SOA governance using WebSphere DataPower and WebSphere Service Registry and Repository, Part 1: Leveraging WS-MediationPolicy capabilities

Mario E. De Armas (drhoek@us.ibm.com)
Software Architect
IBM

Thomas Burke (burket@us.ibm.com)
Advisory Software Engineer
IBM

Evan Jardine-Skinner (eskinner@uk.ibm.com)
Software Engineer
IBM

Dave Spriet (spriet@ca.ibm.com)
Senior Software Engineer
IBM

Starting with the release of IBM® WebSphere® DataPower® firmware V5.0 and WebSphere Service Registry and Repository V8.0, IT service developers and policy administrators can now attach and affect ESB behavior related to Service Level Management, routing, and version mediation capabilities using service policies (similar to how WS-SecurityPolicy performs today). This article explains the new DataPower mediation policy (WS-MediationPolicy V1.6) capabilities and syntax. The article also demonstrates how to create, attach, and deploy them using WSRR and IBM Integration Developer environments.

View more content in this series

Overview
This first article of a three-part series describes WS-MediationPolicy capabilities, basic examples, tips on creating your own policy, and how to attach it to a Web service proxied by WebSphere DataPower SOA appliance (hereafter called the "appliance").

In SOA deployments, the desired goal is to represent policy intent in a form that transcends the specific enforcement platform's configuration details. Thus, many products today (including DataPower and WebSphere Service Registry and Repository) support many types of standardized
Some policy domains have gone through standardization process (such as W3C, IETF, OASIS, and so on), while others are vendor or product specific. The extensibility to create domain-specific policy capabilities, requirements, and general characteristics that can be applied to Web services is the objective of the **WS-Policy 1.5 framework specification**. This article focuses on a new policy domain created by IBM for use with DataPower appliances that allows customers to express and affect Web service proxies deployed in DataPower.

This new policy domain, named WS-MediationPolicy, can express conditions based on traffic management requirements (for example, QoS or SLM policies), routing directives and schema validation, and invoking data translation stylesheet maps. These capabilities are represented by a language made up of policy assertions and policy assertion parameters. The mediation capabilities covered by this new policy domain are intended to provide policy administrators and service deployers with a simple way to attach commonly expressed IT operational requirements onto DataPower from a WebSphere Service Registry and Repository (WSRR) platform - without having to learn DataPower platform configuration details!

You can use WS-MediationPolicy to control interactions between Web services and consumers to meet specific business needs, such as limiting the rate of traffic to a service, validating and translating message data formats, or routing to an alternate endpoint. **Figure 1** shows a simple topology with the DataPower appliance acting as the Policy Enforcement Point (PEP) and enforcing WS-MediationPolicy.

**Figure 1. DataPower enforcing WS-MediationPolicy**

Using WS-MediationPolicy makes it possible to implement the "if-then-else" processing directives to control traffic and translate messages using policy assertions, instead of directly modifying the appliance configuration. Also, defining the operational requirements using policy documents enables reuse within the IT enterprise. Furthermore, WS-MediationPolicy's value is greatly amplified by using it within WebSphere Service Registry and Repository (WSRR) to define Service Level Agreements (SLA).
Service Gateway use case example

First, let's review a typical SOA governance use case that can benefit from using a mediation policy.

Assume there is a service called "GlobalWeather", which provides world-wide weather information based on country and postal code. The service provider wants to protect the deployed service by limiting the amount of traffic that is allowed to access the endpoint.

The service provider likely does not have access to the DataPower appliance (or perhaps the necessary skill) to implement the service gateway configuration necessary to enforce the desired business policy limiting the message traffic. However, since its company has also adopted SOA governance using WSRR product, the service owner decides to leverage WSRR and author and attach a policy to its service using the WSRR policy management user interface features.

The service provider creates a WS-MediationPolicy within WSRR that describes the desired service limits and attaches it to its GlobalWeather service. At that point, the DataPower appliance's WS-Proxy Service detects the new policy attached and re-synchronizes the proxy to enforce new behavior.

The service provider did not have to access or learn DataPower configuration details to implement and apply common traffic management capabilities to their service.

Along with being able to protect the service endpoint, we have also now enabled the service provider policy to be governed properly by policy administration product, such as WSRR. That is an important requirement to embrace SOA governance process and best practices.

In addition, and as described in the next section, policy authors can create policies to perform more common types of mediation actions and patterns for most service providers.

Usage patterns for WS-MediationPolicy

Before discussing the details of WS-MediationPolicy syntax, it is important to understand some common usage patterns applicable to this new policy vocabulary:

- Reject based on Condition *
- Queue based on Condition *
- Route based on Condition *
- Message validate
- Message translate
- Notify (log) based on Condition *

* Condition is optional in the mediation policy syntax.

Along with each pattern description, we identify policy templates that are available on the DataPower appliance for reuse to implement each pattern. These patterns are found in the store://policies/templates directory. Their details are shown in Figure 2.
The **Reject based on Condition** pattern rejects messages that exceed a specified SLM criteria (for example, reject messages that exceed a 10 messages per minute threshold). An example of this pattern is found in the `wsp-mp-1-6-qos-reject-messages.xml` template. This policy document defines a condition of "greater than 10 messages per minute" and invokes a "reject" action if the condition evaluates to true. If 11 or more messages arrive within the 1 minute interval, they are rejected.

The **Queue based on Condition** is similar to the "Reject based on Condition" pattern, except that it limits the rate of messages without explicitly rejecting them. An example of this pattern is found in the `wsp-mp-1-6-qos-queue-messages.xml` template. This policy is similar to `wsp-mp-1-6-qos-reject-messages.xml`, except that it invokes a "queue" action if the condition evaluates to true. Messages that exceed this threshold are held in a queue until the interval elapses, then they are sent for processing.

The **Route based on Condition** pattern specifies an alternate backend for message processing if a specific condition is met. This pattern is useful when there are circumstances (such as a schedule), which dictate that a specified endpoint is used, instead of the default backend specified for the service. An example of this pattern is found in the `wsp-mp-1-6-route-after-date.xml` and `wsp-mp-1-6-route-during-maintenance.xml` templates.

The conditions used in the previous patterns have been based on the Service Level Management (SLM) parameters, but these examples demonstrate the flexibility of WS-MediationPolicy by using a schedule-based condition. The "after-date" template demonstrates how a policy document is created to take effect sometime in the future. The "during-maintenance" template demonstrates how a policy is created to be enforced at periodic intervals.

The **Message Validate** pattern validates a message to ensure it meets a specific schema (perhaps a schema that is more strict than the one specified in the WSDL). This pattern is useful if you want...
to ensure that a message meets requirements that are not dictated by the WSDL used to create the service. An example of this pattern is found in the `wsp-mp-1-6-message-validate.xml` template. This template has no condition at all and performs the validate action for every message at all times.

The **Message Translate** pattern translates messages from one version to another version. This gives a service the ability to handle multiple incoming message versions, yet provide message translation capability such that the backend only needs to handle one version. An example of this pattern is found in the `wsp-mp-1-6-qos-message-version-translation.xml` template. This pattern also has no condition, but it has three actions (validate-transform-validate). In this template, the first action (validate) ensures that the incoming message adheres to the schema (for example, `PurchaseOrder-V10.xsd`) expected by the transform. The transform action converts the message to a new format (for example, `PurchaseOrder-V20.xsd`), and the final action validates the outcome of the transform.

The **Notify based on Condition** pattern writes a message to the appliance's message log when a message arrives while certain conditions are met. There is no template available for the "Notify based on Condition", but you can easily create one based on any of the conditional patterns described above.

The templates shown in Figure 2 are used as the basis of policy documents that you write to perform message validation, message translation, message translation, and Quality of Service functions. A brief explanation of each template on the appliance is described in Table 1.
Table 1. WS-MediationPolicy template descriptions

<table>
<thead>
<tr>
<th>Template name</th>
<th>Description</th>
<th>Ready to use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsp-mp-1-6-message-validate</td>
<td>Performs a validate action using the schema specified.</td>
<td>No, the schema specified in the template does not exist. Replace it with the schema that you want to use.</td>
</tr>
<tr>
<td>wsp-mp-1-6-message-version-translation</td>
<td>Validates a message to see if it adheres to a specific schema (for example, obsolete), translates it, then validates it again to ensure it adheres to the desired schema.</td>
<td>No, the schemas and XSLT specified in the template do not exist. Replace them with the artifacts that you want to use.</td>
</tr>
<tr>
<td>wsp-mp-1-6-qos-errorcount-route</td>
<td>If the message error count exceeds a 5 errors per minute, then route messages to a different endpoint.</td>
<td>No, change the endpoint to the one that you want to use. You may also want to adjust the QoS parameters to suit your needs.</td>
</tr>
<tr>
<td>wsp-mp-1-6-qos-queue-messages</td>
<td>If messages exceed a rate of 10 messages per minute, then queue messages for this interval and send them during the next interval.</td>
<td>Yes, but you may want to adjust the QoS parameters to suit your needs.</td>
</tr>
<tr>
<td>wsp-mp-1-6-qos-reject-messages</td>
<td>If messages exceed a rate of 10 messages per minute, then reject them.</td>
<td>Yes, but you may want to adjust the QoS parameters to suit your needs.</td>
</tr>
<tr>
<td>wsp-mp-1-6-qos-token-bucket-queue</td>
<td>Performs the QoS task using the TokenBucket SLM algorithm, which allows bursts of messages to be serviced while enforcing a lower “steady state” rate of messages.</td>
<td>Yes, but you may want to adjust the QoS parameters to suit your needs. Read the comments in the template to get a better understanding of the TokenBucket algorithm.</td>
</tr>
<tr>
<td>wsp-mp-1-6-route-after-date</td>
<td>Routes all messages that arrive on or after a specific date to an alternate endpoint.</td>
<td>No, adjust the date value and change the endpoint to the one that you want to use.</td>
</tr>
<tr>
<td>wsp-mp-1-6-route-during-maintenance</td>
<td>Routes all messages that arrive within a specific time window (for example, server maintenance window) to an alternate endpoint.</td>
<td>No, adjust the time window values and change the endpoint to the one that you want to use.</td>
</tr>
<tr>
<td>wsp-mp-1-6-route</td>
<td>Routes all messages to an alternate endpoint.</td>
<td>No, change the endpoint to the one that you want to use.</td>
</tr>
</tbody>
</table>

Notice that the templates are named in a way that easily identifies their intent (for example, QoS, route, and message validation or translation operations). Take a look at the templates to get familiar with some of the things you can do with WS-MediationPolicy. Note that the templates can only cover a finite range of possibilities. However, after reading this article, you will be equipped to write more sophisticated policies that meet your specific business operational needs.

**WS-MediationPolicy syntax**

To take the templates described in the previous section to the next level and to meet your business needs, you need to become familiar with the syntax for WS-MediationPolicy. As shown in Figure 1, you can apply WS-MediationPolicy assertions as conditional processing directives. Simpler derivatives are also possible that can omit the conditional parameter and always enforce a specified action set. WS-MediationPolicy assertions typically follow the structure shown in Listing 1.
Listing 1. Basic WS-MediationPolicy structure (pseudocode)

```xml
<!-- WS-MediationPolicy assertion begins -->
<Rule>
  <!-- Declare <Expression> AND/OR <Schedule> here. -->
  <Condition>
    <!-- Declare actions to execute if <Condition> evaluates to true here. -->
    <Action IfCondition="true">
      <!-- Declare actions to execute if <Condition> evaluates to false here. -->
      <Action IfCondition="false">
        <!-- Declare actions to execute if <Condition> evaluates to false here. -->
      </Action>
    </Action>
  </Condition>
</Rule>
<!-- WS-MediationPolicy assertion ends -->
```

WS-MediationPolicy assertions are always contained in a `<Rule>` element. This `<Rule>` may contain a `<Condition>` element that describes a condition, which evaluates to true or false and one or more `<Action IfCondition="xs:boolean">` elements, each containing the actions to be executed when the value of the IfCondition attribute matches the outcome of evaluating the `<Condition>` element. WS-MediationPolicy assertions usually fit into one of these two basic patterns.

- If `<Condition>` evaluates to true, then execute all actions in `<Action IfCondition="true">`. Otherwise, execute all actions in `<Action IfCondition="false">`.
- There is no `<Condition>` and one `<Action>`. Executes all of the actions in `<Action>`.

Note: An `<Action IfCondition="true">` element is syntactically synonymous with `<Action>` (implied "true" attribute). Also, `<Action>` blocks are optional, so the first pattern does not require the @IfCondition="false" action block.

WS-MediationPolicy conditional expressions must comply with the WS-MediationPolicy schema provided in the Downloads section of this article.

The best way to describe WS-MediationPolicy is by showing an example and explaining the behavior described by the sample policy, and then discussing variations of the policy that can be achieved by changing the policy. The first example is shown in Listing 2. This policy routes messages to an alternate server (http://secondary-server:9080) than specified in the Web service (such as WSDL or WS-Proxy service configuration) when the rate of messages exceeds 100 messages per minute (PT60S), but only on Mondays, Wednesdays, and Fridays, between midnight and 6:00 AM.

Listing 2. Sample WS-MediationPolicy

```xml
<wsp:Policy wsp:Name="Route-secondary-schedule-template"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Expression>
        <wsmp:Attribute>MessageCount</wsmp:Attribute>
        <wsmp:Operator>GreaterThan</wsmp:Operator>
        <wsmp:Value>100</wsmp:Value>
      </wsmp:Expression>
    </wsmp:Condition>
    <wsmp:Action>
      <wsmp:TimeWindow>
        <wsmp:Start>Midnight</wsmp:Start>
        <wsmp:End>6:00 AM</wsmp:End>
      </wsmp:TimeWindow>
    </wsmp:Action>
  </wsmp:Rule>
</wsp:Policy>
```
Rule
In the example shown above, the <Rule> element is the WS-MediationPolicy policy assertion within the <wsp:Policy> expression. A Rule policy assertion may contain one <wsmp:Condition> element and always contains at least one <wsmp:Action> element. If a <wsmp:Condition> element is not specified, then the evaluation of the assertion assumes an implied true condition for deciding which actions to enforce.

Condition
The "<wsmp:Condition>" element represents a condition that can be evaluated to a true or a false value. The <wsmp:Condition> may contain an <wsmp:Expression> element, a <wsmp:Schedule> element, or as shown in Listing 2, it may contain both. The <wsmp:Condition> of a WS-MediationPolicy assertion evaluates to true only if the <wsmp:Expression> and <wsmp:Schedule> elements both evaluate to true.

The "<wsmp:Expression>" element represents a criteria or threshold that will be evaluated at runtime to a true or false value for each message processed by a policy enforcement platform. The evaluation will be based on the policy limits expressed in the policy criteria selected in <wsmp:Expression>, and the rate/latencies of the messages being processed (for example, message throughput, latency, or errors per time interval).

The number of elements required for an <Expression> varies, depending on the value selected for <Operator>. Valid values for <Operator> are listed in Table 2, along with their meanings.

Table 2. Valid values for Operator and their meanings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GreaterThan</td>
<td>A simple numeric algorithm that evaluates to “true” when the attribute is greater than the defined value.</td>
</tr>
<tr>
<td>LessThan</td>
<td>A simple numeric algorithm that evaluates to “true” when the attribute is less than the defined value.</td>
</tr>
<tr>
<td>HighLow</td>
<td>An algorithm that evaluates to “true” when attribute reaches the high threshold specified as the value. It then continues to evaluate to “true” until the attribute reaches the low threshold specified as the limit.</td>
</tr>
</tbody>
</table>
TokenBucket

A rate-based algorithm that allows for bursts of messages. The algorithm consists of a bucket with a maximum capacity of limit tokens. The bucket refills at a constant rate of value tokens per interval, while for each unit of the attribute, a token is removed. This algorithm evaluates to "true" when there are no tokens in the bucket. Otherwise, it evaluates to "false".

If "GreaterThan" or "LessThan" are chosen for <Operator>, then the remaining elements required are:

- **<Attribute>**: This represents the aspect of the message that applies to the <Expression> (see Table 2).
- **<Interval>**: This represents the duration of the monitoring interval.
- **<Value>**: This represents the observed quantity that will cause the value of <Expression> to change for the for remainder of the specified <Interval>. An example for GreaterThan is shown in Listing 2. An example for LessThan is shown in Listing 3.

**Listing 3. LessThan example**

```xml
<wsp:Policy wsp:Name="LessThan example"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Expression>
        <wsmp:Attribute>MessageCount</wsmp:Attribute>
        <wsmp:Operator>LessThan</wsmp:Operator>
        <wsmp:Value>100</wsmp:Value>
        <wsmp:Interval>PT60S</wsmp:Interval>
      </wsmp:Expression>
    </wsmp:Condition>
    <wsmp:Action IfCondition="true">
      <wsmp:RouteMessage>
        <wsmp:EndPoint>http://secondary-server:9080</wsmp:EndPoint>
      </wsmp:RouteMessage>
    </wsmp:Action>
  </wsmp:Rule>
</wsp:Policy>
```

The "HighLow" for <Operator> works like GreaterThan, except it requires that an additional element, <Limit>, which represents the message count that causes the <Expression> to evaluate back to false (for example, acts as a "low watermark"). An example for GreaterThan is shown in Listing 4.

**Listing 4. HighLow example**

```xml
<wsp:Policy wsp:Name="HighLow example"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Expression>
        <wsmp:Attribute>MessageCount</wsmp:Attribute>
        <wsmp:Operator>HighLow</wsmp:Operator>
        <wsmp:Value>100</wsmp:Value>
        <wsmp:Limit>50</wsmp:Limit>
        <wsmp:Interval>PT60S</wsmp:Interval>
      </wsmp:Expression>
    </wsmp:Condition>
  </wsmp:Rule>
</wsp:Policy>
```
The "TokenBucket <Operator>" also requires a <Limit> element to be specified. The TokenBucket operator allows bursts of traffic on a service, while maintaining a lower average rate of traffic. This is accomplished by adding "tokens" to a "bucket" at the rate of <Value> tokens per <Interval>. However, the number of tokens in the bucket can never exceed <Limit>. Each message removes a token from the bucket. If the bucket becomes empty, then the Expression will evaluate to true. Otherwise, it is false. MessageCount and ErrorCount are the only <Attribute> values allowed for TokenBucket. An example for TokenBucket is shown in Listing 5.

Listing 5. TokenBucket example

```xml
<wsp:Policy wsp:Name="TokenBucket example"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Expression>
        <wsmp:Attribute>MessageCount</wsmp:Attribute>
        <wsmp:Operator>TokenBucket</wsmp:Operator>
        <wsmp:Value>10</wsmp:Value>
        <wsmp:Limit>100</wsmp:Limit>
        <wsmp:Interval>PT60S</wsmp:Interval>
      </wsmp:Expression>
    </wsmp:Condition>
    <wsmp:Action IfCondition="true">
      <wsmp:RouteMessage>
        <wsmp:EndPoint>http://secondary-server:9080</wsmp:EndPoint>
      </wsmp:RouteMessage>
    </wsmp:Action>
  </wsmp:Rule>
</wsp:Policy>
```

Table 3. Valid values for attribute and their meanings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorCount</td>
<td>The number of faults observed during this monitoring interval.</td>
</tr>
<tr>
<td>MessageCount</td>
<td>The number of messages handled during the monitoring interval.</td>
</tr>
<tr>
<td>InternalLatency</td>
<td>The internal latency (processing time) in milliseconds.</td>
</tr>
<tr>
<td>BackendLatency</td>
<td>The appliance-to-server latency in milliseconds.</td>
</tr>
<tr>
<td>TotalLatency</td>
<td>The sum of the backend and internal latency in milliseconds.</td>
</tr>
</tbody>
</table>

You can use the "<wsmp:Schedule>" element to enable or disable a collection of actions based on the time that a message is processed. For example, a policy assertion can be written to reject messages sent between certain hours of the day (see Listing 6), on certain days of the week (see Listing 7), or between certain dates (see Listing 8).
A daily schedule is configured in a policy expression by using the "<wsmp:Daily>" element. StartTime and StopTime attributes are required for <wsmp:Daily>, and the times must be specified in an xs:time format, without a time zone offset or fractional seconds. The StopTime may be the same value as StartTime, which describes a 24-hour duration, starting at StartTime. If StopTime is less than StartTime (as in Listing 6), then the duration will be calculated across midnight. In this case, the duration is 8 hours long, spanning from 22:00 hours (10 PM) to 06:00 hours (6 AM). Note that StartTime and StopTime are processed to an hour and minute granularity, any seconds specified in time values are ignored. Also, note that time values are relative to the policy enforcement platform's local time, timezone, and DST settings.

**Listing 6. Reject messages based on a daily schedule**

```xml
<wsp:Policy wsp:Name="Reject messages based on a daily schedule"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Schedule>
        <wsmp:Daily StartTime="22:00:00" StopTime="06:00:00"/>
      </wsmp:Schedule>
    </wsmp:Condition>
    <wsmp:Action IfCondition="true">
      <wsmp:RejectMessage/>
    </wsmp:Action>
  </wsmp:Rule>
</wsp:Policy>
```

The "<wsmp:WeekDays>" element describes the days of the week that the schedule is effective. The days of the week are designated by the Days attribute as the weekday names are separated by a "+" character. All weekday names must start with an uppercase character and followed by all lowercase characters. The schedule shown in Listing 7 rejects all messages received on Saturdays and Sundays.

**Listing 7. Reject messages based on a day of the week**

```xml
<wsp:Policy wsp:Name="Reject messages based on a day of the week"
  <wsmp:Rule>
    <wsmp:Condition>
      <wsmp:Schedule>
        <wsmp:WeekDays Days="Saturday+Sunday"/>
      </wsmp:Schedule>
    </wsmp:Condition>
    <wsmp:Action IfCondition="true">
      <wsmp:RejectMessage/>
    </wsmp:Action>
  </wsmp:Rule>
</wsp:Policy>
```

The StartDate and StopDate attributes of the <wsmp:Schedule> element specify the effective date of a schedule. The StartDate represents the first day on which the schedule is effective, and the StopDate represents the first day that the schedule is ineffective. Note that this is not the same as the last day that the schedule is effective. The schedule shown in Listing 8 will reject messages...
received on July 1, 2, 3, and 4. It will allow messages on July 5th. A schedule with only a startDate is ineffective until the date specified. A schedule with only a stopDate becomes ineffective in the date specified.

Listing 8. Reject messages based dates

```xml
<wsp:Policy wsp:Name="Reject messages based on dates"
    <wsmp:Rule>
        <wsmp:Condition>
            <wsmp:Schedule StartDate="2012-07-01" StopDate="2012-07-05"/>
        </wsmp:Condition>
        <wsmp:Action IfCondition="true">
            <wsmp:RejectMessage/>
        </wsmp:Action>
    </wsmp:Rule>
</wsp:Policy>
```

You can combine the different facets of a schedule (WeekDays, Daily start/stop time, startDate, and stopDate) to form a more complex schedule. However, be careful when you construct more complex schedules to make sure the facets are not mutually exclusive and do not lead to unintended consequences as shown in the examples below.

- The startDate and stopDate ranges do not include any of the WeekDays specified.
- A schedule that specifies WeekDays or a stopDate will not truncate a Daily schedule that "crosses midnight". For example, Listing 9 rejects messages starting at 22:00 on Saturday night to 6:00 Sunday morning, and 22:00 on Sunday night to 6:00 Monday morning. The fact that "Monday" is not listed for WeekDays means that the duration for the daily schedule does not start on weekdays Monday through Friday. However, it does not mean that the portion of the schedule that spills over into Monday will be truncated.
- Times are based on the policy enforcement platform's local time.

Listing 9. Schedule containing Daily and Weekdays elements

```xml
<wsp:Policy wsp:Name="Schedule containing Daily and Weekdays elements"
    <wsmp:Rule>
        <wsmp:Condition>
            <wsmp:Schedule>
                <wsmp:Daily StartTime="22:00:00" StopTime="06:00:00"/>
                <wsmp:WeekDays Days="Saturday+Sunday"/>
            </wsmp:Schedule>
        </wsmp:Condition>
        <wsmp:Action IfCondition="true">
            <wsmp:RejectMessage/>
        </wsmp:Action>
    </wsmp:Rule>
</wsp:Policy>
```

Action

The actions defined in WS-MediationPolicy provide general coverage for a variety of operational IT requirement needs.
The Action element contains one or more actions to take if the evaluation of the logical condition described in `<wsmp:Condition>` matches the IfCondition attribute of the Action element. If no IfCondition element is specified for the Action, then IfCondition="true" is implied. In the example shown in Listing 2, the RouteMessage action is invoked for messages that are sent between midnight and 6:00 AM on Mondays, Wednesdays, and Fridays. Such a policy is useful for steering messages away from a Web service periodically for routine maintenance.

The actions that are supported within the Action element are:

- ValidateMessage
- RouteMessage
- QueueMessage
- RejectMessage
- ExecuteXSL
- Notify

The "ValidateMessage" action validates a message using an XML Schema (XSD) or a WSDL. If the XSD option is used, the ValidateMessage Action is declared as shown in Listing 10. If the WSDL option is used, then the ValidateMessage Action is declared as shown in Listing 11.

Listing 10. ValidateMessage with XSD

```
<wsmp:Action>
  <wsmp:ValidateMessage>
    <wsmp:XSD>
      local://MySchema.xsd
    </wsmp:XSD>
  </wsmp:ValidateMessage>
  <wsme:Scope>SOAPBody</wsme:Scope>
</wsmp:Action>
```

Listing 11. ValidateMessage with WSDL

```
<wsmp:Action>
  <wsmp:ValidateMessage>
    <wsmp:WSDL>
      local://MyWebService.wsdl
    </wsmp:WSDL>
  </wsmp:ValidateMessage>
</wsmp:Action>
```

As shown in Listing 10, the XSD option for ValidateMessage also allows a Scope to be defined for message validation. Valid values for Scope are SOAPBody, SOAPBodyOrDetails, SOAPEnvelope, and SOAPIgnoreFaults. The meaning of these values is listed in Table 4.

Table 4. Valid values for Scope and their meanings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPBody</td>
<td>The contents of the SOAP Body element, without special processing for SOAP faults. This is the default behavior if the Scope is not specified.</td>
</tr>
<tr>
<td>SOAPBodyOrDetails</td>
<td>The contents of the detail element for SOAP faults, and the contents of the Body otherwise.</td>
</tr>
<tr>
<td>SOAPEnvelope</td>
<td>The entire SOAP message, including the envelope.</td>
</tr>
</tbody>
</table>
The "RouteMessage" action changes the remote destination (backend) for the policy enforcement to forward the request after being processed by a proxy. This action is performed by substituting the backend URL defined in the enforcement gateway (for example, provided within the WSDL or Remote Endpoint configured in a Web Service Proxy) with the information provided by the RouteMessage Action. As an example, let's assume that all of the services for a Web Service Proxy are configured to use a Remote Endpoint of "http://myPrimaryServer.myco.com:80". You can use the example shown in Listing 12 to reroute messages to "https://myAlternateServer.myco.com:9443" instead.

Listing 12. RouteMessage example

```xml
<wsmp:Action>
    <wsmp:RouteMessage>
        <wsmp:EndPoint>
            https://myAlternateServer.myco.com:9443
        </wsmp:EndPoint>
    </wsmp:RouteMessage>
</wsmp:Action>
```

The "QueueMessage" action, as shown in Listing 13, causes messages to be held back for processing at a later time as dictated by the Expression element, which we will cover later. The QueueMessage action is unique in that it only applies when an <Expression> is declared and evaluates to true. Furthermore, if multiple <Action> or <Action IfCondition="true"> elements are declared, then QueueMessage must be in the first one.

Listing 13. QueueMessage example

```xml
<wsmp:Action IfCondition="true">
    <wsmp:QueueMessage/>
</wsmp:Action>
```

The RejectMessage action, shown in Listing 14, causes a message to be rejected. There are no restrictions on the usage of RejectMessage. It may be used for any logical condition. Message processing stops when a RejectMessage is encountered and an error response is sent back to the service consumer.

Listing 14. RejectMessage example

```xml
<wsmp:Action>
    <wsmp:RejectMessage/>
</wsmp:Action>
```

"ExecuteXSL", as shown in Listing 15, provides a mechanism for message format translation (non-semantic data mediation). This is provided to allow a message to be converted from one version to another version before it is forwarded to the backend Web service for processing.
Listing 15. ExecuteXSL example

```xml
<wspmp:Action>
  <wspmp:ExecuteXSL>
    <wspmp:Parameter Name="SourceVersion" Value="V1" />
    <wspmp:Stylesheet/>
    local:///convertMessageVersion.xsl
  </wspmp:Stylesheet>
</wspmp:ExecuteXSL>
</wspmp:Action>
```

"Notify" provides a mechanism for creating a "notice" level log entry from policy. The DataPower appliance will generate a message event in the system log when it encounters a Notify action as shown in Listing 16.

Listing 16. Notify example

```xml
<wspmp:Action>
  <wspmp:Notify/>
</wspmp:Action>
```

The log entry created by the Notify action is shown in Figure 3. Note that the action name is auto-generated by appliance and subject to change, but the message ID (0x80800101) will be the same for all Notify messages.

Figure 3. Log entry produced by Notify

```
request 99.41432126 0x80800101  
```

Authoring WS-MediationPolicy

This section covers the different choices available when creating WS-MediationPolicy expressions and using them as a DataPower appliance. We will cover three methods that use WSRR, Integration Designer, and a simple text editor.

Authoring with WSSR

WSRR V8.0 has added built-in support for creating WS-MediationPolicy documents with a dedicated user interface editor within the WSRR Business Space UI.

WSRR V7.5 has support for WS-MediationPolicy using the traditional policy editor in the WSRR Web UI console. This article is focused on the new WSRR V8.0 support. For information about how to use WSRR V7.5, see Policy authoring tasks.

To author a WS-MediationPolicy assertion using the WSRR Business Space UI, first ensure that the system administrator has configured the space for the role appropriately. The space used must have been created based on one of the following templates:

- Service Registry for Development
• Service Registry for Operations
• Service Registry for SOA Governance

Alternatively, you can use a custom space. In this case, you must configure the "Service Registry Actions" widget to display a "Create a Policy" action, with a policy type of "Mediation".

To start the process of authoring a new WS-MediationPolicy assertion in the WSRR Business Space UI, first locate the "Service Registry Actions" widget. On a space created from one of the above templates, it will be found on the "Overview" tab. This widget contains the action link shown in Figure 4, which when clicked, starts the authoring process. Note that the text in the action link may be different if the space used is not a space created from one of the templates.

**Figure 4. Create a Mediation Policy action**

Once clicked, the Create a Mediation Policy link, presents the same dialog, as shown in Figure 5, that is used to create a new instance of a Business Modeled object with an extra section entitled "Policy".

**Figure 5. Create a Mediation Policy dialog**

![Create a Mediation Policy dialog](image)

The Policy section represents a single Rule element in the XML. To author multiple rules, you must create multiple WS-MediationPolicy assertions in WSRR. This is a current limitation of WSRR Business Space policy editor. When first entered, the dialog will have the option to add an "Attribute Condition", a "Schedule Condition", and multiple "Actions". The actions added in the "Actions" section are to be performed when there are no conditions present or, once some conditions have been added, as shown in Figure 6, the actions to be performed when the conditions evaluate to true. Once either type of condition has been added, the action section
expands to include an option to add actions that will be performed when the conditions evaluate to false.

**Figure 6. A Mediation policy with one condition added**

![Mediation policy with one condition added](image)

The icon in **Figure 7**, when hovered with the mouse, provides more information for that part of the user interface.

**Figure 7. An Information icon**

![Information icon](image)

### Attribute conditions

After clicking **Add Attribute Condition**, an inline area, as seen in **Figure 8**, appears that allows you to select the attribute the condition will apply to, while also providing a description of the attribute. After the attribute condition has been added by clicking **Add**, the parameters of the condition may be set.

**Figure 8. Adding an attribute condition**

![Adding an attribute condition](image)

The attribute condition is one of the following values that correlate directly with the values defined in **Table 3**:

- Message count
- Error count
- Backend latency
- Internal latency
• Total latency

The operator is one of the following values that correlate directly with the values defined in Table 2, with "Greater Than, Allowing Bursts" mapping to "TokenBucket", and "Greater Than, Until Less Than" mapping to "HighLow":

- Greater Than
- Greater Than, Allowing Bursts
- Greater Than, Until Less Than
- Less Than

Note that the "Greater Than, Allowing Bursts" operator is not available if a latency attribute condition has been chosen, as this operator only applies to counts.

The operators behave as their names suggest. As shown in Figure 6, a condition with an attribute of "Message Count", an operator of "Greater Than", a "Value" of 100, and a "Per interval of" of 30 seconds has been defined. This means that this condition will evaluate to "true" if the number of messages, measured over a 30 second interval, is greater than 100. Once the number of messages drop below 100 measured over the same interval, then the condition will again evaluate to "false".

An asterisk to the left of the field name indicates that the field is mandatory. Note that the WSRR interface considers some fields to be mandatory that are optional according to the XML definition. This ensures that specific values are provided at authoring time, instead of allowing on default values defined by the enforcement point, meaning that the semantics of the policy are entirely deterministic.

The fields that are available for input will vary in number and name according to the attribute and operator chosen. In Figure 9, we have selected the "Greater Than, Allowing Bursts" operator so the first field has changed to "Normal maximum". We have an extra field called "Allowing bursts of up to". In this example, the "Greater Than, Allowing Bursts" operator behaves just like the "Greater Than" operator, with the exception that a burst of 200 messages over a 30 second interval, or two consecutive bursts of 150 messages over two 30 second intervals, and so on, will not cause the condition to evaluate to true.

Figure 9. A condition that uses the "Greater Than, Allowing Bursts" operator

In Figure 10, "Greater Than, Until Less Than" has been selected in the operator dropdown. This operator behaves exactly like "Greater Than". The exception is that once it has begun to evaluate
to true, it will continue to evaluate to true until the number of messages has dropped below 50, as indicated in the "Until less than" field.

**Figure 10. A condition that uses the "Greater Than, Until Less Than" operator**

![Condition Diagram](image)

The "Less Than" operator presents the same fields as the "Greater Than" operator, but evaluates to false while the measured value is above or equal to the defined value, and true once the measured value is below the value defined in the field named "Value".

**Schedule conditions**

By clicking **Add Schedule Condition**, a condition that only evaluates to "true" during a specific time period can be added. Dates for the period to start and end on, days of the week to include, and also restrictions on the time of day that is included in the schedule for each day can be specified. A simple schedule is shown in **Figure 11**. This schedule will evaluate to true for the period of one year and one day starting on the 2nd of April 2012. Note that the "Start date" and "End date" fields are inclusive. For the benefit of those who are used to working with the XML "Stop date", which is exclusive, it is indicated in the text displayed immediately below the end date.

**Figure 11. A simple schedule condition**

![Schedule Condition](image)

When days of the week and time of the day are also specified, the schedule condition looks like the example shown in **Figure 12**. This example will evaluate to true for the period of one year and one day starting on the 2nd of April 2012, but only on Monday to Friday, from 8 PM in the evening until 8 AM in the morning. Due to the times of day specified in this example running over a day boundary, the schedule will still evaluate to true on Saturday, even though Saturday is not included in the schedule, until 8 AM in the morning, as indicated by the text "The next day" in the example. The schedule will also still evaluate to true on Wednesday the 3rd of April until 8am in the morning, as the interval starting at 8 PM and running until 8am will have been started on Tuesday the 2nd of April.
Figure 12. A complex schedule condition

![Schedule Condition](image)

**Actions**

Clicking **Add Action** allows the addition of an action to be performed. If there are no conditions specified for the policy, then the actions will always be executed. If there are conditions specified for the policy, then the actions in the relevant section, "Actions If All Conditions are True" or "Actions If Any Condition is False", will be executed based on the result of evaluating the conditions. The actions available for use in the policy are:

- ValidateMessage
- RouteMessage
- QueueMessage - only available in the "Actions If All Conditions are True" section
- RejectMessage
- ExecuteXSL
- Notify

After clicking **Add Action**, an inline area (see **Figure 13**) appears that allows you to select an action to perform while it also presents a description of the action. After the action has been added by clicking **Add**, the parameters of the action may be set and more actions may also be added by clicking the **Add Action** link again.

Figure 13. Adding an action area

![Add Action](image)
The "Notify", "Queue Message", and "Reject Message" actions have no parameters, as seen in Figure 14.

**Figure 14. Actions with no parameters**

![Actions with no parameters](image)

"Route Message", as shown in Figure 15, takes as an endpoint as the parameter to route the message. This endpoint can be any textual input and is interpreted by the enforcement point at runtime.

**Figure 15. Route message action**

![Route message action](image)

"Validate Message" requires either an XSD or WSDL file to run the validation of the message against. The drop down in the title bar of the action allows for the selection of which type of file the action will use. You can use the radio buttons at the top of the action to decide whether to type in a URL to identify the file using "Enter the XSD document URL". Or, if the file stored in WSRR, use "The XSD document is in the registry" to search for the file by name. If electing to validate by XSD, then specify the "Validation Scope" using the radio buttons at the bottom of the action. Figure 16 shows a validation by an XSD that is identified by a URL.

**Figure 16. Validate message by XSD and action**

![Validate message by XSD and action](image)
**Figure 17** shows a validation by a WSDL. The user is about to identify the WSDL file from WSRR that is using a suggestion given by the search field, after typing basicE into the field. Note that the "Validation Scope" field is not present for the validation by WSDL.

**Figure 17. Validate message by WSDL, action**

The "Execute XSLT Transform" action requires the URL of the stylesheet to use for the transformation and, optionally, some parameters to pass to the stylesheet. **Figure 18** shows a simple example of this action, where there are no parameters needed. Note that the "XSLT stylesheet URL" must start with either "store://" or "local://".

**Figure 18. Execute XSLT Transform with no parameters**

In **Figure 19**, the user has selected the checkbox to specify some parameters. You can add name and value pairs that are passed to the stylesheet when they are executed by the enforcement point. You can add a new parameter by clicking **Add Parameter**. You can reorder these parameters by using the arrow icons at the left of each row, or by dragging and dropping. A parameter may have no value as you can use this to override a default value specified in the stylesheet with a blank value. You can leave empty parameter rows in this section and they will simply be factored out when the policy is saved. However, a parameter value without a parameter name is not valid, as indicated by the red highlight of the name field for the parameter with the "valueC" value.

**Figure 19. Execute XSLT Transform with some parameters**
Saving and editing

After the policy is complete, click the Finish button in the dialog to save the policy to WSRR. If the Finish button is not enabled, then there is a mandatory field. This field is indicated by an asterisk to the left of the field name that has not been filled in, or there is an invalid value in one of the fields. This value is indicated by either a red border round the field, as shown in Figure 19, or a warning icon to the right of the field, as shown in Figure 20.

Figure 20. A field with an invalid value

After the policy has been saved to WSRR, the detail widget will load a view of the policy as shown in Figure 21. When viewing the policy in the detail widget, only the sections that have some content will be shown. In this case, the section titled "Actions If Any Condition is False" has not been displayed as there were no actions contained within it. In the example, a "Source Document" section is also seen. This is a link to the details of the document that WSRR created when the policy was saved. The document is the physical artifact that contains the XML generated by WSRR to represent the policy. Note that amount of detail available when viewing the details of a policy may look different if the system administrator has configured the detail view differently.

To edit the policy after it has been saved, click on the pencil icon in the top right corner of the detail widget. You can then edit the policy in the same way as it was created.

Figure 21. A completed WS-Mediation policy

Using the Web user interface

It is also possible to create WS-MediationPolicy assertions using the Web user interface. Note that this is no longer the recommended way to create WS-MediationPolicy assertions in WSRR. To
learn more about this method, see the tutorial Working with the policy authoring tool in the WSRR Information Center.

**Authoring with IBM Integration Designer**

IBM Integration Designer V7.5 supports authoring, uploading, and testing of DataPower artifacts including XML Schema, WebServices, XML, and XSL transformations.

You can also use Integration Designer to create and modify a WS-MediationPolicy by creating a DataPower library and creating a WS-MediationPolicy within it.

There are two editors available in Integration Designer to modify a WS-MediationPolicy:

- The WS-MediationPolicy editor plugin sample provides a guided entry method for creating policy.
- The XML editor is available without the requirement of using the plugin sample provided in this article, but it does not provide the guided entry or schema validation.

**Using the WS-MediationPolicy Sample Editor plugin within Integration Designer**

In this article, we have provided a sample WS-MediationPolicy editor, which you can install in Integration Designer by unzipping the supplied zip file in the `<IID Install Directory>/drops` directory.

This plugin exposes a WS-MediationPolicy wizard and WS-MediationPolicy Editor to create and modify the WS Mediation policies.

**Creating a new WS-MediationPolicy**

To create a new WS-MediationPolicy, you must first launch Integration Designer, and then launch the WS-MediationPolicy wizard by selecting Business Integration > WS-MediationPolicy wizard.

The WS-MediationPolicy contains a single wizard page (Figure 22), which requires a DataPower Library and the name of the WS-MediationPolicy file (Figure 23).
Figure 22. First panel of the WS-MediationPolicy creation wizard

![First panel of the WS-MediationPolicy creation wizard](image)

Figure 23. Final panel of the WS-MediationPolicy creation wizard

![Final panel of the WS-MediationPolicy creation wizard](image)

Editing and modifying a WS-MediationPolicy

To edit or modify a WS-MediationPolicy, you can open the **Physical Resources** view in Integration Designer and open the created WS-MediationPolicy file. This launches the WS-MediationPolicy editor (Figure 24) to edit and modify a WS-MediationPolicy.

The editor contains two tabs, the Design editor and the Source editor. Both views are kept in synch so if you modify the Design view, the Source view is updated and vice versa.
Using the XML Editor to edit a WS-MediationPolicy within Integration Designer

Integration Designer has an XML Editor that you can use to edit an XML file. The easiest way to create a WS-MediationPolicy is to create an XML file within a DataPower project and place this skeleton XML snippet inside the newly created XML file, as shown in Listing 17.

Listing 17. Empty policy with WS-MediationPolicy namespace declared

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
</wsp:Policy>
```

After the WS-MediationPolicy is created, you can upload it to a DataPower appliance directly from within Integration Designer using the DataPower Appliance View.

For more information, see Using Integration Designer with WebSphere DataPower.

Authoring with an editor

If WSSR or Integration Designer is not available, you can still create a WS-MediationPolicy using any preferred XML or text editor. Figure 25 shows a simple text editor that creates a new mediation policy based on a template cloned from one in the appliance `store://policies/templates` directory.

Figure 25. Creating WS-MediationPolicy in a text editor
This policy routes messages to a new server. It is a best practice to define Name (wsp:Name) and Id (wsu:Id) attributes for your policy. Now that the policy is defined, you can attach it to a WSDL service using the DataPower WebGUI.

The steps to attach a policy are simple (see Figure 26). First, you must know where you want to attach the policy. In this case, we chose the service policy subject, and we selected the WS-Policy button for the service (1). The policy is uploaded to the "local://" directory for the domain (2), and attached it (3). Finally, when we press Done (4), the changes are applied to the Web service proxy. After attaching the policy, all messages that come into the service that has the route policy attached are routed to the new server specified by the policy.

Figure 26. Attaching policy in the Web GUI

One thing you may be asking right now is: "How would the route policy work if the backend was "https://", instead of "http://"?" DataPower supports policy parameters to influence how the policy is enforced on the appliance. In the case of https, an SSL Profile policy parameter is needed so that the proxy can locate the necessary key or certificate to properly negotiate the SSL connection to the new backend server. A full list of policy parameters supported for WS-MediationPolicy are shown in Table 5.

Table 5. List of policy parameters for WS-MediationPolicy and their meanings

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Priority</td>
<td>Sets the log priority for log messages that result from the Condition policy assertion.</td>
</tr>
<tr>
<td>SSL Profile</td>
<td>Sets the SSL Proxy Profile that gets used by the RouteMessage policy assertion. This is needed to handle the client-side part of the SSL handshake to a backend that requires SSL.</td>
</tr>
</tbody>
</table>
Output Type
Sets the output type for the ExecuteXSL policy assertion. Valid values are xml, binary, and default.

SLM Peer Group
Sets the SLM Peer Group to be used by the Condition policy assertion. Peer groups enable the exchange of SLM data among a group of appliances.

SLM Credential Class
Sets the SLM Credential Class to be used by the Condition policy assertion. An SLM credentials class identifies a set of users (credentials) to be subject to an SLM policy. A Credential Class is automatically generated for SLA policy, but the Policy Parameter will override the default.

SLM Resource Class
Sets the SLM Resource Class to be used by the Condition policy assertion. An SLM resource class identifies a set of resources to be subject to an SLM policy.

Configuration Optimization
Configuration generated from each assertion encountered in policy, even if it is unreachable. This is useful for debugging policy, but requires extra capacity at runtime to execute the additional processing actions. The Configuration Optimization parameter, when set to "on", tells the appliance to discard unreachable policy assertions.

WSRR Server
The ValidateMessage policy assertion may use a URL that refers to a WSRR server. This parameter is used to resolve the WSRR server configuration to use if it cannot be determined automatically.

Using WS-MediationPolicy to separate the policy intent ("what to do") from the enforcement platform configuration ("how to do it") has many benefits, for example:

- Separation of concerns between operational policy developers and DataPower administrators by supporting policy documents that contain policy intent that is void of configuration details.
- Facilitates the adoption of policy management and governance processes by embracing products, such as WSRR.
- Allows the scalability of DataPower appliance adoption by increasing the number of individuals who can create policy to affect service gateways.
- Leverages industry standards specifications to decrease learning time and increases the tooling available.

Conclusion

This article presented different methods of authoring WS-MediationPolicy documents, such as the WSRR V8.0 mediation editor, the Integration Designer policy editor plug-ins (download plug-in), and the most common XML editors. The best overall method depends on your development process requirements, as well as the adopted policy governance processes. Regardless of which method you choose to create policies, WS-MediationPolicy enables you to focus on solving business IT requirements instead of operational configuration changes.

Finally, it is important to remember the common service gateway enforcement patterns that this new policy vocabulary allows you to express:

- Traffic management (QoS)
- Routing
- Message validation
- Message translation (versioning)
You can quickly deploy these patterns using the WSRR V8.0 policy editor or copying the example templates provided in the appliance, and then attaching them to desired Web service. WS-MediationPolicy also supports a series of policy configuration parameters that can control how configuration choices are generated.

This article showed how you can take advantage of this versatile feature that can greatly ease and accelerate your path to adopting SOA governance and policy management objectives. Future articles in this article series will cover how to declare Service Level Agreements by using message content filters in conjunction with WS-Policy attachments and how to create your own custom policy domains for DataPower.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-MediationPolicy schema</td>
<td>ws-mediationpolicy-1.6-2011-02.xsd</td>
<td>5KB</td>
</tr>
<tr>
<td>Plug-in for IBM Integration Designer</td>
<td>com.ibm.wbit.wdp.policy_8.0.0.zip</td>
<td>41KB</td>
</tr>
</tbody>
</table>
Resources

- WS-MediationPolicy 1.6 specification
- Web Services Policy 1.5 - Framework specification
- Web Services Policy 1.5 - Attachment specification
- WebSphere Service Registry and Repository Information Center: Authoring policies in IBM WebSphere Service Registry and Repository
- Business Process Manager V7.5 Information Center: Using IBM Integration Designer with WebSphere DataPower
- WebSphere DataPower SOA Appliance forum
About the authors

**Mario E. De Armas**

Mario De Armas is a Software Architect in the WebSphere DataPower Appliance development team and has worked on many different areas and features of DataPower appliances. He currently leads DataPower’s support for WebSphere Service Registry and Repository integration, Web services, WS-Policy, and SOA Governance functionality. His past DataPower projects included Technical Lead for XB60, AS1 protocol handler, B2B gateway, appliance management functionality, database integration, and user interface customization features.

---

**Thomas Burke**

Thomas Burke is an Advisory Software Engineer on the WebSphere DataPower Appliance development team. Thomas has been on the DataPower team for Governance capability. Prior to his role as an SOA Governance Team Lead in DataPower, Thomas served as a Team Lead on the WebSphere Business Monitor team and the Architect for an early version of the IBM Mobile Portal Accelerator.

---

**Evan Jardine-Skinner**

Evan Jardine-Skinner is a User Interface Software Developer on the Service Registry team at the IBM Software Lab in Hursley, United Kingdom. He joined IBM as a member of the Voice Systems Services team in 1998 and later used his services experiences to great effect in the development of Voice Response, mainly working on the ISDN signaling layers. He gained experience in porting IBM applications to new platforms working in the porting center, before his passion for usability drove him to join the Service Registry UI team in 2007. Evan graduated with an Honors degree in Computer Science from Bradford University, United Kingdom.

---

**Dave Spriet**

Dave Spriet is the Mapping and Transformation Tools Architect at the IBM Toronto Software Lab, Canada. He architects tools for Business Process Management and Connectivity and specializes in XML and transformation technologies, which consist of XML, XSLT, XQuery, XPath, and Java. He has a Bachelor’s degree (with Honors) in Computer Science and Statistics from McMaster University, Canada.