Spagic JBI Components

Author: Andrea Zoppello

Document Goal

This document describes the components and services, available in Spagic Studio when on a BPMN Diagram you choose "JBI – Apache Service Mix" as the technology for a BPMN Pool.

Version History

Version/Release n°:	1.0	**Date**	January 14, 2008
Description/Modifications:			

1 Preliminary Concepts

1.1 BPMN

The new Spagic Studio 2, is quite different from the previous one, in particular the main difference is that this version is targeted not only to ESB or technology specialist but also to “business analyst”.

In our vision the development of a Process start from business analyst that captures customer requirements, and design the process in a very simple and graphical formal way.

As you probably already know a very popular language to define processes, is Business Process Modeling Notation defined by OMG at http://www.bpmn.org/, for this reason we choose to start form BPMN to model Spagic processes.

The “main problem” of BPMN is that in his actual form, it’s not an executable language (like for example BPEL), so with a standard BPMN editors is not possible to get some deployable/executable process. To solve this problem we need to find a way to annotate a BPMN diagram with technology details, and then to use the “ BPMN Annotated Model ” to get some deployable code.

The “annotation” of BPMN diagrams to get a specific technology model and the code generation from that model, is done by a technology specialist that use Spagic specific tools to transform the original BPMN model in something richer.

Spagic Studio v2 aims, to provide a complete set of tools, to fill both the requirements of business analyst and technology specialist.

The development lifecycle of the process could be resumed in the following steps:

1. Business Analyst use BPMN Diagram Editor to develop the business process in standard BPMN Notation.
2. Technology specialist “choose the technology” for each single process (Pool) in the BPMN Diagram.
3. After choosing the technology, the specialist provide annotations for each step of the business process, with specific services offered by a particular runtime technology. This is done by drag and drop of services to the BPMN diagram.
4. With the annotated model, it’s possible to get directly executable applications by using specific code generators.

In the next sections of this document we’re going to see how to start from BPMN design, and then how to use a specific JBI Runtime. The second part of the document contains the list of services offered by JBI-ApacheServiceMix Runtime.

1.2 The Role of Eclipse STP Intermediate Model

As we already explained, in the new Spagic Studio we decoupled the roles of business analyst from the roles of technology specialist, and BPMN is a very good starting point for doing that.

The problem is now to define what happens after the ***"Process Design"*** level phase: it's necessary to define how the BPMN annotated/enriched model will be transformed in something deployable in an ESB, or in a workflow engine.

As you probably know a ***lot of standards and specific editors tools exist in SOA/BPMN landscape***.
A possible approach for the BPMN transformation is implementing direct transformations from BPMN into, for example, JBI Service Assemblies.
This approach has some important problems:

1. There will be a high level of coupling of the ***"Process Design phase"*** with the ***"Process Deployment/Execution Phase"***. This means that if I want to use another process designer that is not a BPMN Editor I must rewrite the transformation code.

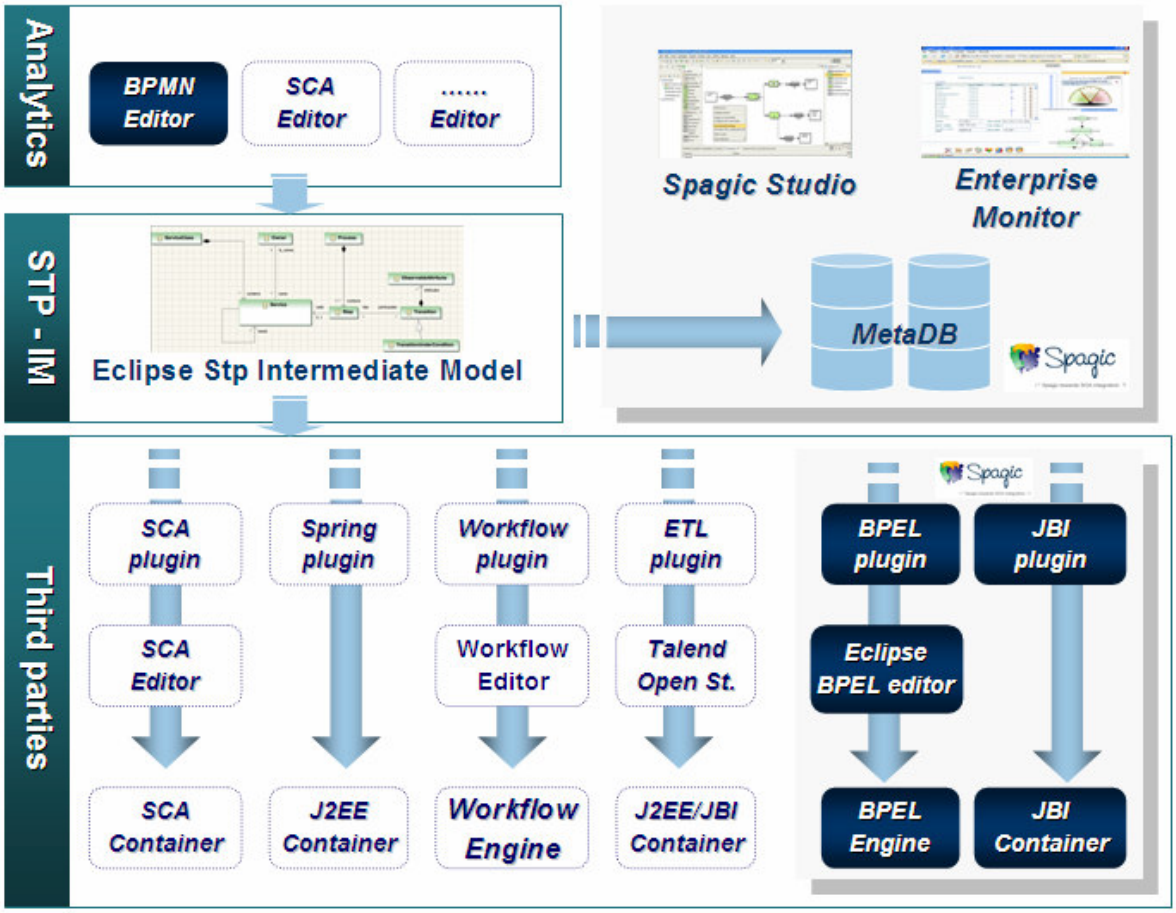
2. Having a direct transformation does not allow sharing "Process Level Information" with different tools: for example there is no way to interact with a SCA editor.

For this reason in the new Spagic Studio, we choose to separate the ***"Process Design phase"*** with the ***"Process Deployment Execution Phase"*** using a process generic model defined in the Eclipse STP subproject called STP Intermediate Model (http://www.eclipse.org/stp/im/index.php).

This document doesn't explain the Intermediate Model, for this you could use the information on the official project site, but it's important to notice that this approach has several advantages:

1. From BPMN we must write **only one transformation** to the Intermediate Model.
2. For each runtime we provide a transformation plugin that starts from Intermediate Model, and we should do the same also starting from BPMN (and not from IM, as we do). The advantage is that then **every tool able to produce Intermediate model could take advantage of existing transformations.**

The picture below resumes interactions between Spagic and STP Intermediate Model.

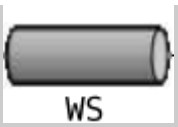


For all the above reason when with Spagic Studio you need to generate deployable code for a particular runtime, you need to generate the Intermediate Model first.

2 JBI Components

2.1 Binding Components

2.1.1 HTTP Component

HTTP Binding Component	
Component family	Standard, Binding Component
Service	StartService/EndService
Service Bindings	• JBI-HttpInputBindingComponent • JBI-HttpOutputBindingComponent • JBI-HttpInputOutputBindingComponent

Function	The Http component permits to communicate on socket connection using HTTP/SOAP protocol.

2.1.1.1 Configuration

Configuration properties	
Component Name	Name of the component
Location URI	The http url where this proxy endpoint will be exposed or the url of the target service.
soap message	If set, the component will parse the soap request and send the content into the NMR
soap version	Version of soap to use, options: ❑ 1.1 ❑ 1.2
soap action	Soap action that will be putted int the header (note the SOAPAction is not available on SOAP v. 1.2 and is discouraged for interoperability reasons)
enable https(ssl)	Set the use of communication encryption.
Client Authentication	The component will accept only connection where also the client is authenticated with a "Digital Certificate".
Keystore	Name of the keystore containing server certificates. The name is a path into the ESB classpath
Keystore Password	Keystore password.
Keystore type	Type of the keystore, options: ❑ JKS ❑ PKCS12
Trust Store	Name of the truststore containing server certificates. The name is a path into the ESB classpath
Trust Password	Truststore password.
Truststore type	Type of the truststore, options: ❑ JKS ❑ PKCS12

2.1.1.2 SSL configuration

The component can be configured to encrypt the communication using SSL. The encryption implies the use of digital certificates; using the configuration properties is possible to configure both the server either the client authentication. This imply the component need a keystore, where the key needed to encrypt will be searched, and a truststore used to trust the certificate of the other side of the connection.

The configuration of the “Client Authentication” permit to ensure the mutual authentication of both sides of the connection.

Keystore and truststore have to be provided to the ESB putting them on its classpath; for example (using ServiceMix) put a generated keystore in a jar with the name “conf/serverKeystore.jks” and copy it into the lib folder, than valorize the “Keystore” property with “conf/serverKeystore.jks” to indicate the keystore to be used.

2.1.1.3 SOAP with attachments

The HTTP component can manage also SOAP calls with attachments: the attachments are stored within the message exchange. The message exchange is composed of the following three sections:

- Header
- Payload
- Attachments

The components invoked after the HTTP BC can then elaborate the attachments using the JBI API to retrieve them.

2.1.1.4 Synchronous component interaction

When you use the "JBI-HttpInputOutputBindingComponent" Spagic studio will automatically generate a Synchronizer Endpoint, in this way is possible to obtaining a synchronous behaviour and permit to use the same binding component (HTTP) as entry and exit point of the process.

In the old version of Spagic studio, it was necessary to put make explicit the usage of the synchronizer component and the connections to it. There's a specific section at the end of this document to make this.

In the new version, the usage of an InputOutput component automatically means the use of the Synchronizer component, and when you want the process to reply it's simply enough to annotate an end service with the "EndService/ReplyService" annotation.

2.1.1.5 Webservice client generation

To call a process where the entry point is an HTTP binding component is possible to generate a Java client using an Eclipse wizard. After the creation of the intermediate model process is possible to generate the WSDL files describing the exposed webservice; this operation can be achieved using the right mouse button on a Spagic process and selecting "Generate WSDL For ServiceMix". The webservice "Generate Client" function can be activated with a mouse right click on the generated WSDL file.
When the wizard terminate the stub generation is possible to invoke them with a simple Java application that invokes the generated proxy class. Follow an example of Java code that invokes the proxy:

```
…
public static void main(String[] args) throws Exception {
        MyWebServiceadvQueryTestProxy proxy = new MyWebServiceadvQueryTestProxy();
        StringBuffer request = new StringBuffer()
                .append("<jforum><userletter>%nonymou%</userletter>"
                        +"<userpassword>%o%</userpassword></jforum>");
        SOAPElement envelope = createSOAPMessageFromString(request.toString());
        proxy.run(envelope);
}
…
```

Spagic is released with an utility library (soapclientutils.jar) that can be used to easly generate soap messages like in the `createSOAPMessageFromString(request.toString())` call of the preceding java code.

2.1.1.6 Configuration examples

2.1.1.6.1 Sample SOAP

```
<http:endpoint
        defaultMep="http://www.w3.org/2004/08/WSDL/in-only"
        service="foo:myScreenOutputhttp-course"
        endpoint="myWebServicehttp-course"
        role="consumer"
        locationURI="http://0.0.0.0:8000/TestHttp/"
        soap="true"
        soapVersion="1.2"
        soapAction="">
</http:endpoint>
```

2.1.1.6.2 Https example

```
<http:endpoint
        defaultMep="http://www.w3.org/2004/08/WSDL/in-only"
        service="foo:myScreenOutputhttps-course"
        endpoint="myWebServicehttps-course"
        role="consumer"
        locationURI="https://0.0.0.0:8888/httpsCourse/"
        soap="true”
        wsdlResource="classpath:https-coursemyWebServicehttps-course.WSDL">

        <http:ssl>
                <http:sslParameters
                        keyStore="classpath:engks/andrea.p12"
                        keyStorePassword="andrea"
                        keyStoreType="PKCS12"
                        wantClientAuth="false"
                        needClientAuth="false" />
        </http:ssl>
</http:endpoint>
```

2.1.2 TCP-IP

TCP-IP Binding Component	
Component family	Standard, Binding Component
Service	StartService
Service Bindings	TCPInputOutputBindingComponent
Function	The TCP component permit to communicate on socket connection using a simple protocol based on header and trailer.

2.1.2.1 Common config

Common Configuration properties	
Component Name	Name of the component
Connection Point type	permit the component to act as a server (POINT_TYPE_SERVER) or a client (POINT_TYPE_CLIENT)
Connection Point mode	configure how the connection point communicate: ❑ OPERATION_MODE_BIDIRECTIONAL: receive or send messages simultaneous (not implemented as client type the component always wait for a response): ❑ OPERATION_MODE_IN: only receive messages ❑ OPERATION_MODE_IN_OUT: When a message is received, the system will refuse to accept further messages until a response has been sent ❑ OPERATION_MODE_OUT: only send messages ❑ OPERATION_MODE_OUT_IN: When a message is sent, the system waits for a response before sending the next message.
Normalize message output envelope	name of the envelope where TCP message is putted sending the message to other components
Normalize message input envelope	name of the envelope where other component puts this component input message.
Base64 encode outgoing message	If TRUE tcp content putted in the Normalized Message will be encoded
Base64 decode incoming message	If TRUE content read from Normalized messages will be decoded.
Use SSL	Set the use of communication encryption.
Use SSL client mode	The component will accept only connection where also the client is authenticated with a "Digital Certificate".
Keystore filename	Name of the keystore containing server certificates. The name is a path into the ESB classpath
Keystore password	Keystore password.
Truststore filename	Name of the truststore containing server certificates. The name is a path into the ESB classpath
Truststore password	Truststore password.
Log connections	If TRUE the connections logging will be enabled
Log data	If TRUE the data logging will be enabled

Log data in hexadecimal	If TRUE the data will be logged in hexadecimal format
Log filename	Name of the file where the log will be putted
Log extra info	Options: ❑ None ❑ Log Time If set to Log Time, the timestamps will be logged with each event
Incoming wrapper	Message wrapping around messages received over the socket connection. Options: ❑ Minimal - HL7 minimal LLP protocol, ❑ User - user defined header and/or trailer
Strip wrapping	Sets whether or not to strip the wrapping off received messages
Incoming header	This property defines the header that identifies the start of a message on the socket connection. [1]
Incoming trailer	This property defines the trailer that identifies the end of a message on the socket connection. [1]
Incoming endianness	Certain codes used in the header and trailer definitions can output binary data. Options: ❑ BIG_ENDIAN (most significant byte first) ❑ LITTLE_ENDIAN (least significant byte first) endian order
Outgoing wrapper	Outgoing wrapper used for messages written to the socket connection.
Outgoing header	This property defines the header that identifies the start of a message on the socket connection.[1]
Outgoing trailer	This property defines the trailer that identifies the end of a message on the socket connection. [1]
Outgoing endianness	Certain codes used in the header and trailer definitions can output binary data. Options: ❑ BIG_ENDIAN (most significant byte first) ❑ LITTLE_ENDIAN (least significant byte first) endian order

[1] The header or trailer is identified by sequence of bytes separated by a blank whith each byte in the form 0x##, where ## is the hex rappresentation of the byte (a valid example is: “0x1C 0x0D”)

2.1.2.2 Client mode configuration

Client mode Configuration properties	
Response timeout	indicate how long the component wait for a response before restart to send messages
Retry count	number of retry in case of no response to a sent message
Fail action	(Not implemented. Need queue management)
Retry number	Number of retry done on connection failure.
Retry type	specify how the component will retry to open a connection after a connection initialization failure ❑ RETRY_TYPE_NO_RETRY: no retry will be done ❑ RETRY_TYPE_IMMEDIATE: the retry is done as soon as the failure happen ❑ RETRY_TYPE_LINEAR: the component wait the "Retry delay" time before a retry ❑ RETRY_TYPE_EXPONENTIAL: the component will wait for the "Retry Delay" before attempting to restore the connection the first time, but for each subsequent attempt, this delay is raised to the power of 2.
Retry Delay	Wait time before retrying to establish the connection after a connection error.
Tcp out-in receiver classname	Name of a class that will receive the TCP server response (deprecated)

2.1.2.3 Server mode configuration

Server mode Configuration properties	
Connection number	Number of connection allowed receiving messages. Clients that connect after the connection number is reached will be disconnected.
Local port	listening port
Local address	listening host
Listen backlog	The number of connections that can be waiting for a connection before others are refused. The specified backlog must be greater than 0 (zero).

2.1.2.4 Configuration example

```
<tcp:consumer
        service="foo:TCPBCtcp-consumer-sync" endpoint="TCPBCtcp-consumer-sync"
```

```
            targetService="foo:mySynctcp-consumer-sync" >
        <tcp:config>

            <tcp:tcpBCConfig
                    localPort="10300"
                    localAddress="192.168.20.110"
                    connectionMode="CONN_MODE_MAINTAIN"
                    remotePort="10222"
                    remoteHost=""
                    incomingWrapper="WRAPPER_USER"
                    incomingHeader="0x0B"
                    incomingTrailer="0x1C 0x0D"
                    stripWrapping="TRUE"

                    logConnections="FALSE"
                    logData="FALSE"
                    logDataAsHex="FALSE"
                    connectionLogFileName=""
                    extraInformation="FALSE"

                    connNumber="-1"
                    useSSL="FALSE"
                    useSSLClientMode="FALSE"
                    keyStoreFileName="classpath:engks/andrea.ks"
                    keyStorePassword="andrea"
                    trustStoreFileName="classpath:engks/andrea.ks"
                    trustStorePassword="andrea"

                    outgoingWrapper="WRAPPER_USER"
                    outgoingHeader="0x0B"
                    outgoingTrailer="0x1C 0x0D"
                    pointName="null"
                    pointType="POINT_TYPE_CLIENT"
                    pointMode="OPERATION_MODE_BIDIRECTIONAL"
                    responseTimeout="10000"
                    retryCount="1"
                    failAction="FAIL_ACTION_CLOSE_CONN"
                    retryNumber="0"
                    retryType="RETRY_TYPE_IMMEDIATE"
                    retryDelay="0"
                    outNmEnvelope="tcp-message"
                    inNmEnvelope="tcp-message"
                    base64encode="TRUE"
                    base64decode="TRUE"
                    defaultMep="http://www.w3.org/2004/08/WSDL/in-out"
                    tcpOutInReceiverClassName="org.spagic.smx.components.tcp.ConsoleTCPReceiver">
            </tcp:tcpBCConfig>
```


```
        </tcp:config>

</tcp:consumer>
```

2.1.3 JDBC Poller

JDBC Poller	
Component family	Standard, Binding Component
Service	StartService
Service Binding	JBI-JDBCPoller
Function	The JDBC poller allows information to be accessed from a database.

2.1.3.1 Database Configuration

The configuration must include the four standard JDBC configuration properties:

Database Configuration properties	
Connection URL	Fully qualified name of a JDBC driver class
Connection Driver	JDBC URL of the database to connect to
Connection Username	The username for database access
Connection Password	The password for database access
Database type	The type of database with which this poller will be connecting. Database types: • Microsoft SQL Server • MySQL • Oracle • Sybase • DB2 • Postgre_SQL • Microsoft Access • JDBC, for other database types

2.1.3.2 Statements Configuration

The configuration include the main statement and the childs SQL elements:

Statements	
SQL	This is the root SQL statement. The statement element contains one SQL element that will be executed and a number of other statement elements that will each be executed for every row returned by the SQL statement.
SubQuery Sql	The SQL element contains a SQL statement to execute.

	This statement can contain @columnName, @messageFieldName or $propertyName variables.

2.1.3.3 XML Generation

The JDBC binding component essentially watches one main table. Every time new rows are inserted into this table, the JDBC binding component will retrieve these rows and return the information contained in them as an XML message.
Retrieval of these rows is the responsibility of the root SQL statement. The user must define a “key” column for the main table, which is used by the root statement to order and restrict the rows returned from the main table.

XML generation	
Period in millisecond	How often to run the SQL statements that generate the messages.
Key	The name of the “key” column. It should be returned from the root statement, and the value of which is used to update the value of stored key. For further explanation refer to next section.
Initial Key Value	The initial value for the stored key. This value is only used once to initialize the key.
Comm Point ID	A unique ID for this JDBC binding component (unique from other JDBC binding components). It is used to uniquely identify the key that is stored inside Spagic.
Rows for each message (leave blank for all)	The number of rows to be used to generate an outgoing message out of the root statement result set. Typically, the value 1 is used to generate an XML message per each row. In this case, the tranction of the root SELECT SQL statement is executed when all the rows previously returned are processed.
Row Name as Attribute	If this property exists and the name property is defined for a statement element then the output messages will have that name as an attribute instead of as an element name.
Column Name as Attribute	If this property exists then the output messages will have the column name as an attribute instead of as an element name.
Value as Attribute	If this property exists then the output messages will have the column result value as an attribute instead of as a CDATA section.

The root SQL statement must always have the following general form:

```
SELECT col_1, col_2, …
FROM the_main_table
WHERE some_other_conditions
AND key_column > ?
ORDER BY key_column ASC
```

Or

```
SELECT col_1, col_2, …
FROM the_main_table
WHERE some_other_conditions
AND key_column < ?
ORDER BY key_column DESC
```

The set of columns returned (col_1, col_2, …) must contain the “key” column so that it can be stored and used for the next query. The current value of the key column after each successful query is stored in Spagic's own database using the given “Comm Point ID” as a unique id to store it with. This means the value is saved and if Spagic is restarted, the JDBC binding component will pick up where it left off.
The first time the JDBC binding component is run, no value will exist in the Spagic database for the key value. In this case the initialKeyValue property must be used to set an initial value for the property.

If a value exists in the Spagic database for the key, the InitialKeyValue property is ignored. The SQL for the root statement must return a column with the name of the key column and should have appropriate where and order by clauses referring to the key column to prevent the same row being returned more than once.

2.1.3.4 Configuration Example

The following listing is a sample XML configuration file for a JDBC poller.

This configuration is generated by Spagic Studio when you configure the component, but can be useful as a configuration sample.

```
<jdbc:poller service="foo:myJdbcBC 1" endpoint="myJdbcBC 1"
  targetService="foo:myScreenOutputsimpleJDBC"
  databaseType="MySQL"
  driver="com.mysql.jdbc.Driver"
  URL="jdbc:mysql://athos:3306/smx"
  userName="smx"
  password="smx"
  period="60000"
  key="id_attribute"
  initialKeyValue="0"
  rowsInMessage="1"
  commPointID="attribute"
  rowNameAsAttribute="true"
  columnNameAsAttribute="true"
  valueAsAttribute="true">
   <rootStatement>
      <bean class="org.spagic.components.jdbc.StatementElement">
         <property name="name" value="main" />
         <property name="sql" value="SELECT id_attribute,id_catalog,name
          FROM `attribute` where id_attribute &gt; ? order by id_attribute asc" />
         <property name="subQueries">
            <list>
               <bean class=" org.spagic.smx.components.jdbc.StatementElement">
                  <property name="name" value="sub1" />
                  <property name="sql" value="SELECT id_relevant_data,value
                  FROM `relevant_data` where id_attribute = @id_attribute" />
               </bean>
               <bean class=" org.spagic.smx.components.jdbc.StatementElement">
                  <property name="name" value="sub2" />
                  <property name="sql" value="SELECT id_rule,expr
                  FROM `rule` where id_attribute &lt; @id_attribute" />
               </bean>
            </list>
         </property>
      </bean>
   </rootStatement>
</jdbc:poller>
```

2.1.3.5 XML Message Format

The general form of the XML message is as follows:

```
<message>
<rowname1>
<columnName1>column value</columnName1>
<columnName2>column value</columnName2>
<columnName3>column value</columnName3>
…
</rowname1>
<rowname2>
<columnName1>column value</columnName1>
<columnName2>column value</columnName2>
<columnName3>column value</columnName3>
…
</rowname2>
<rowname3>
<columnName1>column value</columnName1>
<columnName2>column value</columnName2>
<columnName3column value</columnName3>
…
</rowname3>
</message>
```

Essentially for each statement, starting at the root statement, you have...

```
<rowName>
<columnName>column value</columnName>
<columnName>column value</columnName>
<columnName>column value</columnName>
</rowName>
```

...groups of elements for each row returned by the SQL statement from the database. The “row” group contains exactly one “column” element for each column in the row returned and then contains “row” groups for every child statement.

The following listing is a sample XML configuration for the root statement and some child statements.

```
<rootStatement>
   <bean class=" org.spagic.smx.components.jdbc.StatementElement">
      <property name="name" value="main" />
      <property name="sql" value="SELECT id, date FROM main_table
            WHERE date > ? ORDER BY date ASC" />
      <property name="subQueries">
         <list>
            <bean class="org.spagic.smx.components.jdbc.StatementElement">
               <property name="name" value="names" />
               <property name="sql" value="SELECT firstName, secondName
                     FROM names_table WHERE id = @id" />
            </bean>
            <bean class="org.spagic.smx.components.jdbc.StatementElement">
               <property name="name" value="address" />
               <property name="sql" value="SELECT address, country_code
                     FROM address_table WHERE id = @id" />
            </bean>
         </list>
      </property>
   </bean>
</rootStatement>
```

Assume the database tables used in this example have the data from the tables below.

id	date
97320	2002-1-1 12:12:34
23409	2002-1-1 12:14:56
20234	2002-1-2 10:32:25

Data for main_table

id	firstName	secondName
97320	bob	brown
23409	doug	green
20234	mary	johns

Data for names_table

id	address	country_code
97320	12 Nowhere St	US
23409	43 Higher Ave	NZ
20234	123 Long Rd	UK
20234	321 Short St	US

Data for address_table

Then the XML produced would be:

```
<message>
<main>
<id>97320</id>
<date>2002-1-1 12:12:34</date>
<names>
<firstName>bob</firstName>
<secondName>brown</secondName>
</names>
```

```
<address>
<address>12 Nowhere St</address>
<country_code>US</country_code>
</address>
</main>
<main>
<id>23409</id>
<date>2002-1-1 12:14:56</date>
<names>
<firstName>doug</firstName>
<secondName>green</secondName>
</names>
<address>
<address>43 Higher Ave</address>
<country_code>NZ</country_code>
</address>
</main>
<main>
<id>20234</id>
<date>2002-1-2 10:32:25</date>
<names>
<firstName>mary</firstName>
<secondName>johns</secondName>
</names>
<address>
<address>123 Long Rd</address>
<country_code>UK<country_code>
</address>
<address>
<address>321 Short St</address>
<country_code>US</country_code>
</address>
</main>
</message>
```

The output XML can be changed by using the *columnNameAsAttribute*, *rowNameAsAttribute* and *valueAsAttribute* config properties. If *columnNameAsAttribute* is specified, the columnName, instead of being used as the element name in the XML, is included as the value of a “name” attribute and the element is called “column” instead.

Similarly the row name can be included as an attribute instead of as an element name. If *valueAsAttribute* is specified, the column result, instead of being the element CDATA in the XML, is included as the value of a “value” attribute.

So if both *columnNameAsAttribute*, *rowNameAsAttribute* and *valueAsAttribute* exist in the config file, the above XML would become:

```
<message>
<row name="main">
<column name="id" value="97320"/>
<column name="date" value="2002-1-1 12:12:34"/>
<row name="names">
<column name="firstName" value="bob"/>
<column name="secondName" value="brown"/>
</row>
<row name="address">
<column name="address" value="12 Nowhere St"/>
<column name="country_code" value="US"/>
</row>
</row>
<row name="main">
<column name="id" value="23409"/>
…
</message>
```

2.1.4 StreamWriter (Screen)

Screen	
Component family	Lightweight, Binding Component
Service	EndService

Service Bindings	JBI-ConsoleOutput
Function	This component can be used only as an output binding component to end an integration process with a log in standard out. It's useful for debugging purpose or for integration processes that doesn't need to end to a particular channel adapter.

This component doesn't' have configuration parameters.

2.1.4.1 Configuration Example

Screen (StreamWriter component) is a ServiceMix lightweight component so the configuration will be produced in lw service unit, in the ServiceMix.xml file, the fragment below is a sample of xml produced:

```
<!-- ######################## crmScreenOutput  ######################## -->
    <sm:activationSpec componentName="crmScreenOutput " service="foo:crmScreenOutput ">
         <sm:component>
          <bean  xmlns="http://xbean.org/schemas/spring/1.0"
                      class="org.apache.ServiceMix.components.util.StreamWriterComponent">
       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.1.5 File

File	
Component family	Standard, Binding Component
Service	StartService/EndService
Service Bindings	• JBI-FileInputBindingComponent • JBI-FileOutputBindingComponent
Function	This component can be used as input binding components or as output binding components. When used as input it acts as a poller on a particular directory; Each file found in that folder will began a process instance. When used as output binding component the messages of last service engine in flow will produce files in a specified path.

2.1.5.1 Configuration Parameters when used as Input

Poller	
File(*)	The name of the *directory* to poll.
Delete polled file	If TRUE the file will be deleted after read.
Polling period (ms)	Amount of time between polls.
Select the File Management	Configure the marshaller that will manage the content of the file; possible choices: ❑ Default File Marshaller: will put the content of the

	file as the payload of the normalized message. ❑ Binary File Marshaller: will put the file as an attachment of the normalized message.

In future releases of Spagic Studio we'll support other parameters as *period, recursive, and filters* on name files. At the moment this parameter are configured with default values.

2.1.5.1.1 Configuration Example

```
<?xml version="1.0"?>

<beans xmlns:sm="http://ServiceMix.apache.org/config/1.0"
               xmlns:file="http://ServiceMix.apache.org/file/1.0"
               xmlns:foo="http://ServiceMix.org/cheese"
               >
<file:poller service="foo:fileStore"
     endpoint="fileStore"
     targetService="foo:store"
     file="file:/temp/file"
     deleteFile="true">
</file:poller>
</beans>
```

2.1.5.2 Parameters Configuration for Output File

Sender	
Directory(*)	The name of the *directory* where the files will be written
Append	To write the file in append mode. Valid for file .txt
Select the File Management	Configure the marshaller that will manage the content of the file; possible choices: ❑ Default File Marshaller: will put the content of the file as the payload of the normalized message. ❑ Binary File Marshaller: will put the file as an attachment of the normalized message.
File content type	Set the content type of the file attached with the Binary file Marshaller. Default "text/plain"

In future releases of Spagic Studio we'll support other parameters for example marshallers.

2.1.5.2.1 Configuration Example

```
<beans xmlns:sm="http://ServiceMix.apache.org/config/1.0"
               xmlns:file="http://ServiceMix.apache.org/file/1.0"
               xmlns:foo="http://ServiceMix.org/cheese"
               >
<file:sender service="foo:fileOrder"
     endpoint="fileOrder"
     directory="file:/temp/file">
</file:sender>
</beans>
```

2.1.6 Mail

Mail	
Component family	Lightweight, Binding Component
Service	EndService
Service Bindings	JBI-SendMailOutBindingComponent
Function	This component is used as output binding component to send an email message as the end of an integration process.

2.1.6.1 Configuration Parameters

Hostname(*)	The mail server host name or ip address
Communication Port(*)	The port used to send email (usually 25 smtp)
Mail To(*)	The address destination where to send mail
Mail From	Optional, if you want to set sender in the email that will be send
Subject	Optional, if you want to set the subject of the email that will be sent
Body	Optional, If you leave blank the message content will be the body of the email sent.

2.1.6.2 Configuration Example

```
<!-- ######################## myMailServerProcess1 ######################## -->
 <sm:activationSpec componentName="myMailServerProcess1" service="foo:myMailServerProcess1">
      <sm:component>
       <bean class="org.apache.ServiceMix.components.email.MimeMailSender">
           <property name="marshaler">
             <bean class="org.apache.ServiceMix.components.email.MimeMailMarshaler">
                  <property name="from">
                   <bean class="org.apache.ServiceMix.expression.ConstantExpression">
                         <constructor-arg value="SMX" />
                   </bean>
                  </property>
                  <property name="to">
                      <bean class="org.apache.ServiceMix.expression.ConstantExpression">
                      <constructor-arg value="zoppello@eng.it" />
                      </bean>
                  </property>
                   <property name="subject">
                      <bean class="org.apache.ServiceMix.expression.ConstantExpression">
                         <constructor-arg value="Automatic mail from SMX" />
                      </bean>
                  </property>
```

```
                </bean>
                </property>
                <property name="sender">
                    <bean class="org.springframework.mail.javamail.JavaMailSenderImpl">
                            <property name="host"  value="mail.eng.it" />
                            <property name="port"  value="25" />
                    </bean>
                 </property>
            </bean>
    </sm:component>
</sm:activationSpec>
```

2.1.7 JMS Binding Component

JMS Binding Component	
Component family	Standard, Binding Component
Service	StartService/EndService
Service Binding	• JBI-JMSInputBindingComponent • JBI-JMSOutputBindingComponent
Function	A Standard binding component that handle input or output of the flows from and to JMS queues.

2.1.8 Quartz Timer Component

Timer (Quartz) Binding Component	
Component family	Standard, Service Engine
Service	StartService
Service Binings	JBI-QuartzTimer
Function	A Standars service engine to send messages at predefined interval of time. It use quartz concept like cron and triggers

See ServiceMix documentation. In future release of the documentation we'll explain this component configuration better.

2.2 Service Engines

2.2.1 JDBC Query Component (Simple)

JDBC Query Component	
Component family	Lightweight, Service Engine

Service	Task Service
Service Binings	JBI-JDBCSimpleService
Function	This component expect a message with content: <sql> <!--sql query-> </sql> the query will be executed and return an xml in the format of : <ResultSet> <rows> <row colName=”value” colName2=”value2” …> </rows> </ ResultSet >

This component doesn’t have configuration parameters.

2.2.1.1 Configuration Example

```
<!– This is the Spring configuration of a datasource defined in SERVICE_MIX_HOME/conf/jndi.xml ->
<bean id="mySql" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="java:comp/env/jdbc/servicemixDB"/>
</bean>
…….

<!–Note that JDBC Endpoint declare a property ref to the dataSource defined by the bean above ->
<!-- ######################## myJDBCdbAndrea ######################### -->
    <sm:activationSpec componentName="myJDBCdbAndrea" service="foo:myJDBCdbAndrea"
          destinationService="foo:myScreenOutputdbAndrea">
          <sm:component>
           <bean class="org.apache.ServiceMix.components.jdbc.JdbcComponent">
            <property name="dataSource" ref="mySql"/>       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.2 JDBC Advanced Query & Stored Procedure Component

JDBC Advanced Query Component	
Component family	Lightweight, Service Engine
Service	Task Service
Service Bindings	JBI-JDBCAdvancedService
Function	Component able to use information from incoming NM to execute querys or stored procedures throught JDBC.

2.2.2.1 Configuration Properties

Use configured datasources	If TRUE the connection will be obtained using datasources, otherwise all configuration parameters can be passed.

DataSource	Name of the datasource
Jdbc driver class	Name of the JDBCdriver class
Jdbc connection url	Connection url in JDBC format.
Database user name	Name of the user that connect to the db.
Database user password	Password for the connecting user.
Database vendor	Name of the database vendor used for some specific configuration.
Is stored procedure call	If TRUE the component has to execute a stored procedure.
Query	Query or stored procedure call to execute (depend on "Is stored procedure call" property).
Enrich input message	If TRUE the component will try to enrich the input message instead of create a new one
Xml envelop	Envelope in the incoming message where to put execution output
Rows Xml envelop	Name of the output envelope collecting all row results.
Row Xml envelop	Name of the output envelope use to collect data of a single row
Fault management	Type of fault management, options: ❑ FAULT_JBI: a fault will be managed by the container than the process will have an error status. ❑ FAULT_FLOW: the fault will be enveloped in the output message giving the other component in the flow the possibility to manage the error.

Based on the "Query" parameter will be presented a list of parameter configurations. These configurations change if the component has to execute a query or a stored procedure call.

2.2.2.2 Query Parameter configuration

Parameter type	Type of the parameter value
Parameter XPath expression	XPath expression used to retrieve parameter value from incoming NM.

2.2.2.3 Stored Procedure parameter configuration

Is output parameter	Says if the parameter is an output one or not.
Parameter type	Type of the parameter value.
Parameter XPath expression	XPath expression used to retrieve parameter value from incoming NM. Present only if the parameter is an input one.

2.2.2.4 Configuration Example

2.2.2.4.1 Query example

```
<!-- ######################## myJdbcQueryadvJdbcDatasource2 ######################## -->
<sm:activationSpec componentName="myJdbcQueryadvJdbcDatasource2"
        service="foo:myJdbcQueryadvJdbcDatasource2"
        destinationService="foo:myScreenOutputadvJdbcDatasource2">
        sm:component>
```

```
        <bean class="org.spagic.smx.components.jdbcquery.JDBCAdvancedQueryComponent">
        <property name="connConfig">
                <bean class="org.spagic.smx.components.jdbcquery.JDBCConnectionConfig">
                      <property name="datasource" ref="jforum"/>
                 </bean>
                </property>
                <property name="queryConfig">
                <bean class="org.spagic.smx.components.jdbcquery.JDBCQueryConfig">
<property name="query"  value="SELECT * FROM jforum_users where username like $name and user_password
like $password" />
                                        <property name="enrichMessage"  value="TRUE" />

                                        <property name="xmlEnvelope"  value="Customer" />

                                        <property name="queryParams">
                                        <list>
                                        <bean
class="org.spagic.smx.components.jdbcquery.QueryParameterConfig" >
                                        <property name="placeHolder" value="name" />
                                        <property name="outputParam" value="FALSE" />
                                        <property name="XPath" value="/jforum/userletter" />
                                        <property name="paramType" value="java.lang.String" />
                                        </bean>
                                        <bean
class="org.spagic.smx.components.jdbcquery.QueryParameterConfig" >
                                        <property name="placeHolder" value="password" />
                                        <property name="outputParam" value="FALSE" />
                                        <property name="XPath" value="/jforum/userpassword" />
                                        <property name="paramType" value="java.lang.String" />
                                        </bean>
                                        </list>
                                        </property>
                                        </bean>
                                        </property>
        </bean>
       </sm:component>
    </sm:activationSpec>
```

2.2.2.4.2 Stored procedure example:

```
<!-- ###################### myJdbcQuery 1 ######################## -->
<sm:activationSpec componentName="myJdbcQuery 1" service="foo:myJdbcQuery 1"
        destinationService="foo:myScreenOutputStoreProcedureTest">
                                    <sm:component>
                                    <bean
class="org.spagic.smx.components.jdbcstore.JDBCStoreProcedureComponent">
                                    <property name="connConfig">
            <bean class="org.spagic.smx.components.jdbcquery.JDBCConnectionConfig">
        <property name="databaseVendor"  value="Oracle" />
        <property name="datasource" ref="engiprj"/>
        </bean>
                                    </property>
                                    <property name="queryConfig">
                                    <bean class="org.spagic.smx.components.jdbcquery.JDBCQueryConfig">
```


```
                                    <property name="query"  value="{call
pkg_prova.get_resultset($rSet)}" />                <property name="enrichMessage"  value="FALSE" />
                                    <property name="xmlEnvelope"  value="store-results" />
                                    <property name="queryParams">
                                    <list>
                                    <bean
class="org.spagic.smx.components.jdbcquery.QueryParameterConfig" >
               <property name="placeHolder" value="rSet" />
               <property name="outputParam" value="TRUE" />
               <property name="XPath" value="null" />
               <property name="paramType" value="-10" />
               </bean>
                                    list>
                                    </property>
                                    </bean>
                                  </property>
       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.3 Syntax Validator

Syntax Validator Component	
Component family	Lightweight, Service Engine
Service	Task Service
Service Binding	JBI-XSD-SyntaxValidator
Function	This component is used to validate the message content againt xsd file

2.2.3.1 Configuration Parameters

Schema Resource(*)	The path of the xsd file used to validate the message content. The xsd**, must be** deployed in servicemix, under the folder **SMX_HOME/resources/xsd** That we'll call ***XSD_FOLDER*** The format is: ***<path_to_xsd_relative_to_SMX_XSD_FOLDER>*** For example if under the ***SMX_XSD_FOLDER*** the xsd file sample.xsd under validation folder the path will be: ***validation/sample.xsd*** *The xsd file **will not be included** when the service assembly will be generated, but must be deployed before on servicemix before the service assembly deploy.*

Error Handling(*)	With this parameter it's possible to choose if handle syntax validation errors as 1. JBI Fault. In that case the error will be handled by by listener and restart mechanism provided with Spagic. 2. Generate a "**correct JBI message" with a "fault content"** that will be routed to the next component that can inspect the message content and handle the fault within the flow.

2.2.3.2 Configuration Example

```
<bean id="messageAggregatingErrorHandlerFactory"
        class="org.apache.ServiceMix.components.validation.MessageAggregatingErrorHandlerFactory">
       <property name="rootPath" value="Fault/messages"/>
       <property name="includeStackTraces" value="false"/>
</bean>
<sm:serviceunit id="JBI">
       <sm:activationSpecs>

    <!-- ######################## mySyntaxValidatorTestSyntaxVal ######################## -->
    <sm:activationSpec componentName="mySyntaxValidatorTestSyntaxVal"
          service="foo:mySyntaxValidatorTestSyntaxVal"
         destinationService="foo:myRouterTestSyntaxVal">
         <sm:component>
           <bean class="org.apache.ServiceMix.components.validation.ValidateComponent">
              <property name="schemaResource"  value="classpath:xsd/ContattiRichiestaServiziIn.xsd" />
              <property name="handlingErrorMethod"  value="APP" />
              <property name="errorHandlerFactory" ref="messageAggregatingErrorHandlerFactory"/>
           </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.3.3 Notes on XSD

Unlike other resources xsd files used by Syntax Validator component will not been included in service assembly but we must deploy it manually on separate on classpath. Schema files are accessed as spring classpath resource so the preferred way to work with xsd is to have a fixed classpath folder in servicemix located at **SMX_HOME/resources/xsd**, where to put xsd files.

1. We could have different subfolders under **SMX_HOME/resources/xsd.** For example all xsd about hl7 validations could be located under **SMX_HOME/resources/xsd/hl7** and so on
2. In the Syntax Validator endpoints set in the schema resources properties the relative path of xsd file from **SMX_HOME/resources/xsd,** with the following syntax:

   <subfolder_name>/<xsd_file_name>.xsd

If you've xsd that need to import other XSD make sure to use only relative imports. (In the xsd file should not appear absolute path). The following image show a correct import declaration:

```
1
2<?xml version="1.0" encoding="UTF-8"?>
3<xsd:schema elementFormDefault="qualified"
4        xmlns="http://www.w3.org/2001/XMLSchema"
5        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6        xmlns:tns1="urn:overit:geocall:service:types"
7        targetNamespace="urn:overit:geocall:service">
8
9    <xsd:import namespace="urn:overit:geocall:service:types"
10           schemaLocation="ComunicazioneContattiRichiesta.xsd"/>
11   <xsd:element name="in" type="tns1:ComunicazioneContattiServiziRichiesta"/>
12
13</xsd:schema>
```

2.2.4 Semantic Validator

Semantic Validator Component	
Component family	Standard, Service Engine
Service	TaskService
Service Binding	JBI-DroolsValidator
Function	This component use a rule file expressed in drools syntax where it'spossible to write a set of rules that defines when the semantic validation will fails. If none of this rules will be verified the semantic validation succeed an the process continues

2.2.4.1 Configuration Parameters

Rule Base Resource File Path(*)	The path of the drools file (drl) where the ruleset are defined. The rule base resource path must be in the form: **SemanticRules/<file_name>.drl** This means that the drools file must be in the SemanticRules folder of the Spagic Project that contains the integration process file using this component. **In Spagic Studio 2 you could simply fill this property by do a drag and drop of a ".drl" file from the workbench to the bpmn task activity annotated as a JBI-DroolsValidator** *This file will be included when the service assembly will be generated.*

2.2.4.2 Configuration Example

```
<beans xmlns:sm="http://ServiceMix.apache.org/config/1.0"
```

```
xmlns:drools="http://ServiceMix.apache.org/drools/1.0"
xmlns:foo="http://ServiceMix.org/cheese">
       <drools:endpoint service="foo:mySemValidatorTestDroolsValidator"
              endpoint="mySemValidatorTestDroolsValidator"
                     ruleBaseResource="classpath:rulesetcondb.drl"
                     namespaceContext="#nsContext"
                     defaultTargetService="foo:myRouterTestDroolsValidator"/>
```

2.2.4.3 Writing rules file

Rules are expressed in drools syntax. Basically a rule has the general form:

```
Rule <RuleName>
when < precondition >
then < action >
```

In the semantic validator rulebase resource ***we must write rules that if verified will cause a validation error***.
To help to write rules, the rule engine is populated with a set of objects that help us:

- ***Asserted Objects.*** This object populates the working memory and they can be used in the precondition part of the rule. The asserted object automatically available are:
  - ***me:***It's basically a wrapper on message exchange where we can for example evaluate XPath expression By using this object you can access **in message** and evaluate a boolean XPath expression on it.
  - ***db:***It's an object that expose a very easy api to express in the rule precondition some constraint against a database.

```
public boolean exist(String value, String valueType, String tableName, String
fieldName,String dsName)

public boolean existOne(String value, String valueType, String tableName, String
fieldName,String dsName)
```

  - The first method execute the following task:
    - Get a connection from the datasource defined in jndi as `java:comp/env/jdbc/ <dsName>` where `dsname` is the last parameter passed
    - Execute the sql query:
      `"SELECT T."+fieldName+" FROM "+tableName+" T WHERE T."+fieldName+" = ? ";`
      where the parameter value will be the `value` parameter
    - Return true if this query return one or more row.
  - The second method execute the same tasks as the first the only difference is that it returns true if and only if the number of rows returned is one.

- ***Helper Objects.*** These objects help us to write the "action" part of the rule. The helpers object automatically available are:
  - ***JBI:*** It's the object used to generate an application or JBI fault.
    Method exposed are:

```
public void fault(String content) throws Exception

public void faultToDefaultTarget(String faultInFlowContent) throws MessagingException
```

  - The first method is used when we want the semantic validator will fails with a JBI Fault and the error will be handled by listener and restart mechanism provided with Spagic
  - The second method is for project that needs to handle the error in flow. In that case a "**correct JBI message" with a "fault content"** will be generated and be routed to the next component that can inspect the message content and handle the fault within the flow.
    A Spagic convention is to generate this type of messages in the following form:

```
<Fault>
        <!-- Fault content can be other xml
         The mportant thing is that root elemet is called Fault
         so we can handle with a XPath Router
        -->
</Fault>
```

  - So you can used in method for example as:

```
JBI.faultToDefaultTarget( "<Fault> ACTION STATUS IS NOT VALID </Fault>" );
```

2.2.4.4 Example of rulebase resource

```
package org.apache.ServiceMix.drools

import org.apache.ServiceMix.drools.model.Exchange;
import org.apache.ServiceMix.drools.model.DbHelper;

global org.apache.ServiceMix.drools.model.JbiHelper JBI;

        rule "Rule1"
        when
                me : Exchange( status == Exchange.ACTIVE, in : in != null )
                eval( in.XPath("/ACTION/@status = 'NOT_VALID'") )
        then
                JBI.faultToDefaultTarget( "<Fault> ACTION STATUS IS NOT VALID </Fault>" );
        end

        rule "Rule1"
        when
            me : Exchange( status == Exchange.ACTIVE, in : in != null );
            db : DbHelper ( );
            eval( db.exist( in.valueOf("/ACTION/@nome"), "STRING", "attributes", "name", "metadb" ) );
        then
                JBI.fault( "<ERROR> The value is already present in db </ERROR>" );
        end


        rule "Rule3"
        when
                me : Exchange( status == Exchange.ACTIVE, in : in != null )
                eval( in.XPath("/ACTION/@status = 'NOT_VALID'") )
```

```
then
        JBI.fault("<ERROR> The status is not valid </ERROR>");
end
```

2.2.5 Groovy Scripting Component

Groovy Scripting Component	
Component family	Lightweight, Service Engine
Service	TaskService
Service Bindings	JBI-GroovyScriptingComponent
Function	This component enables the use of scripting code within an Integration Process. Scripts are expressed in groovy language.

2.2.5.1 Configuration Parameters

Groovy File Path(*)	The path of the groovy file (.groovy) that contains the script code. The groovy file path must be expressed as: **Scripts/<file_name>.groovy** This means that groovy file must be in the Scripts folder of the Spagic Project that contains the integration process file using this component. **In Spagic Studio 2 you could simply fill this property by do a drag and drop of a ".groovy" file from the workbench to the bpmn task activity annotated as a JBI-GroovyScriptingComponet.** *This file will be included when the service assembly will be generated.*

2.2.5.2 Configuration Example

```
<!-- ######################## myGenericGroovymioGroovy ######################## -->
 <sm:activationSpec componentName="myGenericGroovymioGroovy" service="foo:myGenericGroovymioGroovy"
      destinationService="foo:myScreenOutputmioGroovy">
      <sm:component>
       <bean class="org.apache.ServiceMix.components.groovy.GroovyComponent">
        <property name="script"  value="classpath:testsmx.groovy" />
    </bean>
  </sm:component>
 </sm:activationSpec>
```

2.2.5.3 Notes on Groovy

In this document we don't cover Groovy Scripting Language; you can find very good documentation here: http://groovy.codehaus.org/.

While inside the script you have access to the some useful objects that you can use inside groovy script.

Variable	Description
inMessage	***The in message***
outMessage	***The out message***
context	The JBI ComponentContext
deliveryChannel	The DeliveryChannel
exchange	The JBI MessageExchange
bindings	A Map which is maintained across invocations for the component which allows you to share state across requests. This state is not persistent, but will last for the duration of the JVM

Most important are the inMessage and outMessage objects, that give the ability to manipulate the messages contents and headers.

2.2.5.4 Example of groovy script

2.2.6 Talend Job Caller

Talend Job Caller	
Component family	Lightweight, Service Engine
Service	StartService
Service Bindings	JBI-TalendScript
Function	This component enables to call a job designed with Talend Open Studio, to be called in an integration process. This is needed where process need to do database intensive job for whch the already available jdbc components are not enough.

2.2.6.1 Configuration Parameters

Talend Job Full Class Name(*)	The complete name of the class implementing the talend job. This class must be in ServiceMix classptah. In Spagic we

	provide a folder usually: ***<SERVICE_MIX_HOME>/lib/talend*** where we put all jars needed by talend jobs.

2.2.6.2 Configuration Example

```
<!-- ###################### myTalendTalendTest ###################### -->
    <sm:activationSpec componentName="myTalendTalendTest" service="foo:myTalendTalendTest"
          destinationService="foo:myScreenOutputTalendTest">
          <sm:component>
           <bean class="org.spagic.smx.components.talend.TalendCaller">
            <property name="talendJobClassName"  value="corso_talend.talendprova.TalendProva" />
       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.6.3 Current Limitations

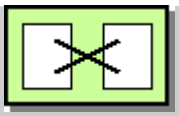
At the moment, from Spagic it's not possible to pass parameters to Talend Job classes.

In Talend v2.00 it's possible to pass parameters to job using context files.

Instead we need:

- To pass parameters dynamically using getters and setters methods.
- To get a list of parameters that a job needs.
- To configure endpoints with a set of XPath rules to popultate talend job parameters with values extracted from message content.

2.2.7 Transformer (XSLT Mapper) Component

Transformer XSLT Mapper Component	
Component family	Lightweight, Service Engine
Service	TaskService
ServiceBinding	JBI-XSLT-TransformerComponent
Function	The component main feature is to transform the message content using an xslt stylesheet. If you remember all message content inside the ESB must have xml content, so this component is important.

2.2.7.1 Configuration Parameters

XSLT File(*)	The path of the xslt file (.xslt) used for the transformation. The xslt file path must be expressed as: **Mappings/<file_name>.xsl(t)**

	This means that xslt file must be placed in the Mapping folder of the Spagic Project that contains the integration process file using this component. **In Spagic Studio 2 you could simply fill this property by do a drag and drop of a ".xsl(t)" file from the workbench to the bpmn task activity annotated as a JBI-XSLT-TransformerComponent.** *This file will be included when the service assembly will be generated.*

2.2.7.2 Configuration Example

```
<!-- ###################### myTransformerInOut ####################### -->
    <sm:activationSpec componentName="myTransformerInOut" service="foo:myTransformerInOut"
         destinationService="foo:myScreenOutputTestCaseHTTPWithMap">
         <sm:component>
          <bean class="org.apache.ServiceMix.components.xslt.XsltComponent">
           <property name="xsltResource"  value="classpath:ActionToPages.xslt" />
       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.7.3 Tools for Developing XSLT Transformation

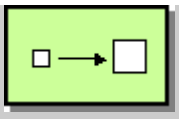
As we've just seen the Transformer xslt mapper component allow us to insert xslt transformation inside the integration process flow. That componet expect an xslt file just ready to use. XSLT is a quite complex language and write manually an xslt file can be quite complex so we need a tools where we can visually design the transformation and the generate the xslt code.

Actually we found two solutions:

- **Altova MapForce.** It's a **commercial solution**, but at the moment is probably the best solution in the market. and if you need only XSLT Transformation the price is quite good for the feature that this tool provide. **http://www.altova.com/products/mapforce/data_mapping.html**
- **Jamper.** It's an alternative **open source solution**. You can find here:**http://jamper.sourceforge.net**

2.2.8 WS Pipeline

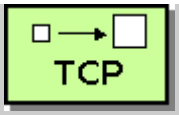
The Pipeline components permit to contact an external Web Service.

WS Pipeline	
Component family	Standard, Service Engine
Service	TaskService
Service Bindings	JBI-WebServicePipeline

Function	Pipeline on WS socket connection

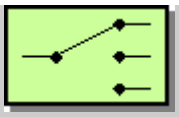
The configuration is similar to the HTTP component configured as provider.

2.2.9 TCP-IP Pipeline

TCP-IP Pipeline	
Component family	Standard, Service Engine
Service	TaskService
Service Bindings	JBI-TCPIP-Pipeline
Function	Pipeline on TCP socket connection

The configuration for this component is similar to the TCP-IP component configured as a client (or provider in JBI terminology).

2.2.10 Router

Router	
Component family	Standard, Service Engine
Service	RouterService
Service Bindings	JBI-Router
Function	A Router is used to route each message to the correct destination based on message content or headers. In particular a router evaluates a set of rules to take decisions.

2.2.10.1 Configuration Parameters

Number of rules(*)	The number of rules to be evaluate
Rule[1..n]	A set of (n) rules where n is the value above. Routing rule can be of two types: • Routing rule based on message content • Routing rule based on message header See the following section.

2.2.10.2 Routing Rules

As we seen at the beginning of this document normalized message are made of message headers, message content or payload and attachments.

Message content and headers can be used to express routing rule. In particular:

- **Rule based on Message Content** are expressed in the form:

```
when < boolean xpath on payload >
then route to <service>
```

- **Rule based on Message Header are** expressed in form:

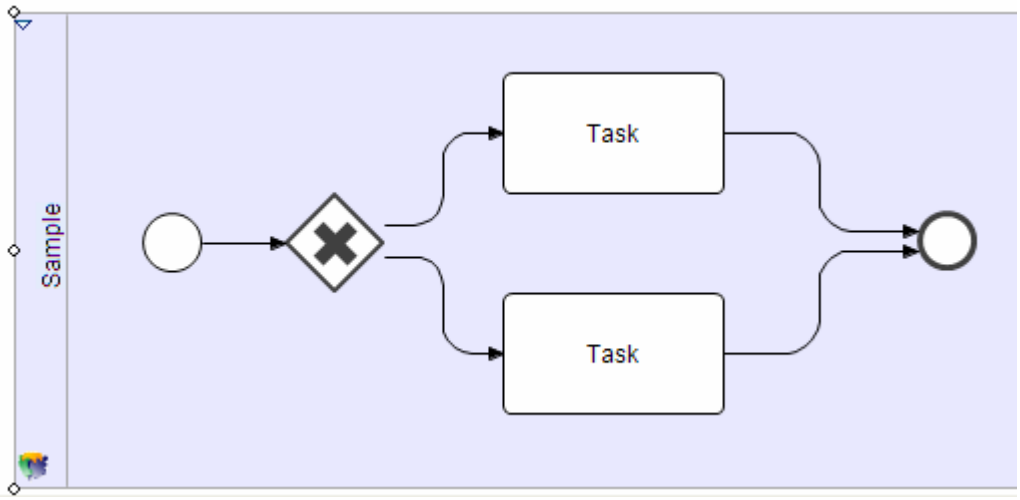
```
when header[<header_name>] <op> <value>
then route to <service>
```

```
<op>
=
equal | not equal | contains | not contain
```

2.2.10.3 Configuration Example

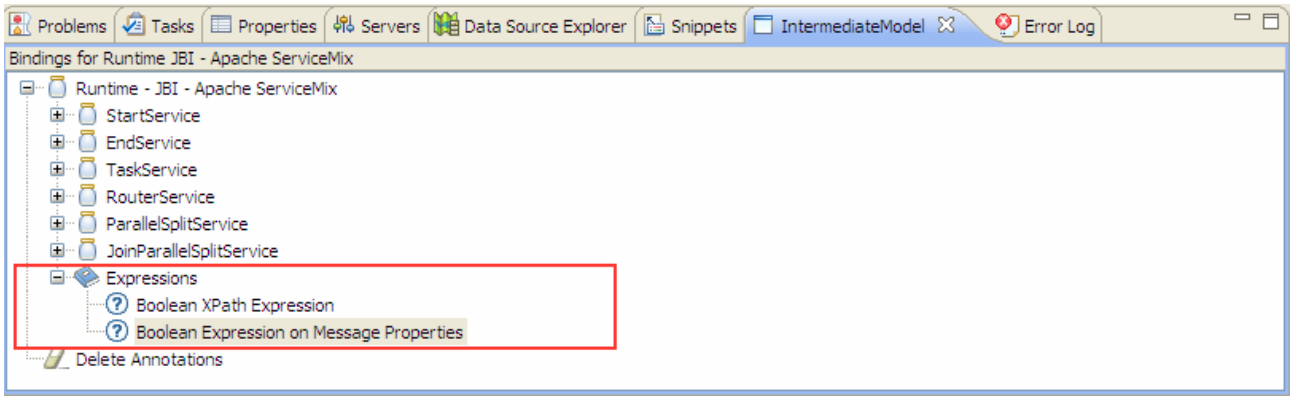
When the SA will be generated routing rules will be expressed in the drools syntax, and in the deployable artefacts a router will be a drools endpoint where the drools file has being automatically and included in the service unit by Spagic Studio.

In BPMN a router is configured using an **ExclusiveDataGateway with a JBI-Router Service Binbing Annotation,** with n exclusive outgoing sequence edge.



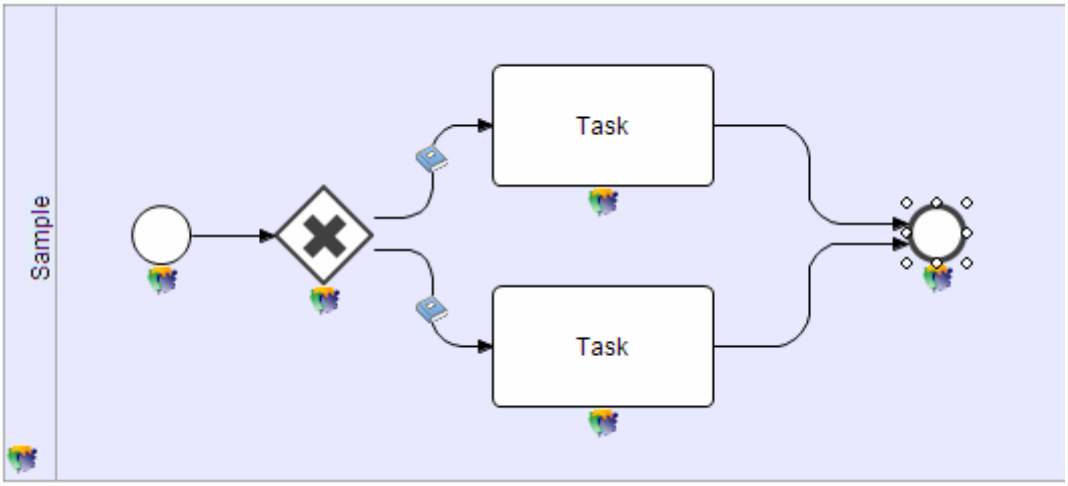
Each ***outgoing sequence edge from the router, represent a rule in corresponding drools file***, so we need a way to express in the sequence edge the relative rule.

To do this the JBI Runtime let's you to annotate each outgoing sequence edge with two type of conditions:

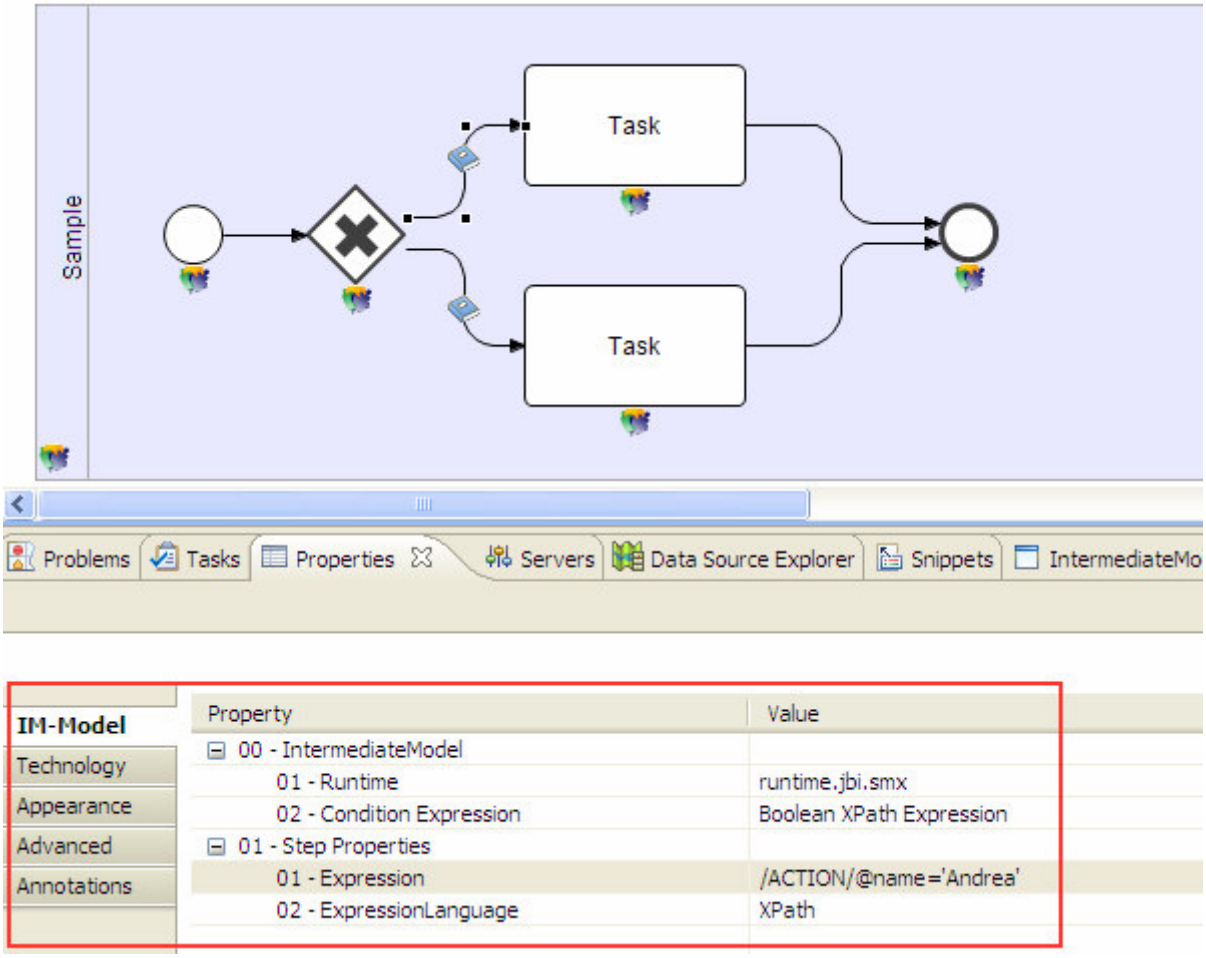


- Boolean XPath Expression
- Boolean Expression on Message Properties

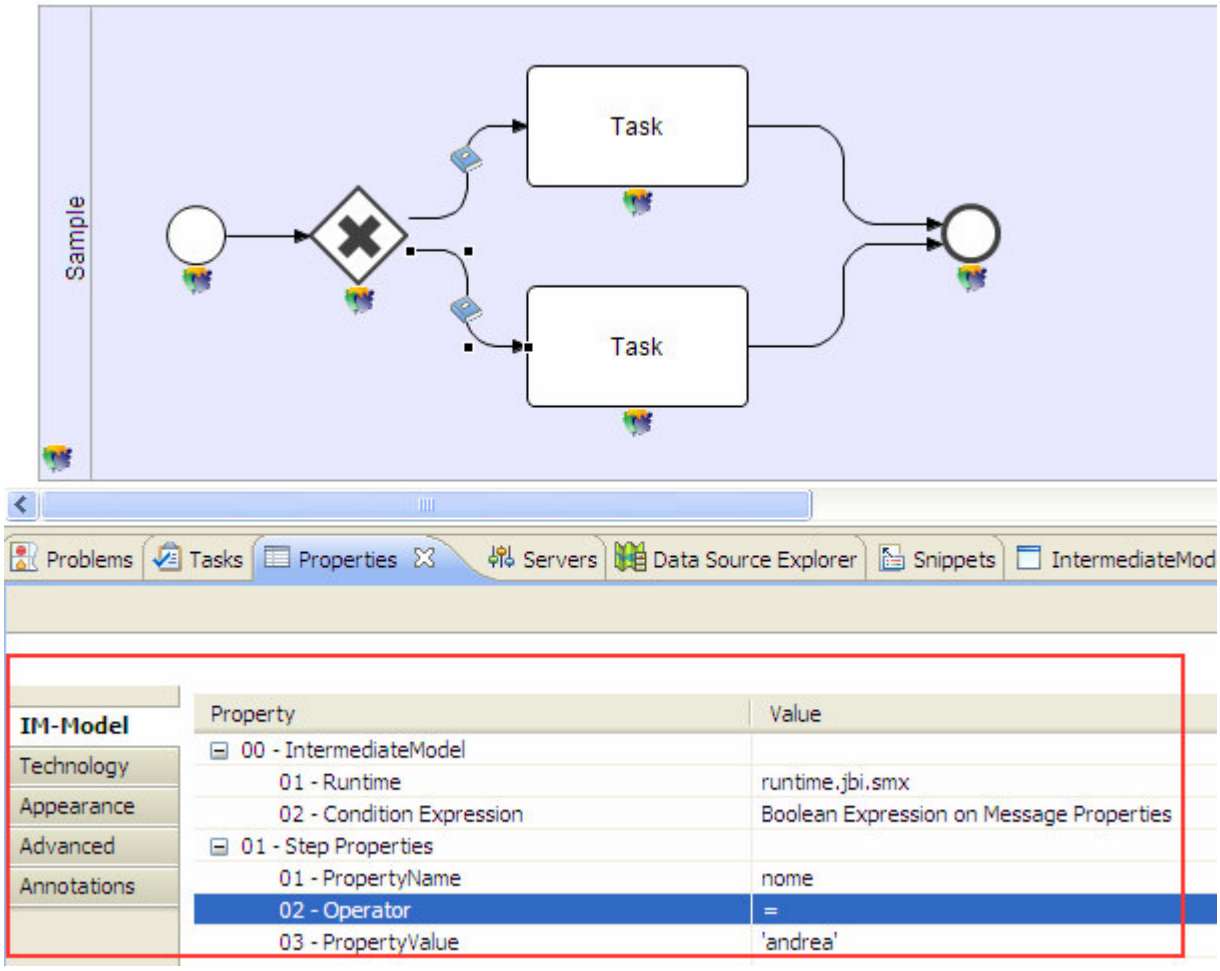
You simply must drag and drop the annotation on the sequence edge as follow:



This will allow you to use the ***Intermediate Model Property Section*** to insert xpath expression:



or the condition on meassge headers:



We have a configuration of a drools endpoint in drools service unit:

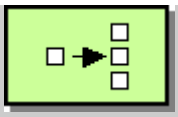
```
<beans xmlns:sm="http://ServiceMix.apache.org/config/1.0"
               xmlns:drools="http://ServiceMix.apache.org/drools/1.0"
               xmlns:foo="http://ServiceMix.org/cheese"
               >

               <drools:endpoint
                        service="foo:router1"
                        endpoint="router1"
                        ruleBaseResource="classpath:router1.drl"
                         namespaceContext="#nsContext"/>
               <drools:namespace-context id="nsContext">
                      <drools:namespaces>
                             <drools:namespace prefix="restart">
                                    urn:it:eng:Spagic:restart
                             </drools:namespace>
                             <drools:namespace prefix="Spagic">
                                    urn:it:eng:Spagic
                             </drools:namespace>
```

```
                    </drools:namespaces>
                </drools:namespace-context>
</beans>
```

The **router1.drl** file has been automatically generated and included in the service unit by Spagic Studio:

```
package org.apache.ServiceMix.drools

import org.apache.ServiceMix.drools.model.Exchange;

global org.apache.ServiceMix.drools.model.JbiHelper JBI;


        rule "RuleNum1"
        when
                me : Exchange( status == Exchange.ACTIVE, in : in != null )

                eval( in.XPath("/DATI/@name='andrea'") )
        then
                JBI.route("service:http://ServiceMix.org/cheese/myGenericGroovyRouter1");
        end

        rule "RuleNum2"
        when
                me : Exchange( status == Exchange.ACTIVE, in : in != null )

                eval( in.XPath("/DATI/@name!='andrea'") )
        then

                JBI.route("service:http://ServiceMix.org/cheese/myGenericGroovyRouter2");
        end
```

2.2.11 XPath Splitter

Splitter (or XPath Splitter)	
Component family	Standard, Service Engine
Service	TaskService
Service Bindings	JBI-XPathSplitter
Function	The best definition can be found here: http://www.enterpriseintegrationpatterns.com A Splitter can be used to break an incoming message into a series of individual smaller messages, each containing data related to one item. Use a Splitter that consumes one message containing a list of repeating elements, each of which can be processed individually. The Splitter publishes a one message for each single element (or a subset of elements) from the original message.

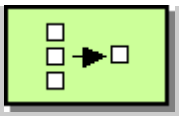
2.2.11.1 Configuration Parameters

XPath Expression(*)	An XPath expression that select a list of nodes from original input message. When the original message is splitted, the list of resulting message will be processed in parallel way by the next component.

2.2.11.2 Configuration Example

```
<beans xmlns:sm="http://ServiceMix.apache.org/config/1.0"
               xmlns:eip="http://ServiceMix.apache.org/eip/1.0"
               xmlns:foo="http://ServiceMix.org/cheese">

               <eip:XPath-splitter
                      service="foo:mySplitterProcess1" endpoint="mySplitterProcess1"
                      XPath="//spago:PAGES/spago:PAGE" namespaceContext="#nsContext">
                      <eip:target>
                             <eip:exchange-target service="drools:myRouterProcess1"/>
                      </eip:target>
               </eip:XPath-splitter>

               <eip:namespace-context id="nsContext">
                      <eip:namespaces>
                             <eip:namespace prefix="spago">http://it.eng.spago</eip:namespace>
                      </eip:namespaces>
               </eip:namespace-context>
</beans>
```

2.2.12 SplitAggregator

SplitAggregator	
Component family	Standard, Service Engine
Service	TaskService
Service Bindings	JBI-SplitterAggregator
Function	The best definition can be found here: http://incubator.apache.org/ServiceMix/ServiceMix-eip.html#ServiceMix-eip-SplitAggregator The SplitAggregator is an aggregator mainly usefull to collect messages that have been created using a splitter. It relies on several properties that should be set on the exchanges (count, index, correlationId).

2.2.12.1 Configuration Parameters

Aggregate Envelope Name	The tag associate to the envelope containing all

	messages.
Message Envelope Name	The tag associate to the envelope containing the single messsage.

2.2.12.2 Configuration Example

```
<eip:split-aggregator service="foo:myAggregatoraggregator"
                endpoint="myAggregatoraggregator"
                aggregateElementName="MY-ACTIONS"
                messageElementName="">
                <eip:target>
                        <eip:exchange-target service="foo:myScreenOutputaggregator"/>
                </eip:target>
</eip:split-aggregator>
```

2.2.13 Message Filter

2.2.14 AttachmentSplitter

Attachments Splitter	
Component family	Lightweight, Service Engine
Service	TaskService
Service Bindings	JBI-AttachmentSplitter
Function	The attachment splitter component is designed to break an original input normalized message with **n** attachments in **n+1** smaller messages as follow: • The first message will contain only the message content of the original and no attachments. • the others n messages will contain an attachment and a standard message identifying the attachment in the form of: <spagic:ATTACHMENT name=\""+attName+"\"> </ spagic:ATTACHMENT>

2.2.14.1 Configuration Example

```
<sm:activationSpec componentName="myAttachmentSplittersplit_att"
                   service="foo:myAttachmentSplittersplit_att"
                   destinationService="foo:myRoutersplit_att">

        <sm:component>
           <bean class="org.spagic.smx.components.attachments.AttachmentsSplitter">
            </bean>
        </sm:component>

</sm:activationSpec>
```

2.2.15 Attachment To Base64

Attachments to Base64	
Component family	Lightweight, Service Engine
Service	TaskService
Service Bindings	JBI-EncodeAttachments
Function	This component move the attachment contained in the normalized message in the message-content part, encoded in base64 form.

2.2.15.1 Configuration Parameters

Destination element for attachments in base64(*)	The name of the tag that will contain the attachment in base64 encoded form. Note that if there is more than one attachment multiple tag with this name will be produced under the root element of the message content.

2.2.15.2 Configuration Example

```
<sm:activationSpec componentName="myAttachmentToBase64split_att"
service="foo:myAttachmentToBase64split_att"
          destinationService="foo:myBase64ToAttachmentsplit_att">
          <sm:component>
           <bean class="org.spagic.smx.components.attachments.AttachmentsToMessageContent">
            <property name="base64attachmentElementName"  value="BASE64-ATTACHMENT" />
       </bean>
      </sm:component>
</sm:activationSpec>
```

2.2.16 Base64 To Attachment

Base64 To Attachment	
Component family	Lightweight, Service Engine
Service	TaskService
Service Bindings	JBI-DecodeAttachments
Function	This component move the base64 text content of some nodes from the message content as attachment of the normalized message.

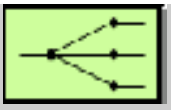
2.2.16.1 Configuration Parameters

Elements that contains base64 attachments	An xpath expression for the selection of nodes that contains base64 encoded text. Note that if there is more than one nodes that satisfy the xpath expression all of this will be transformed in attachments.

2.2.16.2 Configuration Example

```
<sm:activationSpec componentName="myBase64ToAttachmentsplit_att"
service="foo:myBase64ToAttachmentsplit_att"
          destinationService="foo:myScreenOutputsplit_att">
          <sm:component>
           <bean class="org.spagic.smx.components.attachments.MessageContentToAttachments">
            <property name="base64attachmentElementName"  value="BASE64-ATTACHMENT" />
       </bean>
      </sm:component>
    </sm:activationSpec>
```

2.2.17 StaticRecipientList

StaticRecipientList	
Component family	Standard, Service Engine
Service	ParallelService
Service Bindings	JBI-RecipientList
Function	The best definition can be found here: http://incubator.apache.org/servicemix/servicemix-eip.html#servicemix-eip-StaticRecipientList The StaticRecipientList component will forward an input In-Only or Robust-In-Only exchange to a list of known recipients. This component implements the Recipient List pattern, with the limitation that the recipient list is static.

2.2.17.1 Configuration Parameters

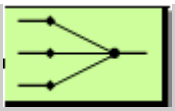
Destination list	The list of the destination endpoints.

2.2.17.2 Configuration Example

```
<eip:static-recipient-list service="foo:myRecipientListsecondProcess"
              endpoint="myRecipientListsecondProcess">
  <eip:recipients>
    <eip:exchange-target service="foo:myScreenOutput 2"/>
    <eip:exchange-target service="foo:myScreenOutputsecondProcess"/>
    <eip:exchange-target service="foo:myScreenOutput 1"/>
  </eip:recipients>
```

```
</eip:static-recipient-list>
```

2.2.18 AggregatorRecipientList

AggregatorRecipientList	
Component family	Standard, Service Engine
Service	ParallelService
Service Bindings	JBI-RecipientListAggregator
Function	The AggregatorRecipientList is an aggregator mainly useful to collect messages that have been created using a StaticRecipientList. It relies on several properties that should be set on the exchanges (count, index, correlationId).

2.2.18.1 Configuration Parameters

Aggregate Envelope Name	The tag associate to the envelope containing all messages.
Message Envelope Name	The tag associate to the envelope containing the single messsage.

2.2.18.2 Configuration Example

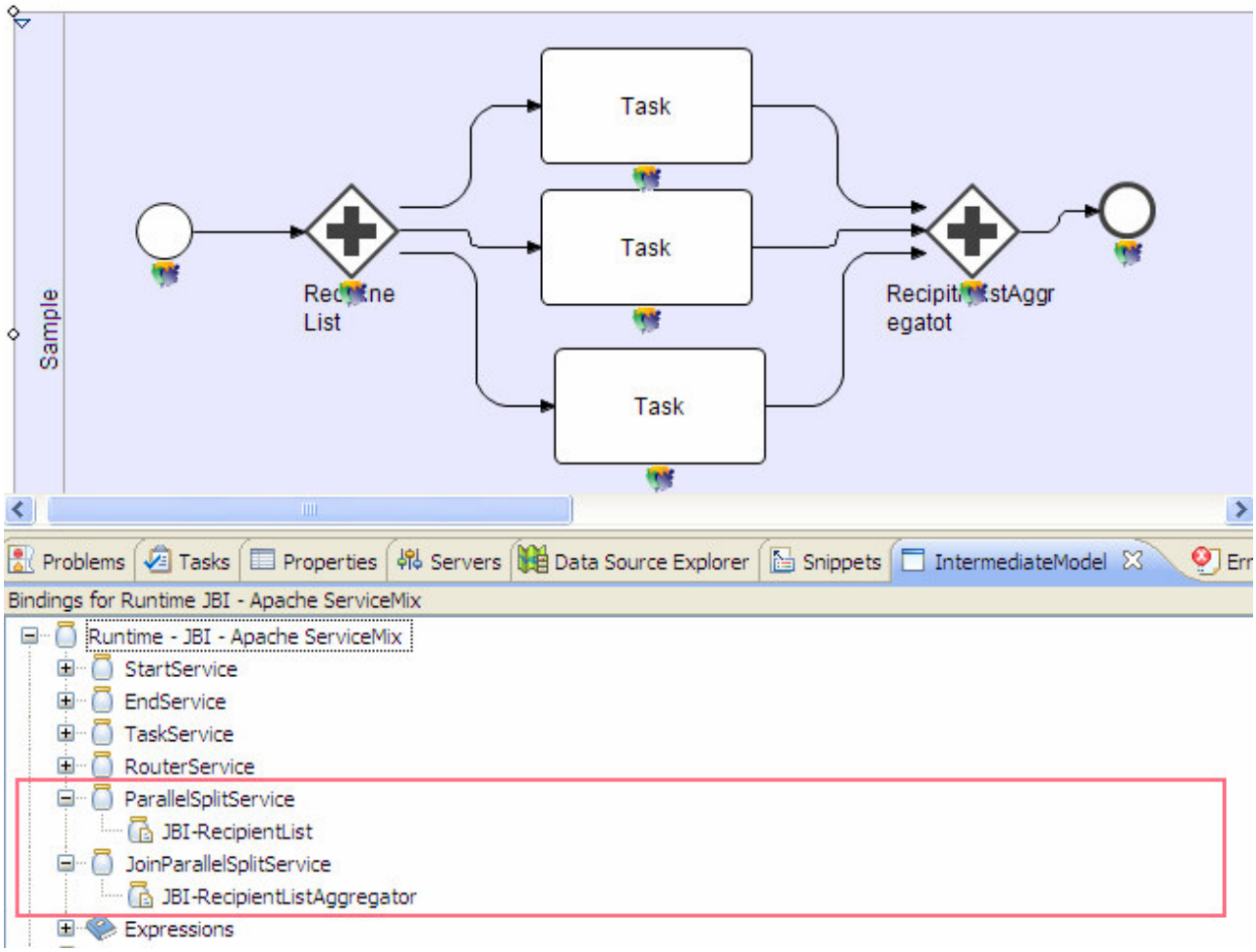
```
<eip:recipient-list-aggregator service="foo:myRecListAggregatorsecondProcess"
endpoint="myRecListAggregatorsecondProcess"
        aggregateElementName="aggregate" messageElementName="message">
        <eip:target>
                <eip:exchange-target service="foo:myScreenOutputsecondProcess"/>
        </eip:target>
</eip:split-aggregator>
```

2.2.19 StaticRecipientList And RecipientListAggregator in BPMN

Usually RecipientList and RecipientList Aggregator are used together, there could be some process, where RecipientList is used alone but you cannot have a RecipientListAggregator without a StaticRecipientList.

Unfortunately in BPMN, the same ***"Parallel Gateway"*** is used both for recipient List and RecipienList Aggregator.

In Spagic Studio as you already seen we're using the annotation to ditiguish it:



In particular we have:

- **Parallel Gateway with ParallelSplitService/JBI-RecipientList**
- **Parallel Gateway with JoinParallelSplitService/JBI-RecipientListAgregator**

2.3 Synchronizer and the JBI Replyservice

Basically the synchronizer component is needed when we want do deploy in JBI some process, that must be invoked with a **synchronous request/response paradigm**, and the same connector (binding component) is used to pass input and get response from process.

This was supported quite well, with the old spagic studio, but the problem is that the new version is targeted to business analyst too, that probably would not want to deal with specific details, like to have a technology specific Synchronizer component and flow within it.

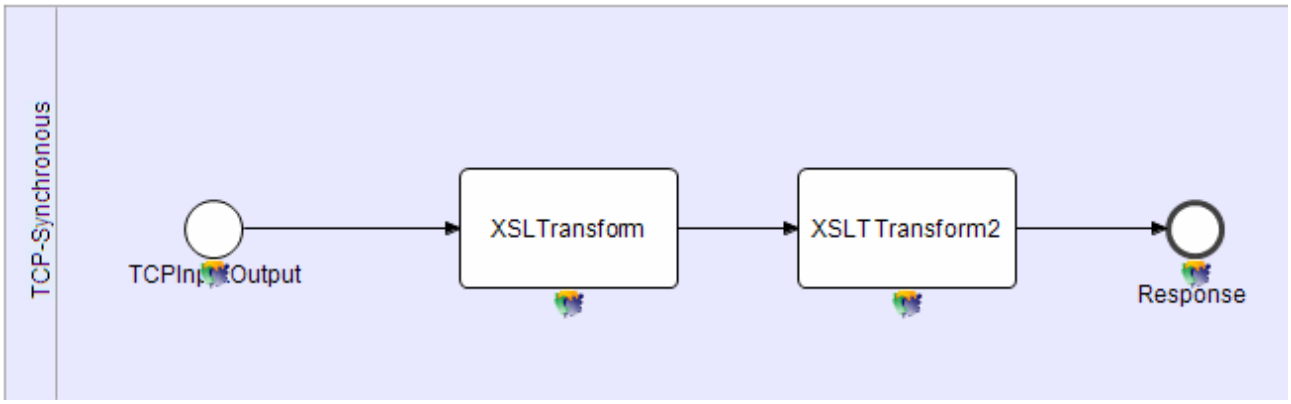
In the new version of Spagic Studio there's a solution, that automatically detect the needs for a "Synchronizer" and provide to change automatically the process flow when we need it.

In particular:

1. There are some specific Service Bindings like **JBI-HttpInputOutputBindingComponent** and **JBI-TcpInputOutputBindingComponents** that automatically configure a synchronizer component

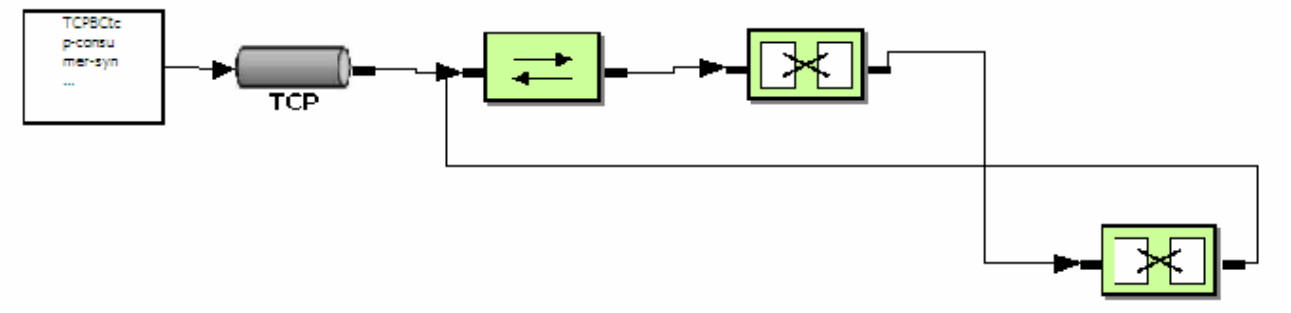
2. There a very easy to use **JBI-ReplyService**, that simple means we want to send response. (This is translated automatically by Spagic Studio in a response to SyncronizerComponent). This means tha when the process developer want that a process end with a response only need to put a bpmn end event and annotate this with a **JBIEndService/JBI-Reply service.**

Suppose that you need to create a Service Assembly using a binding component that supports synchronous request/response pattern, like for example the TCP binding component (or HTTP binding component).
In Spagic Studio 2 you'll model this like the picture below:



This sample case is shown in the following diagram, where the TCP binding component is used together two transform components.If you notice this is very simple and easy to use for process designers.

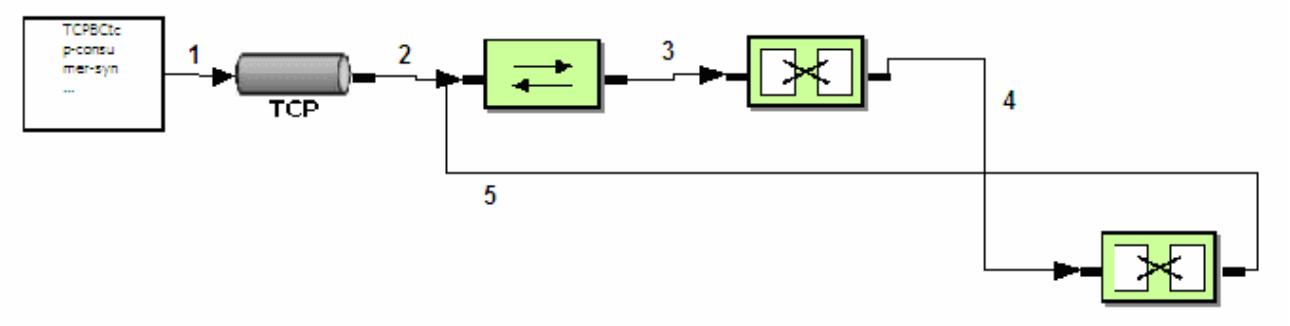
Under the hood spagic will transform automatically this process so there will be the Synchronizer and the right interactions. The result composite applications is the same process, that we've with old spagic studio:



The real difference with the previous version is that know, process developers, could be unware of this.

2.3.1.1 Error management

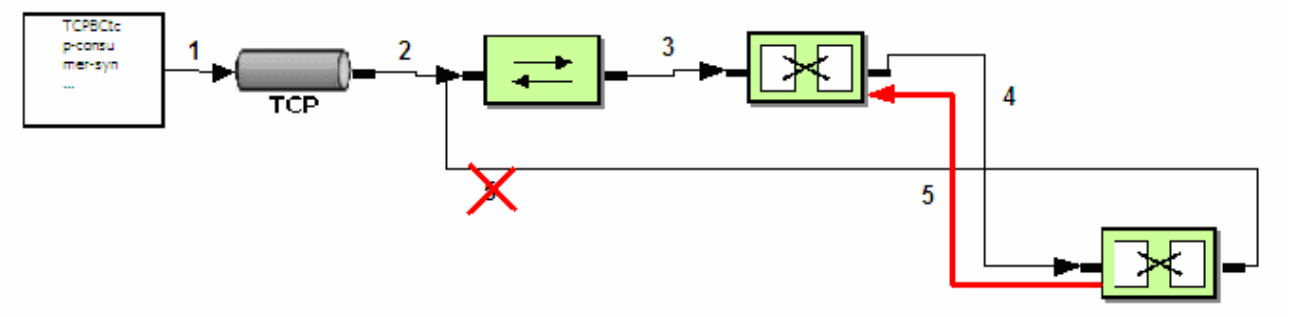
Using the Synchronizer can cause problems when an error arises: to understand the problem, consider first the case when no errors are generated.



The number of exchanges sent for this Service Assembly is 5, and you can see that the Synchronizer receives 2 exchanges: an InOut from the TCP BC, and an InOnly from the second transformer component.

Consider now the case when the second transformer generates an Error.

In this case when the second transformer produce the error, an error message (not an exchange) is sent from the second transformer, to the first one (step 5), and the Synchronizer will never receive the InOnly exchange to use for sending the response to the TCP binding component.



To manage this case we introduced a special listener (*org.spagic.smx.listeners.**SynchronizerFaultListener***) that is able to recognize this case by some properties introduced in the messages by the Synchronizer, and if an error message is generated, it send an Exchange to the Synchronizer, allowing it to send the response for the InOut exchange. It's important for the proper behaviour of this listener, that all components propagate the properties found in the input messages.