Stateless Enterprise Integration Patterns on SAP NetWeaver Process Orchestration

This Paper Describes how to Configure Stateless Enterprise Integration Patterns on SAP NetWeaver Process Orchestration applying its Enterprise Services Bus Capabilities.

Applies to SAP NetWeaver Process Orchestration, release 7.31 SP4 and above.

Author: Alexander Bundschuh
Company: SAP AG
Created on: 28th of February 2013
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>PATTERNS</td>
<td>4</td>
</tr>
<tr>
<td>Content Based Router (Message Routing)</td>
<td>4</td>
</tr>
<tr>
<td>Definition</td>
<td>4</td>
</tr>
<tr>
<td>Solution</td>
<td>4</td>
</tr>
<tr>
<td>Recipient List (Message Routing)</td>
<td>8</td>
</tr>
<tr>
<td>Definition</td>
<td>8</td>
</tr>
<tr>
<td>Solution</td>
<td>8</td>
</tr>
<tr>
<td>Dynamic Router (Message Routing)</td>
<td>14</td>
</tr>
<tr>
<td>Definition</td>
<td>14</td>
</tr>
<tr>
<td>Solution</td>
<td>14</td>
</tr>
<tr>
<td>Splitter (Message Routing)</td>
<td>15</td>
</tr>
<tr>
<td>Definition</td>
<td>15</td>
</tr>
<tr>
<td>Solution</td>
<td>15</td>
</tr>
<tr>
<td>Variant 1: Split depending on line items</td>
<td>15</td>
</tr>
<tr>
<td>Variant 2: Split depending on receivers</td>
<td>22</td>
</tr>
<tr>
<td>Message Filter (Message Routing)</td>
<td>28</td>
</tr>
<tr>
<td>Definition</td>
<td>28</td>
</tr>
<tr>
<td>Solution</td>
<td>28</td>
</tr>
<tr>
<td>Content Enricher (Message Transformation)</td>
<td>30</td>
</tr>
<tr>
<td>Definition</td>
<td>30</td>
</tr>
<tr>
<td>Solution</td>
<td>30</td>
</tr>
<tr>
<td>Content Filter (Message Transformation)</td>
<td>37</td>
</tr>
<tr>
<td>Definition</td>
<td>37</td>
</tr>
<tr>
<td>Solution</td>
<td>37</td>
</tr>
<tr>
<td>Message Translator</td>
<td>38</td>
</tr>
<tr>
<td>Definition</td>
<td>38</td>
</tr>
<tr>
<td>Solution</td>
<td>38</td>
</tr>
<tr>
<td>Value mapping</td>
<td>39</td>
</tr>
<tr>
<td>Adapter</td>
<td>42</td>
</tr>
<tr>
<td>References</td>
<td>45</td>
</tr>
</tbody>
</table>
INTRODUCTION

Enterprise Integration Patterns (EIP) help in solving recurring problems faced in the integration of enterprise applications. For a detailed description of all common patterns, see “Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions” by Gregor Hohpe and Bobby Woold ([1]). This article focuses on stateless Integration patterns in the context of SAP NetWeaver Process Orchestration.

SAP NetWeaver Process Orchestration has been introduced with EhP1 for SAP NetWeaver 7.3. It runs on Java-only, and is a co-installation of the Java-only installation option of SAP NetWeaver Process Integration aka. Advanced Adapter Engine Extended (AEX), Business Process Management (BPM), and Business Rules Management (BRM). For more details, refer to the blog about Installation Options for Process Integration and Orchestration Use Cases on SCN ([3]).

The following stateless patterns are covered, grouped by more generic pattern categories:

- **Message Routing**
  - Content Based Router
  - Recipient List
  - Dynamic Router
  - Splitter
  - Message Filter
- **Message Transformation**
  - Content Enricher
  - Content Filter
- **Message Translator**

For each pattern, I briefly describe how it is defined and how it can be implemented on SAP NetWeaver Process Orchestration. The definitions and corresponding figures have been taken from the book of Gregor Hohpe and Bobby Woold ([1]). The implementation of stateless patterns is usually done as Integration Flows running in the messaging system of SAP NetWeaver Process Orchestration, i.e., on the AEX. Optionally, some of the patterns (i.e., splitter, and content enricher) can be implemented using a process in BPM. A description of the BPM process models can be accessed from the SAP Process Orchestration Integration Patterns blog on SCN ([4]). In this paper however I only focus on the implementation on AEX. As a general guideline, stateless patterns are best implemented as Integration Flows on AEX. This applies especially if no additional business logic is required. If the pattern is part of a larger business process involving multiple integration and/or workflow patterns, it could be implemented within NetWeaver BPM, as the overhead costs become less relevant in the overall scenario.

To implement the stateless patterns on AEX, I mainly work in the Enterprise Services Repository and the Integration Directory. In former, I create the define time artifacts such as data types, message types, mappings, service interfaces, etc. In latter, I configure the message flows. For both, you can decide between Swing clients, or Eclipse-based tools (Enterprise Services Browser in Eclipse, and the PI Designer to implement so called Integration Flows). Latter have been introduced with SAP NetWeaver 7.3 and EhP1 for SAP NetWeaver 7.3 only. In this paper, I mainly stick to the new tools. However, you will also find references to the Swing clients.
PATTERNS

Content Based Router (Message Routing)

Definition
Assume you have an order process where the order can be handled by a specific inventory system only depending on the product type. This addresses the following question:

“How do we handle a situation in which the implementation of a single logical function is spread across multiple physical systems?”

You need to send the orders to various inventory systems whereas data within the message payload, i.e., the product type, determines to which particular target system the message should be passed to. The Content Based Router helps to address this requirement. It scans the relevant message payload, and routes the message to the right channel depending on the data read. Here, it is important that the message is routed to exactly one channel.

Figure 1: Sample – Content Based Router (taken from [2])

Solution
In the Integration Flow you can add a Recipient List and maintain a list of potential receivers. Note that we do not explicitly distinguish between the Recipient List and the Content based Router patterns. We configure the Recipient List in that way that it becomes a Content Based Router. For each receiver branch, you can maintain a condition in form of an xpath expression. The xpath can be either based on the payload data or the message header. During message processing, the condition is evaluated, and if met, the message is routed to the respective receiver. If no receiver could be determined, you have various options to proceed: raise an error, send to default receiver, or ignore. If your conditions are disjoint, each message will be routed to exactly one receiver strictly following the definition of a Content Based Router, see above. However, you have the possibility to define the conditions in such a way that multiple conditions apply during runtime, and hence the message can be routed to multiple receivers following the definition of a Recipient List, see below. See Defining Content-Based Routing on SAP Help Portal.
See Defining a Recipient List in Integration Flows on SAP Help Portal.
See Defining Conditions in Integration Flows on SAP Help Portal.
To demo the scenario, I refer to a simple flight booking scenario where an agency sends flight booking requests to airline booking systems based on the respective airline code. In the SAP Process Integration Designer, you need to define an Integration Flow of pattern Recipient List.

The list of receivers is fixed. Ensure that you select Static Routing Condition as Routing Technique in the Properties section of the Integration Flow, tab Routing Behaviour.
Select the Recipient List to maintain the conditions.

In the Condition Editor, you can maintain the xpath expression selecting from a list of operations and context objects.

Once done, the conditions should look like below.
As mentioned above, you can choose between three different options to react if receiver cannot be found. Here, we choose a default receiver. Note, that the default receiver is also reflected in the message flow diagram.
Recipient List (Message Routing)

**Definition**
Assume you like to send an order to a number of suppliers to request for the best quote. However, the order should not go to all of your suppliers, it should actually depend on the product that you like to order. So, the particular suppliers should be dynamically determined based on the respective goods. The requirement is defined as follows:

“How do we route a message to a dynamic list of recipients?”

For each receiving system, a channel is defined. So, the potential list of receivers is fixed. A Recipient List reads the incoming message, determines based on the message content the list of systems the message should be sent to, and forwards the message to all systems in the list. Other than for the content based router, the message, or to be precise a copy of the message, is sent to multiple receivers.

![Recipient List Diagram](https://via.placeholder.com/150)

**Figure 2: Sample – Recipient List (taken from [2])**

**Solution**
In the Integration Flow you can add a Recipient List and maintain a list of potential receivers. There are two options to implement the Receiver List pattern: you can either use static routing conditions or dynamic conditions. For former, you maintain conditions in form of an xpath expression as seen above for the Content Based Router. Other than for the Content Based Router, here the conditions do not have to be disjoint, and hence the same message may be routed to multiple receivers. For the dynamic conditions option, you run a mapping that returns a list of receivers that the message should be forwarded to. Note, although we call this routing behavior within the Integration Flow Dynamic Message Router this should not to be confused with the Dynamic Router pattern, see also below. For both options, if no receiver could be determined, you have various options to proceed: raise an error, send to default receiver, or ignore.

See [Defining Content-Based Routing](https://help.sap.com/hx) on SAP Help Portal.

In the following example, I stick to the dynamic conditions option. For the other option using an xpath expression as condition, refer to the previous chapter about the Content Based Router. The implementation of the Recipient List pattern is shown based on a scenario for governing master data creation.

**Note:** The scenario here just serves as an example. In general, BPM is not required to implement the Recipient List pattern.

In the overall scenario, customer master data is created and sent for approval. Upon approval, the master data is distributed to various backend systems. The list of receiving systems depends on the country code which is part of the master data maintained. Once the master data is entered in the backend systems, each system sends a confirmation which is then passed to the process, and aggregated. The process below only
shows the integration-centric part of the overall process which takes care of the message correlation. In the process below, only one message is sent from BPM to AEX, the actual distribution of the message to the particular receivers is done in the messaging system of the AEX where a mapping is carried out to determine the list of receivers. However, the same mapping is called from a service task in the BPM process right after the start event to determine the number of receivers in the receiver list. This number is required further down in the process to check within the exclusive gateway if all expected confirmation messages were received.

In the ESR, create a mapping between the incoming master data message type and the specific Receivers message type that is shipped within the software component SAP BASIS. The Receivers message type describes a list of receivers. For each receiver, you need to add a Receiver node. The element Service below each Receiver contains the particular communication component name. In the example below, I have created a user defined function that returns a list of receiver system based on the country code. The SplitByValue standard function adds a context change after each entry. This is necessary to have only one Service node below each Receiver node. Note, this is an example only. Your specific business logic may
look completely different. Use this mapping in an operation mapping with the service interface ReceiverDetermination of software component SAP BASIS as target interface.

In order to be able to call the mapping in the BPM process, switch to the Process Development perspective, and import the operation mapping from ESR.
Select the operation mapping, and click on Finish. This will automatically create a.wsdl describing a synchronous web service encapsulating the mapping.

Assign the service interface to the automated activity.
We are only interested in the number of systems and not the actual system names. So, map the Receiver node to the numberOfSystems context object.

Using the standard function count, the number of occurrences of node Receiver is returned.
As mentioned above, the actual distribution of the message happens in the messaging system. In the SAP Process Integration Designer perspective, create a new Integration Flow of pattern Recipient List (Dynamic Conditions), and maintain a list of potential receivers and their particular communication channels. In the Properties pane, switch to tab Routing Behaviour, and ensure that the Dynamic Message Router Routing Technique is selected.

Select the Recipient List, and maintain the beforehand created operation mapping.
Dynamic Router (Message Routing)

**Definition**

In the Recipient List pattern before, we have seen that the receivers were determined dynamically based on a mapping carried out during message processing. However, the overall list of potential receivers has to be maintained in the Integration Flow, and hence must be known in advance. But what if the receivers are not known in advance? This question is described as follows:

“How can you avoid the dependency of the router on all possible destinations while maintaining its efficiency?”

The Dynamic Router needs to be configured based on information that the receivers provide. It uses a so-called control channel that the receivers send messages to indicating their availability and under what conditions or rules they like to receive messages. During message processing, the Dynamic Router evaluates the rules from the control channel, and routes the message to those receivers whose rules apply.

![Diagram of Dynamic Router](image)

**Solution**

We do not fully support the Dynamic Router pattern unless you like to allow read and write access of your configuration objects from external. You may use the Integration Directory Programming Interface to let the potential receivers dynamically update the routing rule in the Integration Directory. But for security reasons this is not advisable. See [Integration Directory Programming Interface](SAP help Portal) on SAP help Portal.

Alternatively, you can at least implement the control channel part. However, the list of potential receivers would be fixed. The implementation would be a combination of the Recipient List and the Content Enricher patterns. You can apply the same configuration as seen for the Recipient List pattern above, i.e., using a mapping that is carried out during runtime to determine the actual receivers. In addition, in the mapping you can implement a look-up towards the control channel. The control channel might be a DB table that you maintain dynamically or a BRM rule for instance. See [Defining Extended Receiver Determination](SAP Help Portal) on SAP Help Portal.

See [Defining Dynamic Routing in Integration Flows](SAP Help Portal) on SAP Help Portal.

See [Designing and Configuring Mapping Lookups](SAP Help Portal) on SAP Help Portal.
Splitter (Message Routing)

**Definition**
Assume you have an order with multiple line items whereas each needs to be handled by a different supplier. This addresses the following question:

“How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?”

Using a Splitter pattern, we can break up the order into multiple individual messages according to the number of items within the original order. If we know where the items belong to in advance, we can break up the order into as many messages as there are suppliers the items should be sent to. Each message contains only the items that are meant for the particular supplier.

**Solution**
The Splitter can be implemented using PO’s mapping capabilities. As described above, we distinguish between two use cases:

- The order should be broken up into as many individual messages as there are line items. This is done implementing an interface splitting using a one-to-many mapping. Since the mapping is carried out only after the receiver determination, all individual messages are routed to the same receiver.
- The order should be broken up into as many individual messages as there are receivers. Here, we first apply content-based routing to determine all receivers the items should be sent to, followed by content filter removing the items that are not meant to be sent to the respective receiver.


To demo the scenario, I refer to a connecting flight booking scenario where an agency sends flight booking requests to airline booking systems based on the respective airline code. Each single message can contain multiple flight requests. See also article series [5], [6], and [7].

**Variant 1: Split depending on line items**
In this scenario, a consumer likes to book connecting flights, and sends a booking request with multiple flights to an agency. The agency can only handle individual flight requests, so the message is broken up into single flight requests. The common information of the original booking request such as order number needs to be copied to each individual message.
In the ESR, create a data type for the single flight booking request.

Create a data type for the multiple flight booking request. The node BookingOrderList which keeps the flight information occurs unbounded.
Create a one-to-many message mapping between the message type holding the multiple booking request and the message type of the single booking request. Furthermore, create an operation mapping between the respective service interfaces. Note that the occurrence of the single booking request needs to be unbound.

As mentioned above, the BookingOrderList node in the multiple booking request occurs multiple times. Each occurrence keeps the information of a particular individual flight. This node needs to be mapped to the root node of the single booking request so that for each line item an individual single booking request message is created.
The standard CopyValue function ensures that the common data such as agency ID and order number is copied to each single message.

Let's test the mapping. First, we will generate a sample xml file of the mapping source structure. In ESR, select the message mapping, and select Generate → XML File from the context menu.
Maintain the sample xml file. Here, I maintained 3 items in total.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Messages xmlns:ns0="http://sap.com/xi/XI/SplitAndMerge">
  <ns0:Message1>
    <ns0:MultipleBookingOrderRequest xmlnsns0="http://demo.sap.com/patterns/stateless">
      <AgencyID>0001</AgencyID>
      <OrderNumber>123456789</OrderNumber>
      <BookingOrderList>
        <ItemNumber>10</ItemNumber>
        <FlightID><AirlineID>LH</AirlineID><ConnectionID>102</ConnectionID><FlightDate>01.07.2013</FlightDate></FlightID>
        <BookingOrderList>
          <ItemNumber>20</ItemNumber>
          <FlightID><AirlineID>SQ</AirlineID><ConnectionID>325</ConnectionID><FlightDate>02.07.2013</FlightDate></FlightID>
          <BookingOrderList>
            <ItemNumber>30</ItemNumber>
            <FlightID><AirlineID>AA</AirlineID><ConnectionID>999</ConnectionID><FlightDate>09.07.2013</FlightDate></FlightID>
        </BookingOrderList>
      </BookingOrderList>
    </ns0:MultipleBookingOrderRequest>
  </ns0:Message1>
</ns0:Messages>
```
Select Run As → Run Configurations to carry out the mapping.

Create a new Configuration of type ESR Message Mapping, select your message mapping, and the beforehand generated xml sample, and click on button Run.
A new tab opens showing the result of the mapping. You can see that 3 messages have been created, and the common data has been replicated.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns0:Messages xmlns:ns0="http://sap.com/xi/XI/SplitAndMerge">
    <ns1:SingleBookingOrderRequest xmlns:ns1="http://demo.sap.com/patterns/stateless">
        <AgencyData>
            <AgencyID>0001</AgencyID>
            <OrderNumber>123456789</OrderNumber>
            <ItemNumber>10</ItemNumber>
        </AgencyData>
        <FlightID>
            <AirlineID>LH</AirlineID>
            <ConnectionID>102</ConnectionID>
            <FlightDate>01.07.2013</FlightDate>
        </FlightID>
    </ns1:SingleBookingOrderRequest>
    <ns1:SingleBookingOrderRequest xmlns:ns1="http://demo.sap.com/patterns/stateless">
        <AgencyData>
            <AgencyID>0001</AgencyID>
            <OrderNumber>123456789</OrderNumber>
            <ItemNumber>20</ItemNumber>
        </AgencyData>
        <FlightID>
            <AirlineID>SQ</AirlineID>
            <ConnectionID>325</ConnectionID>
            <FlightDate>02.07.2013</FlightDate>
        </FlightID>
    </ns1:SingleBookingOrderRequest>
    <ns1:SingleBookingOrderRequest xmlns:ns1="http://demo.sap.com/patterns/stateless">
        <AgencyData>
            <AgencyID>0001</AgencyID>
            <OrderNumber>123456789</OrderNumber>
            <ItemNumber>30</ItemNumber>
        </AgencyData>
        <FlightID>
            <AirlineID>AQ</AirlineID>
            <ConnectionID>999</ConnectionID>
            <FlightDate>09.07.2013</FlightDate>
        </FlightID>
    </ns1:SingleBookingOrderRequest>
    <ns0:Message1>
    </ns0:Message>
</ns0:Messages>
```
Switch to the SAP Process Integration Designer perspective, and create an Integration Flow between the consumer and the agency. Add a mapping step, and maintain the beforehand created operation mapping.

**Variant 2: Split depending on receivers**
In this scenario, an agency sends a multiple flight booking request to a list of airline booking systems. The multiple booking flight request is split into as many messages as there are receiving airlines. Each individual message contains all items intended for a particular airline. A more detailed description of this variant can also be taken from [7].
In the ESR, create a message mapping with the multiple flight booking request as source and target structure. For re-usability reasons, define a parameter airline of simple type string.)
The mapping is used to implement a content filter pattern, i.e., to remove all items that are not intended for a particular airline. The respective airline ID is passed to the mapping using the parameter airline. So, we create a BookingOrderList node in the target structure only if the source field AirlineID equals the parameter airline. We use the standard function equalsS to compare the value of the AirlineID with the value of the parameter airline. The standard function Constant holds the value of the parameter airline. The result of the function equalsS is a boolean. The standard function createIf expects a boolean, and creates a node only if its input value is true. All other fields are mapped using the default matcher, i.e., same field names are mapped one by one.

Change the context of the AirlineID field to the root node of the structure to ensure that for each line item that fulfills the filter criteria a new node BookingOrderList is created.
As mentioned above, the function Constant holds the value of the parameter airline.

Let’s test the mapping. Here, I have generated a sample xml file with 3 items, 2 flights for Lufthansa (LH), and one flight for Singapore Airline (SQ).
In the configuration I have chosen LH as value for the parameter airline.

The mapping result only contains the LH flights.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns0:MultipleBookingOrderRequest xmlns:ns0="http://demo.sap.com/patterns/stateless">
  <AgencyID>0001</AgencyID>
  <OrderNumber>123456789</OrderNumber>
  <BookingOrderList>
    <ItemNumber>10</ItemNumber>
    <FlightID>
      <AirlineID>LH</AirlineID>
      <ConnectionID>102</ConnectionID>
      <FlightDate>05.08.2013</FlightDate>
    </FlightID>
    <BookingOrderList>
      <ItemNumber>30</ItemNumber>
      <FlightID>
        <AirlineID>LH</AirlineID>
        <ConnectionID>101</ConnectionID>
        <FlightDate>11.08.2013</FlightDate>
      </FlightID>
    </BookingOrderList>
  </BookingOrderList>
</ns0:MultipleBookingOrderRequest>
```
Switch to the SAP Process Integration Designer perspective, and create an Integration Flow of pattern Recipient List between the agency and the airlines. Select the Recipient List, and maintain the xpath conditions. Here, the message should be routed to the communication component AirlineA if AirlineID equals LH, and AirlineB if AirlineID equals SQ.

For each branch, add the beforehand created mapping. For the AirlineA branch, maintain value LH as parameter airline, for the AirlineB branch, maintain value SQ as parameter airline.
Message Filter (Message Routing)

**Definition**
Referring to the order processing example above, let’s assume that you like to send promotion information to your customers. However, some customers are only interested in a subset of your product range, and would like not to get any other information. This matter is defined as follows:

“How can a component avoid receiving uninteresting messages?”

Use the Message Filter pattern to remove any data from a channel that you are not interested in. The Message Filter is a specific type of the Message Router pattern having only one single receiver channel. Any incoming message is evaluated, and if it meets the criteria specified by the Message Filter, the message is routed to the receiver, otherwise it is discarded.

**Solution**
The Message Filter pattern can be implemented applying the Content-Based Router pattern, see above. You forward a message only if the xpath routing condition is met. Otherwise the message should be discarded. A key competence of a message system is actually that it supports guaranteed delivery. So, if a receiver cannot be determined during message processing, it usually should go into an error, and you have to resolve the issue and restart the message. However, here it is actually intended that the message gets lost. In this case, we need to ensure that if no receiver can be determined no error is triggered.
In the SAP Process Integration Designer perspective, define an Integration Flow of pattern Recipient List, and maintain the routing conditions respectively. In the properties pane, switch to the Routing Behaviour tab, and change the Receiver Not Found setting to Ignore this Error.
Content Enricher (Message Transformation)

**Definition**
Assume you like to send an order to a supplier however you cannot provide all information that the receiving system requires to process the order. For instance, the shipment address only contains a ZIP code but state and city are missing. This addresses the following question:

“How do we communicate with another system if the message originator does not have all the required data items available?”

In this specific example, we need to gather the missing address information from a telephone directory for instance. The Content Enricher reads data synchronously from an external system, and appends the additional information to the original message before routing to the actual receiver.

**Solution**
In order to retrieve the additional data from the external system, you can carry out a look-up in the message mapping. For RFC and DB (JDBC) look-ups, standard functions are provided that allow you to implement the look-up by pure configuration. Those functions make use of the schemas stored in the ESR. In the mapping editor, you can simply select the input and output fields of the look-up. It is recommended to use configurable parameters of type channels to make the mappings independent from your actual landscape. For look-ups via Web Service calls (SOAP) however, we do not provide standard functions. Here, you have to code the structure of the input and the output on your own using a user defined mapping function.

See Designing and Configuring Mapping Lookups on SAP Help Portal.
See Designing and Configuring Parameterized Mapping Programs on SAP Help Portal.
See Adding Lookups to Mapping Programs in NWDS on SAP Help Portal.
Let's enhance the simple flight booking scenario from the previous chapter. The flight booking request sent by the agency only contains the connection ID and flight date however the exact flight time and distance are missing. This information needs to be gathered from an external system. To append the missing information to the original booking request, we use a mapping where we call the external system via remote look-up, in our case via RFC. In the ESR in the NWDS, create a message mapping. Define a configurable parameter of category Adapter of type RFC.

**Signature**
Select the source and target messages for this message mapping

**Source Messages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Namespace</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleBookingOrderRequest</td>
<td><a href="http://demo.sap.com/patterns/stateless">http://demo.sap.com/patterns/stateless</a></td>
<td>1</td>
</tr>
</tbody>
</table>

**Target Messages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Namespace</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleBookingOrderRequest</td>
<td><a href="http://demo.sap.com/patterns/stateless">http://demo.sap.com/patterns/stateless</a></td>
<td>1</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>Adapter</td>
<td>RFC</td>
<td>Import</td>
<td></td>
</tr>
</tbody>
</table>
In the expression editor of the message mapping, use a standard conversion function from the navigation pane, here rfclookup. Connect input and output parameters to the lookup function depending on its configuration, see below.
Choose an RFC module. I assume that the RFC schema definition has been already imported to your SWCV in the ESR. Depending on the RFC module that should be called, define input and output parameters. Furthermore, select the channel parameter that you defined beforehand.
Create an operation mapping between your service interface operations. Like for the message mapping, create a parameter of category Adapter and type RFC.

### Operation Definition
Select the source and target operations for this operation mapping

#### Source Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Namespace</th>
<th>Software Component Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleBookingOrderRequest_OB</td>
<td><a href="http://demo.sap/">http://demo.sap/</a> penny AB_DEMO_SWCV 1.0 of demo</td>
<td></td>
</tr>
</tbody>
</table>

#### Target Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Namespace</th>
<th>Software Component Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleBookingOrderRequest_IB</td>
<td><a href="http://demo.sap/">http://demo.sap/</a> penny AB_DEMO_SWCV 1.0 of demo</td>
<td></td>
</tr>
</tbody>
</table>

#### Parameters
Define the parameters used in the binding of the mapping programs. You assign values to the parameters in the interface determination editor in the Integration Directory.

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rfcChannel</td>
<td>Adapter</td>
<td>RFC</td>
<td>Import</td>
<td></td>
</tr>
</tbody>
</table>
In the operation mapping definition, select the beforehand created message mapping, and click on Binding.

In the binding dialog, assign the operation mapping parameter to the message mapping parameter.
Switch to the SAP Process Integration Designer perspective, and create a new communication channel of type RFC pointing to your external system, here FlightInfo.

For each receiver branch, add a mapping step, and maintain the operation mapping and the channel parameter. Assign the beforehand created channel to the channel parameter.
Content Filter (Message Transformation)

**Definition**
Assume you get an order from a partner in a standard format upon you have agreed on. The message contains quite a lot of fields that are required for communicating with your partner however your backend system only needs a small fraction of it. This addresses the following question:

“How do you simplify dealing with a large message when you are interested only in a few data items?”

You use a Content Filter to remove data from a message which you do not need in your application system. You only keep the important information. Very often a Content Filter is also used to simplify the structure of your message, i.e., to flatten a hierarchy into a simple structure which is better readable.

Figure 7: Sample – Content Filter (taken from [2])

**Solution**
The content filter pattern can be implemented using the mapping capabilities within the ESR. For more details about the mapping capabilities, see below.
See Developing and Configuring Mappings on SAP Help Portal.
See Creating a Message Mapping in NWDS on SAP Help Portal.

We actually implemented the Content Filter pattern in the multiple flight booking scenario described above when handling the Splitter pattern. For the variant where we split the multiple flight booking request according to the particular airline, we applied the Content Filter pattern for removing the items that were not intended for the respective airline. For the variant where we split the multiple flight booking request according to the individual line items, we applied the Content Filter pattern to flatten the hierarchy structure. For more details, see above.
Message Translator

**Definition**
You like to exchange messages with various partners each having their own message format and type of communication. This actually addresses the key competence of a messaging system:

"How can systems using different data formats communicate with each other using messaging?"

You use a Message Translator pattern to translate one data format into another. We distinguish different levels of transformations:

- **Data structures:** this describes the different entities such as customer, address, order, etc., and how they are related to each other.
- **Data types:** describing the data on field level, e.g., field name, type such as integer or string, constraints such as field length, range, etc.
- **Data representation:** dealing with data formats such as xml or flat file format, what character set is used, or whether the data is encrypted or compressed, etc.
- **Transport:** dealing with how data is transferred between two systems, especially how guaranteed delivery is ensured, and how the different communication protocols such as http, SOAP, JMS, etc., are translated.

**Figure 8: Sample – Message Translator (taken from [2])**

**Solution**
PO does support all levels of transformations. On the level of data types and data structure, the transformation is usually carried out in a mapping. You can choose between the following mapping tools: message mapping in ESR, Java mapping, XSLT. All mapping tools support structural mappings as well as data transformations. The message mapping in ESR comes with a graphical mapping tool (for ESR in Eclipse, only from release 7.31 SP6 on) that allows you to create mappings between message types defined in the ESR. Any cardinality is supported, i.e., one-to-one for mappings from one message type to another, one-to-many for message split (see above), many-to-one for merging messages (stateful, hence need to be called within a BPM process). Furthermore, on the level of data transformation, PO supports value mapping. Within an operation mapping between the operations of source and a target service interfaces, you need to select the respective mapping program. In the Integration Flow, you can add a mapping task. In the properties of the mapping task, maintain the operation mapping.

See [Developing and Configuring Mappings](https://help.sap.com) on SAP Help Portal.
See [Designing and Configuring Value Mappings](https://help.sap.com) on SAP Help Portal.

On the level of data representation and transport, the transformation is usually carried out in an adapter. PO provides a number of different adapter types such as http, SOAP, RFC, IDoc, JDBC, JMS, file/FTP, Mail, RNIF, CIDX, etc. Beside the adapters that come with a standard PO installation, we provide two add-ons accessible from the SAP Service Marketplace in the area of Business-to-Business communication, namely the Secure Connectivity add-on comprising an SFTP adapter and a PGP module, and the B2B add-on comprising an OFTP, an AS2, and an X.400 adapter. Furthermore, SAP partners with various 3rd party adapter vendors providing a huge range of additional adapters. Depending on the protocol, different security mechanisms are supported to ensure confidentiality of the data exchange. We distinguish between security mechanisms on transport layer, such as http over SSL, and on message layer such as digital signature and
message-level encryption. Former is used when confidentiality is required on the communication lines, latter if message content needs to be confidential also in intermediate message stores.

You configure an adapter by maintaining a communication channel, and assigning the same to the end points of an Integration Flow.

See Configuring Adapters on SAP Help Portal.
See Defining Channels in Integration Flows on SAP Help Portal.
See Message-Level Security on SAP Help Portal.
See Business-to-Business add-on on SAP Help Portal.
See Secure Connectivity add-on on SAP Help Portal.

We have seen a couple of examples of PO's mapping capabilities already in the chapters above. In the following, I just like to emphasize on the value mapping capability within a mapping which hasn't been covered so far in this paper. Furthermore, although briefly mentioned in the Integration Flow examples above, let's show how communication towards systems is configured.

**Value mapping**

In the master data governance example above, let's enhance the master data before it is sent to the backend systems. The customer master data contains a field country that holds a country code, e.g., DE for Germany. This value should be transformed into a description whereas the language depends on the system the master data is sent to, e.g., Germany for System 2.

I have maintained the value mappings in a CSV file. In the first column, the value mapping groups are displayed. A value mapping group holds the different representations of an object, here of object Country. The subsequent columns contain the value mappings for each agent and scheme tuple whereas agent and scheme are separated by a pipe (|). An agency stands for a source or target system of the value mapping. The scheme identifies the type of the value.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOL_PM269_Sys3</td>
<td>CountryText</td>
<td>SOL_PM269_ICP</td>
<td>CountryID</td>
</tr>
<tr>
<td>2</td>
<td>SOL_PM269_DE</td>
<td>DE</td>
<td>Germany</td>
<td>Deutschland</td>
</tr>
<tr>
<td>3</td>
<td>SOL_PM269_US</td>
<td>US</td>
<td>United States of America</td>
<td>Vereinigte Staaten von Amerika</td>
</tr>
<tr>
<td>4</td>
<td>SOL_PM269_IN</td>
<td>IN</td>
<td>India</td>
<td>Indien</td>
</tr>
</tbody>
</table>

In the SAP Process Integration Designer Perspective, you can import the value mappings from the CSV file by selecting Process Integration → Value Mapping → Import Value Mapping Groups from the main menu.
Select the beforehand created file, and click on Next.

On the next screen, a summary of all groups created is displayed. Click on Finish.
Switch to tab Value Mapping in the navigation pane to display the beforehand uploaded value mappings. You can see that the Agency column holds the Communication Component names, i.e., the names of the sending and receiving systems. The schema for the sending system is CountryID, the schema for all receiving systems is CountryText. For the different receiving systems, the description is entered in a different language.

In the ESR, change the message mapping. Add the standard function Value mapping, and link source and target fields.
Maintain the parameters of the Value mapping function as follows. The Source Agency is assigned to the header parameter SenderService, the Target Agency to ReceiverService. During runtime, those parameters are substituted by the actual sender and receiver system names taken from the message header. The schemas are maintained according to the definition of the value groups, see above. You can choose between different behaviors if the lookup fails, i.e., if the actual value is not stored in the value mapping table. You can either use the original key, use a default value, or throw an exception.

**Adapter**

The customer master data should be sent to the receiving system via file adapter. To maintain the channel settings, you have two options, either using inline channels or referenced channels. Former are defined as part of the Integration Flow, and get automatically generated when the Integration Flow is deployed on the runtime. Latter are distinct objects and have their own lifecycle, i.e., they can be created, edited, deleted, and activated. On activation they get deployed to the runtime. Referenced channels are associated with a system during their creation. So, they can only be assigned to the Integration Flow when you use the associated system. Inline channels have been supported with the introduction of Integration Flows. The referenced channels are supported starting with SAP NetWeaver 7.31 SP6. You use inline channels if you like to control a particular Integration Flow with the activation and deactivation of a channel. You use referenced channels if you like to re-use the same channel settings across different Integration Flows.
Let’s create a referenced channel. In the navigation pane of the SAP Process Integration Designer Perspective pick the business system, and select New Channel from the context menu.

Maintain the channel, i.e., select a channel name, direction, adapter type, transport protocol, message protocol, and the adapter-specific settings, etc. Here, we chose file as adapter type.

In the Integration Flow, click on the connection between the main pool and the receiver, and select Assign Referenced Channel from the context menu.
Select the beforehand created channel. Note, you will only be able to select from a list of channels associated with the respective system.

The channel has been assigned. The icon ( ) next to the channel name indicates that it is a referenced channel.
References

[4] Blog on SCN about SAP Process Orchestration Integration Patterns by Abdul-Gafoor Mohamed