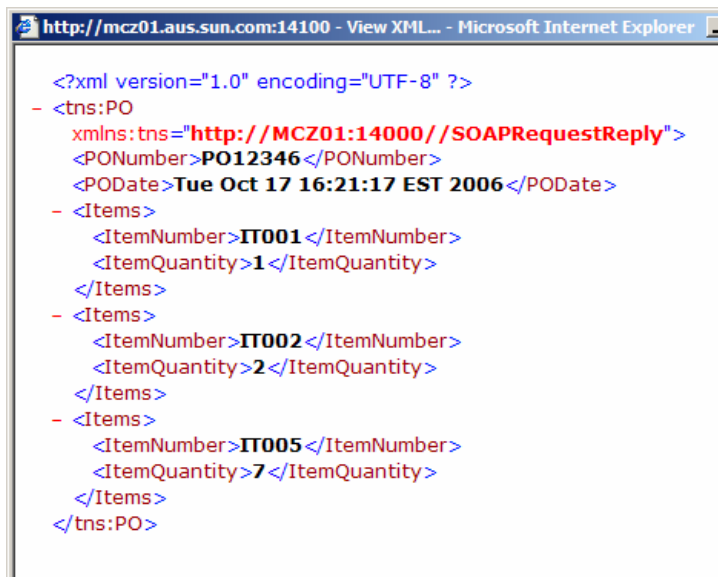
Observe, in qPOOut, the following two assembled Purchase Order messages:

Message Correlation



```
<?xml version="1.0" encoding="UTF-8" ?>
- <tns:PO
  xmlns:tns="http://MCZ01:14000//SOAPRequestReply">
  <PONumber>PO12345</PONumber>
  <PODate>Tue Oct 17 16:21:01 EST 2006</PODate>
  - <Items>
    <ItemNumber>IT001</ItemNumber>
    <ItemQuantity>2</ItemQuantity>
  </Items>
  - <Items>
    <ItemNumber>IT002</ItemNumber>
    <ItemQuantity>3</ItemQuantity>
  </Items>
  - <Items>
    <ItemNumber>IT003</ItemNumber>
    <ItemQuantity>4</ItemQuantity>
  </Items>
  - <Items>
    <ItemNumber>IT001</ItemNumber>
    <ItemQuantity>8</ItemQuantity>
  </Items>
</tns:PO>
```

And



```
<?xml version="1.0" encoding="UTF-8" ?>
- <tns:PO
  xmlns:tns="http://MCZ01:14000//SOAPRequestReply">
  <PONumber>PO12346</PONumber>
  <PODate>Tue Oct 17 16:21:17 EST 2006</PODate>
  - <Items>
    <ItemNumber>IT001</ItemNumber>
    <ItemQuantity>1</ItemQuantity>
  </Items>
  - <Items>
    <ItemNumber>IT002</ItemNumber>
    <ItemQuantity>2</ItemQuantity>
  </Items>
  - <Items>
    <ItemNumber>IT005</ItemNumber>
    <ItemQuantity>7</ItemQuantity>
  </Items>
</tns:PO>
```

Notice that correct items were associated with appropriate orders as expected.

This implementation of the Items Trailer Message Relationship Pattern takes advantage of the JMS Message Server to support storage and selective retrieval of related messages. If configured for discrete timeout, which could be done when messages are queued by the `jcdAddCorrelationID`, message for which there is no trailer would be discarded by the JMS Message Server when expired.

Message Correlation

11.10.2 @@ Timed Items Correlation

[TBD]