Spagic Studio Components

Author: Andrea Zoppello
Nicola Buso
Gianfranco Boccalon

Document Goal

In this document we will focus on how to use the Spagic Studio IDE to deliver SOA solutions based on Spagic solution.
After a brief introduction we will list all available components and we'll show how to use in process designed with Spagic Studio.

Version History

Version/Release n°:	1.0	**Date**	June 02, 2007
Description/Modifications:	First Release (English version)		
Version/Release n°:	1.1	**Date**	August 03, 2007
Description/Modifications:	Added the components AttachmentSplitter, Attachment To Base64 and Base64 To Attachment.		
Version/Release n°:	2.0	**Date**	October 16, 2007
Description/Modifications:	Added the components StaticRecipientList and AggregatorRecipientList.		

1 Introduction

1.1 JBI Concepts

Java Business Integration is a standard specified in JSR 208 that defines a way to develop applications in term of services with a high degree of decoupling. This document doesn't contain the JBI specifications, but it contains some important concepts that must be explained before introducing Spagic Studio IDE.
In a very simple way we can say that in the JBI model an **Application** is composed by a set of ***Services***, and by a set of rules that defines the flow between services. Services are exposed by what we call ***Components***. To be clearer making a comparison with the Object Oriented world we can think to the Component similar to Classes and services to Objects.

An important think to know is that in JBI model each service communicates with others not directly, but passing by a distributed message bus called ***Normalized Message Router*** and with a fixed specified message format called ***Normalized Message.*** Without enter in the details of JBI spec, a normalized message is composed of three parts:

1. A set of properties called **Message Headers**
2. The content of the message called **Payload** or simply **Message Content**
3. A set of attachments

This means that in an application we can have more services belonging to the same components. In a JBI application all services of a particular component are grouped in deployment structures called ***Service Units***.

The JBI specification classifies the components in:

- ***Binding Components***. These are the components that constitute the entry and exit point for JBI application, often called protocol adapters, because their main function is to handle communication and to perform conversion from specific protocols to Normalize Message when they're input components or from Normalized Message to specific protocols when they're output components.
- ***Service Engines***. These are the components that implements business features, inside a JBI application.

The JBI applications are often called ***Composite Applications*** or ***Service Assemblies*** because we've a set of services related to each other by a flow that defines a complex application. Usually JBI applications are used to solve enterprise integration problems; for this reason a Service Assembly can also be called ***Integration Process.*** In this document the term *Integration Process* is preferred over *Service Assembly* or *JBI Application*, because it's easier to understand, but the meaning is the same.
Service assemblies or JBI applications are deployed on ***JBI compliant enterprise service bus*** (ESB).

Finally we can say that a **JBI *Application or Service Assembly is composed of a set of Service Units where each one of these specifies one or more services belonging to a particular component.***

1.2 Spagic and Spagic Studio

As seen in the previous chapter, JBI specifications provide a way to compose applications in term of services, which can be deployed in a JBI compliant ESB.
However there are some important points about enterprise integration that the JBI specification doesn't address:

- **Business Process Management:** At the moment the ESB and JBI are good solution to address integrations problems scenario, but today to have a complete solution we need to address also business problems scenario where the relevant concept are ***(business) processes, processes instance, and relevant data associated to them.***
  In the next topic we're going in the details about BPM.

- **Business Monitoring Capabilities:** Like for the previous point JBI specification at the moment address only the problem of *System monitoring* exposing data with JMX interface, the problem is that this is centred on system resources, like memory usage, JMS queues, etc. *Business monitoring* instead is focused on ***processes instance*** (for both business and integration processes) and relevant data associated to it. Typical business monitoring tool provide features like:
  - **View of all processes (business and integration processes) deployed.**
  - **View of all processes instances,** with information about execution status (terminated with success, failure)
  - **View of relevant data related to a particular process instance**
  - **Querying and Reporting capabilities to obtain high level view of the system**

- **Reporting and Dash boarding:** It's very important in some organizations to have reports and dashboards on what's happening to integration processes.

- **Availability of Visual Graphical Tools**: It's very important to have a graphical tool to visually compose integration processes and generate deployable artefacts for ESB.

Spagic project aims to solve these problems:

- **Providing a Relational Model to support Business Monitoring:** The database model is where data about monitoring are organized and stored.

- **Adding extensions to JBI Server and components**: Spagic provide extension to the JBI server to provide the concepts of process instance and relevant data.

- **Providing an Enterprise Business Monitoring Solution:** Providing one console for all levels of monitoring, from system monitoring to business monitoring and dash boarding.

- **Providing a Graphical Tool (Spagic Studio)**: designer of integration processes can easily model process without worrying about technical details. As we've seen Integration Process can have very complex structure, and defining

manually files that defines the deployment artefact for applications can be a very complex task.

Spagic Studio provides:

- A way to visually design Integration processes in term of services and flow between them
- A generator to get the deployment artefact without writing code and configuration files manually.
- Wizards to publish the service assembly in a relational database to enable monitoring and production of statistic information.
- Tools to configure rules for relevant data extraction after that process have been published in the database.
- Wizard to publish the Service Assemblies in a UDDI registry.

This document covers all Spagic Studio features and the available components.

1.3 Spagic and Business Process Management and Monitoring

ESB solutions and JBI are very good solutions to define integrations scenario processes between applications and services. So they're perfect if we look at them from a technology point of view. The problem is that in most cases target users of software solutions are interested to business process management scenario.
In other terms they want to have a view in terms of:

1. ***Business Process*** (or logical use case)
   A business process simply describes a scenario as a set of a logical tasks:
   a. Business processes are not necessary related to technology concepts.
   b. Some of these tasks could be described with technologic details, but this is not mandatory.
   c. A particular business process instances is identified by a set of relevant data.

2. ***Integration Processes***
   An integration process describes flows and interaction between technology related services.

3. ***Relevant Data.***
   Set of data that will be extracted using rules during process execution. Relevant data are referred to both Business and Integration Process.
   Using relevant data we're able to correlate two or more integration process instances to a particular business process instance.

So there could be business processes without having anything related to technology. Obviously JBI and ESB are related to technology so in Spagic we're interested in Business Processes that has some tasks mapped directly to integration flow.

In Spagic we provide:

- A way to describe a business process as a set of integration processes.
  In Spagic a business process is called ***Iter***
- A way to define rules to extract data from running integration processes**.**
- A way to define the set of data that univocally identify a business process instance, chosen by the set of data of integration processes composing the iter.

1.4 Installation

To install Spagic Studio on a client machine execute the following steps:

1. Get and install a Java Virtual Machine version 1.5.x

2. Get graphviz installation package from http://www.graphviz.org/ and install it. If you're using Windows, simply run graphviz executable file and follow the wizards. During the installation steps please remember the location where the executable dot program is located.
   In Windows (Italian language), default installation graphviz will install the dot program in
   C:\Programmi\ATT\Graphviz\bin\dot.exe

3. Spagic Studio is distributed as an Eclipse plugin. Get it from Spagic distribution and install it on a clean eclipse with the complete WebTools distribution. This can be found here: http://download.eclipse.org/WebTools/downloads/. This release is certified for Eclipse 3.2 and WTP 1.5.

4. Spagic Studio needs to connect to Spagic metadabase. In this section we assume that Spagic metadabase is already installed and configured.

5. A Tomcat installation for JUDDI if you want to publish your services in UDDI Registry. To install JUDDI please refer to the section below.

To start you need to launch eclipse.exe in *SPAGIC_STUDIO_HOME.*

1.4.1 JUDDI installation

Here you find a summary of the JUDDI installation. The project is published on the site: http://ws.apache.org/juddi/; refer to the provided documentation for further info.

Follow the installation steps:

1. Download the release at http://ws.apache.org/juddi/releases.html
2. Extract downloaded archive. Assuming (JUDDI_RELEASE) the folder where the archive is expanded
3. Create the database where to store juddi data. The next steps assume the oracle installation will be done.
4. Execute the script on (JUDDI_RELEASE)/sql/oracle/create_database.sql on the created database.
5. Execute the script on (JUDDI_RELEASE)/sql/oracle/insert_publisher.sql
6. Copy (JUDDI_REALEASE)/webapp/juddi on (TOMCAT_HOME)/webapps/
7. Edit (TOMCAT_HOME)/webapps/juddi/WEB-INF/juddi.properties and change properties as you need.
   i.e.: juddi.operatorName, juddi.discoveryURL
8. Add the following lines at the ${CATALINA_HOME}/conf/Catalina/localhost/juddi.xml file (create it if does not exist):
   Correct parameters based on the juddi database created.

IMPORTANT: note that different version of tomcat have different ways to configure datasources.
Check tomcat docs for further infos. (provided config example is for tomcat 5.0.5)

```
<?xml version='1.0' encoding='utf-8'?>
<Context displayName="jUDDI" docBase="/juddi" path="/juddi"
        workDir="work/Catalina/localhost/juddi">
```

```
<Resource auth="Container" description="jUDDI DataSource" name="jdbc/juddiDB"
     type="javax.sql.DataSource"/>

  <ResourceParams name="jdbc/juddiDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>

  <!-- Maximum number of dB connections in pool. Make sure you
       configure your mysqld max_connections large enough to handle
       all of your db connections. Set to 0 for no limit.
       -->
  <parameter>
    <name>maxActive</name>
    <value>100</value>
  </parameter>

  <!-- Maximum number of idle dB connections to retain in pool.
       Set to -1 for no limit.  See also the DBCP documentation on this
       and the minEvictableIdleTimeMillis configuration parameter.
       -->
  <parameter>
    <name>maxIdle</name>
    <value>30</value>
  </parameter>

  <!-- Maximum time to wait for a dB connection to become available
       in ms, in this example 10 seconds. An Exception is thrown if
       this timeout is exceeded.  Set to -1 to wait indefinitely.
       -->
  <parameter>
    <name>maxWait</name>
    <value>10000</value>
  </parameter>

  <!-- MySQL dB username and password for dB connections  -->
  <parameter>
   <name>username</name>
   <value>juddi</value>
  </parameter>
  <parameter>
   <name>password</name>
   <value>juddi</value>
  </parameter>

  <!-- Class name for the old mm.mysql JDBC driver - uncomment this entry and comment next
       if you want to use this driver - we recommend using Connector/J though
  <parameter>
```

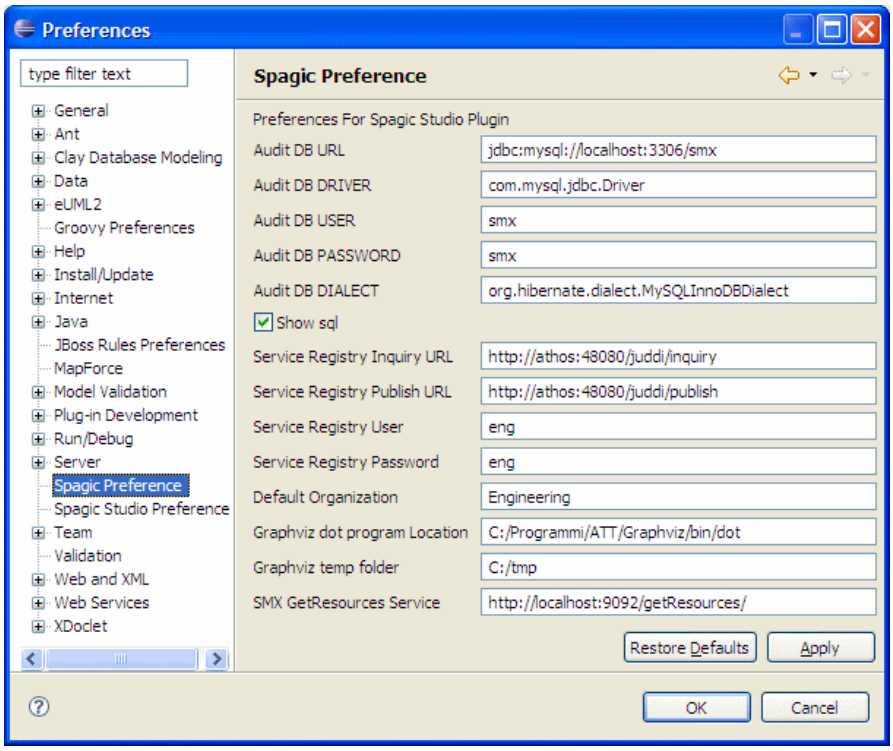
```
            <name>driverClassName</name>
            <value>org.gjt.mm.mysql.Driver</value>
         </parameter>
          -->

         <!-- Class name for the official Oracle driver -->
         <parameter>
            <name>driverClassName</name>
            <value>oracle.jdbc.driver.OracleDriver</value>
         </parameter>

         <!-- The JDBC connection url for connecting to your MySQL dB.
              The autoReconnect=true argument to the url makes sure that the
              mm.mysql JDBC Driver will automatically reconnect if mysqld closed the
              connection.  mysqld by default closes idle connections after 8 hours.
              -->
         <parameter>
           <name>url</name>
           <value>jdbc:oracle:thin:@localhost:1521:thebit01</value>
         </parameter>
       </ResourceParams>

    </Context>
```

9. Check the correct jdbc driver library is present in the tomcat classpath. I.e.: ${CATALINA_HOME}/common/lib
10. Start tomcat.
11. Open a browser on http://<tomcat-host>/juddi/happyjuddi.jsp

1.5 Spagic Preference Page

The first step to do when you open Spagic Studio is to configure preference page.

The Spagic Preference Page is integrated in Eclipse preference dialog (Window\Preference) as shown in the following image:



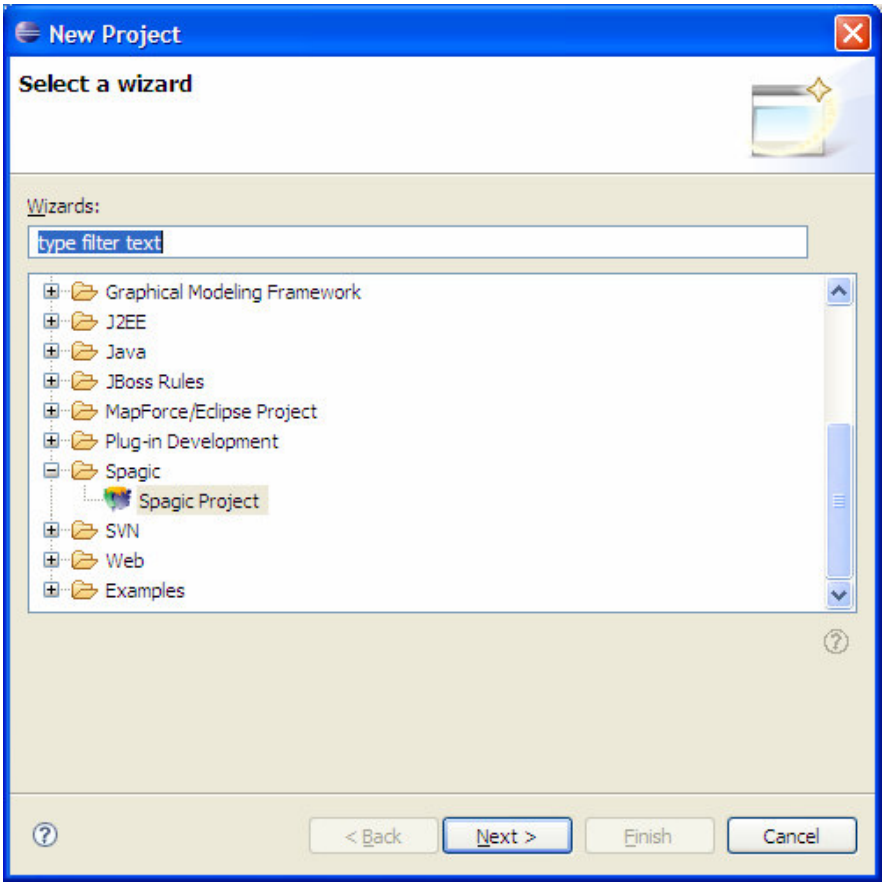
Some parameters relate to connection with Audit Database and service registry, and others are related to the connection with ServiceMix and graphviz.

- **Audit DB URL**: the jdbc url of the database created in the previous section
- **Audit DB Driver**: the full jdbc driver class name
- **Audit DB User**: username for accessing the audit DB.
- **Audit DB Password**: password for accessing the audit DB.
- **Audit DB Dialect**: the name of the hibernate class for the database used (org.hibernate.dialect.MySQLInnoDBDialect for mysql)
- **Show SQL**: Check this only for debug purpose
- **Service Registry Inquiry URL**: the url of inquiry service exposed by juddi
- **Service Registry Publish URL**: : the url of publish service exposed by juddi
- **Service Registry User**: the user that Spagic Studio use to connect to juddi
- **Service Registry Password**: the password that Spagic Studio use to connect to juddi
- **Default Organization:** the default organization where to publish the services
- **Graphviz Dot Program Location:** The path to graphviz dot program
- **Graphviz Temp folder:** The path that graphviz will use as temporary folder
- **SMX Get Resources Services:** The url of the services that Spagic Studio uses to get resources information from a running ServiceMix.

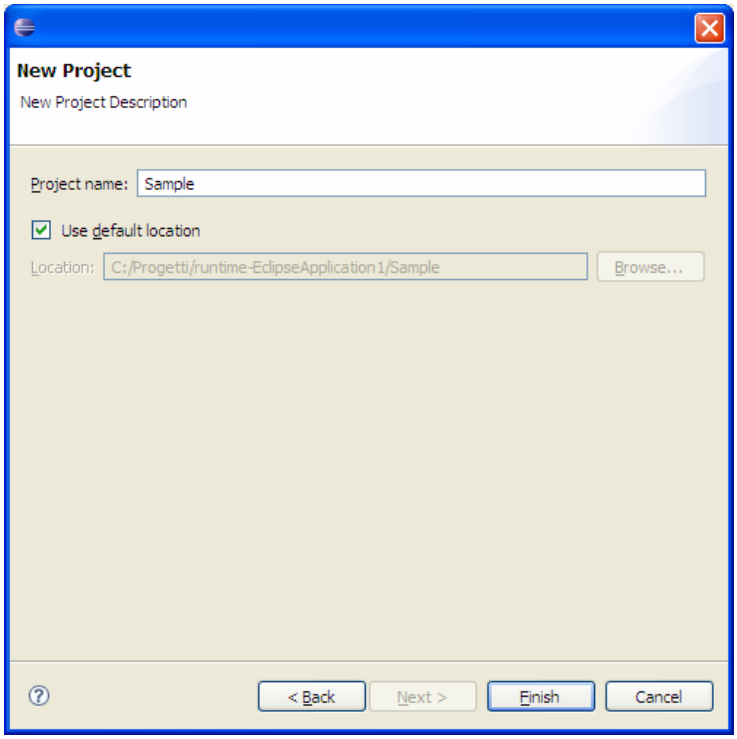
1.6 Creating a Spagic Project

In Spagic Studio the work is organized in Spagic Projects. The better way to understand how a project is organized is to create a new one and to see its structure.

To create a new Spagic Project the Spagic Studio plugin provide a specific wizard in ***File\New\Project*** eclipse menu. So if the tool is installed correctly the new ***Spagic Project Wizard*** should be located in the *Spagic* category:

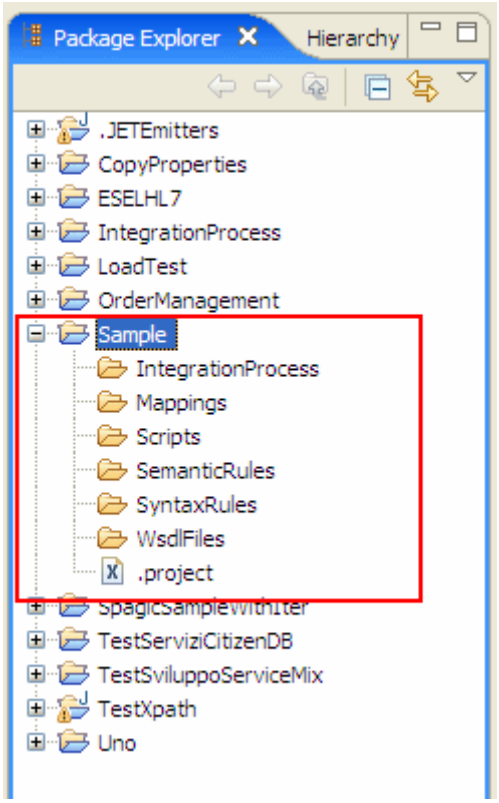


If you click the next button a dialog require to insert the project name and location:



Insert the name (for example “Sample”) of the project and click *Finish*.

In the workspace the Sample Project have just been created with the standard project structure:



As you can see from the image a Spagic Project is composed of the following folders:

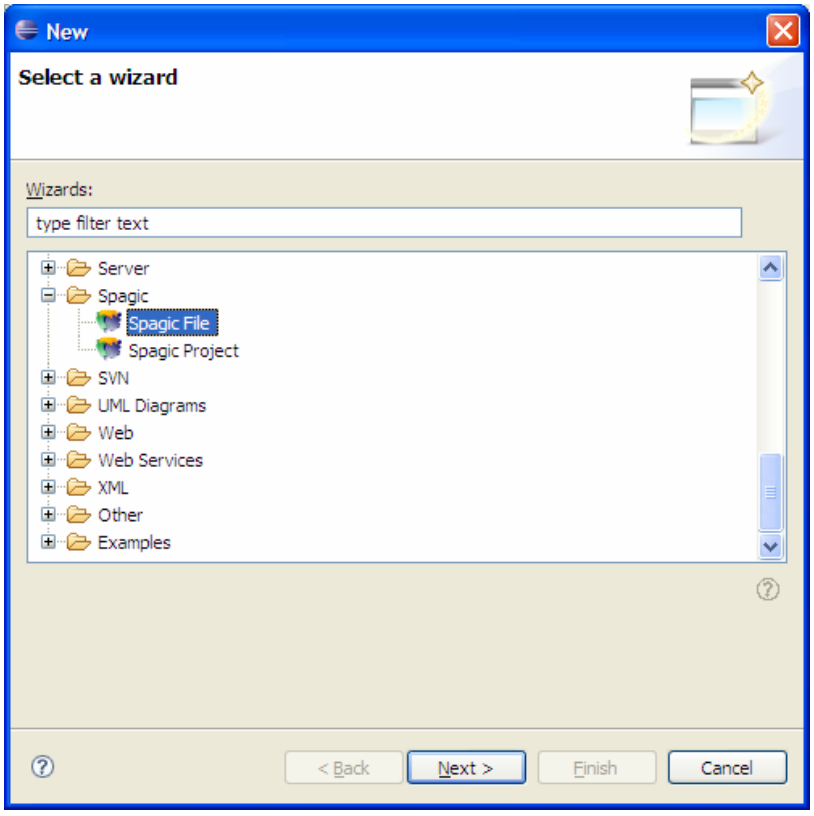
- **Integration Process**: This folder is the most important and contains all Spagic file describing the service assembly in terms of endpoints and flow between them.
- **Mappings**: This folder contains resources that are used by the mapping component. Almost of this resource will be XSLT file.
- **Scripts**: This folder contains resources that are used by Scripting Components. Actually groovy is the language for the scripting, so this folder will contain groovy file.
- **SemanticRules**: This folder contains resources that are used by Semantic Validator Component. Rules are expressed in Drools 3.0 syntax.
- **SyntaxRules**: This folder contains resources that are used by the Syntax Validator Component. The validation of normalized messages is performed by xsd files.
- **WsdlFiles**: This folder contains resources that are automatically generated by Spagic Studio if your process contains entry endpoint relative to HTTP Component configured to be a SOAP Provider. Automatic WSDL generation is provided by Spagic Studio for two important reasons:
    - Client applications of your service assembly needs WSDL to automatically generate clients stub (for example with axis)
    - Once you've generated a WSDL for a particular service assembly some type checking and restrictions can be made in the input of the application changing manually the WSDL generated. If you change the WSDL manually the versions manually modified will be deployed in service assembly structure.

The most important folder of this structure is the IntegrationProcess folder because it's the container of the Spagic file that defines Service Assemblies. Other folders are just container of resources organized in standard structure.

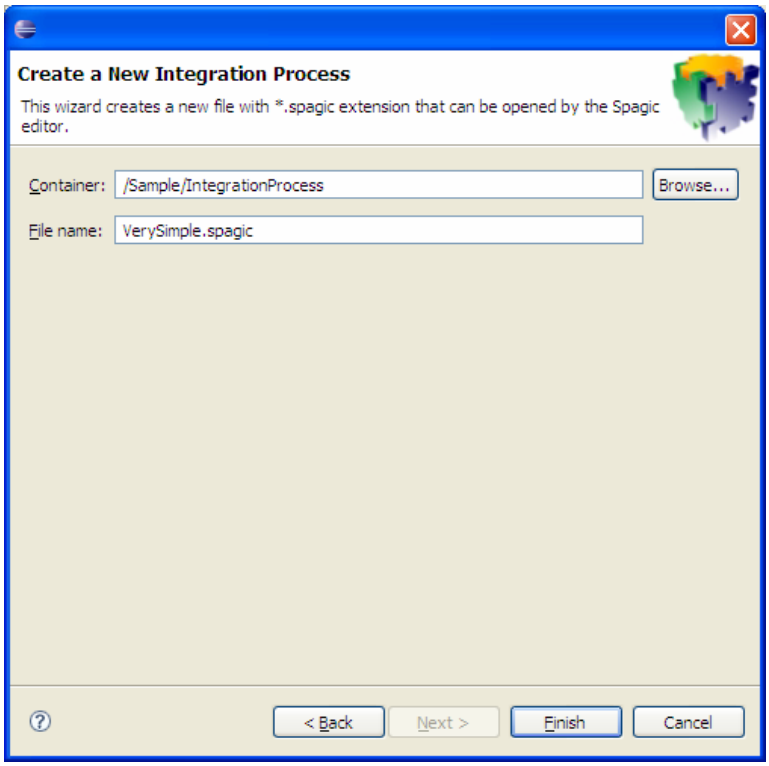
1.7 Creating an Integration Process (Spagic File)

After the creation of the Spagic Project, the next step is to create a new Integration Process.
In this section a very simple example will be created to show the tool usage.

To create the process point the mouse on the IntegrationProcess folder of the "Sample" Spagic project and open the context menu, from here select *File\New\Other* and choose ***Spagic File Wizard***:



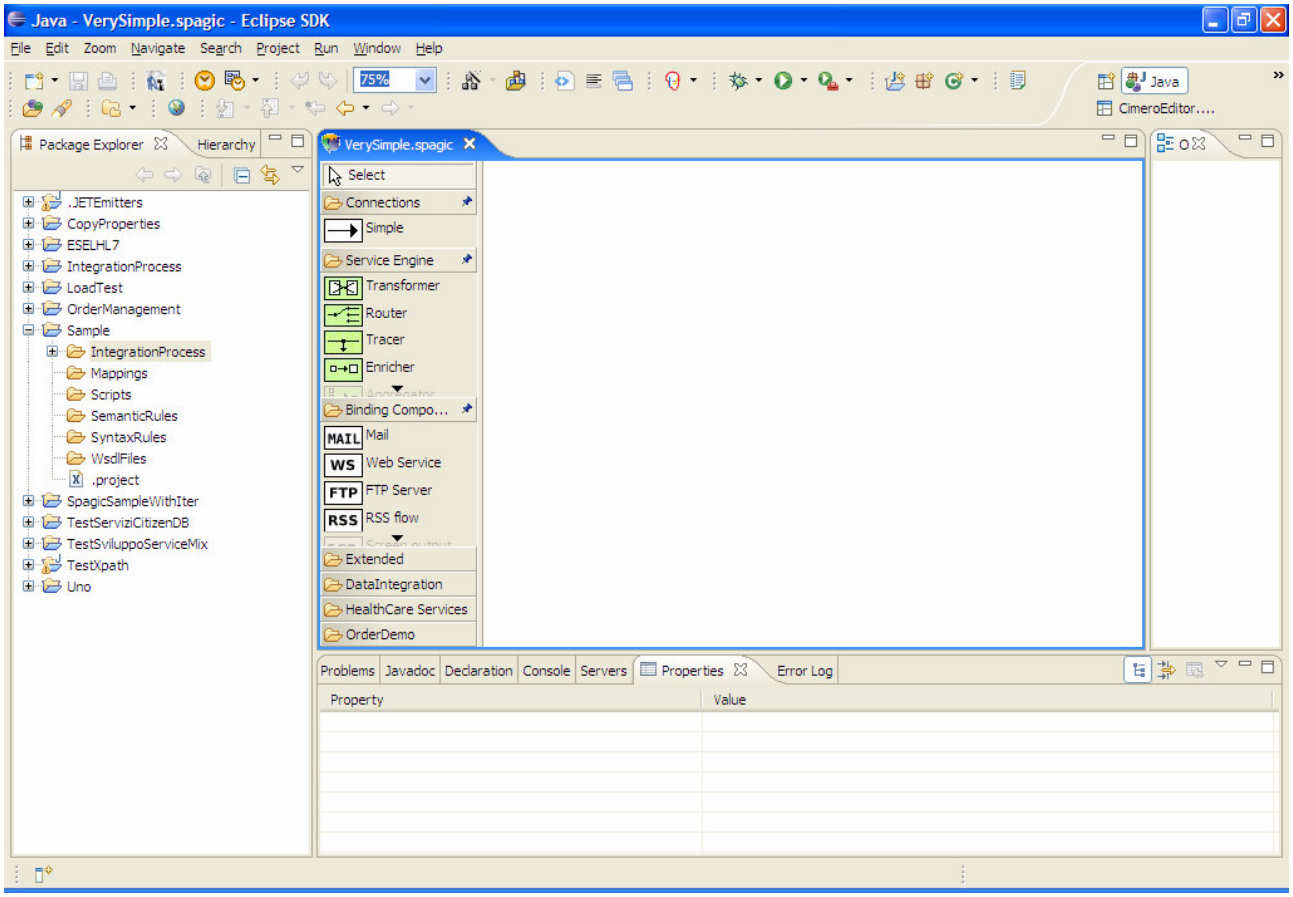
If you click the next button the dialog for creating a new empty project Spagic file will be opened:



Only the container for the processes (preconfigured with the label *IntegrationProcess*) and the file name are required to create the new process.

The file name must have the .Spagic extension.

At the end of the wizard the Spagic file has been created in the project and the visual Spagic editor will be opened.



As you can see the Spagic File Editor is a typical “GEF Editor” where you find:

- **The Components Palette**: It’s located at the left of the editor, when you drag a component from the palette to the editing area, a service relative to the component selected is created in the Service Assembly. Components are classified in the palette.

- **The Editing Area**: When an endpoint is created by a drag and drop from the palette, it’s visible on the editing area. When you click on a particular endpoint, the properties view is populated with the property associated to the endpoint. The editing area is used to connect endpoints using the connection tool of the palette. From the editing area with the context menu it’s also possible to configure rules for relevant data extraction within the endpoint.

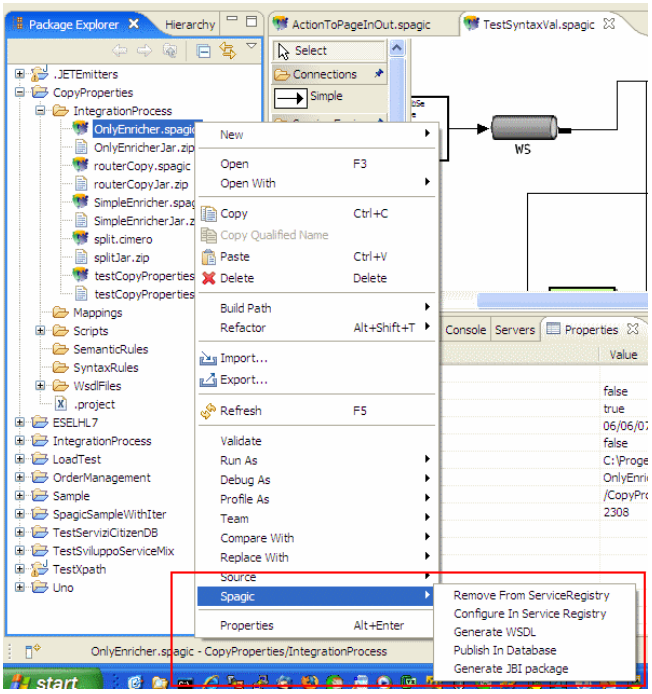
- **The Properties View**: In the properties view you can change the properties of the endpoint currently selected in the editing area.

1.8 Available Operations on Integration Processes

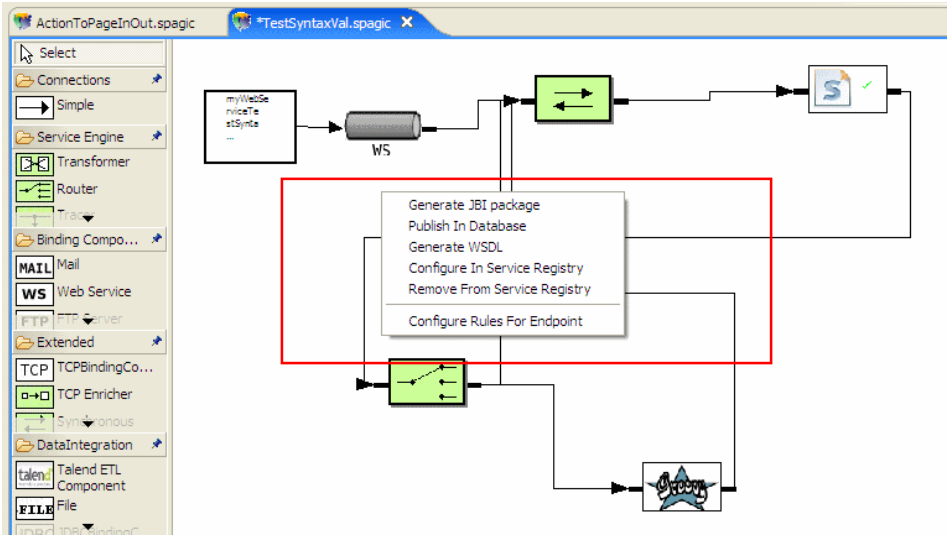
After the creation of a Spagic file, when the composition of the process is finished, the following features are available:

- **Generation of the deployable artefact (Generate JBI Package)**: This feature generates the deployable package for the process composed in the visual editor. The file generated is packaged as a zip file that you must copy in the *deploy* folder of ServiceMix ESB, the default ESB in Spagic architecture.
- **Publishing of the Process in the database (Publish in Database)**: This feature stores the structure of your integration process in a relational database. This information will be used by the auditing listener for monitoring the processes.
- **Automatic generation of WSDL file (Generate WSDL):** It's not mandatory, but for processes that have HTTP endpoints as input it's possible to get a generic WSDL associated. This feature generates a generic WSDL file that can be manually modified. The WSDL file is deployed within the process when the generation of the package is requested.
- **Configuration of the process in a business registry:** If the process can be published in a UDDI registry, with this feature it's possible to publish the service in a UDDI server and classify it. Service classification is based on the concept of *taxonomy* as a predefined set of values. Classifying a service means to assign it one or more values on one or more *taxonomy*. ***Before publishing and classifying a service you must publish it in the database.***
- **Delete a Process from business registry:** The process can be removed by UDDI registry using this feature.

All this operations are available on Spagic file opening the context menu on the navigator view



Or opening the context menu on the editing area:



All operations use the parameters defined in Spagic Preference Page, except for the configuration in business registry described in the next paragraph.

1.9 Business Registries, Publishing and classification of processes

As described from the previous section one of the operation available on integration processes is the publication on a business registry. From a high level point of view we can say that a ***business registry is a repository where we can publish services with their definitions, and provide a classification of these based on taxonomies.***

The important concepts about business registry are:

- ***Organizations***: A service published within a business registry must be associated to an organization that is the provider of the service.

- ***Taxonomies*** are finite set of values defininig “domains”, and classifying a service by a taxonomy means to choose one or more value from the taxonomies and assign to service. Important concepts are:
  1. A service can be classified one or more time within the same taxonomy
  2. A service can be classified one or more time in different taxonomies

- The main feature of a business registry is to provide search features based on organizations and classifications. Typical query for a business registries are:
  1. Search all services provided by “organization X”
  2. Search all services classified as “payments” by the business taxonomy
  3. Search all services provided by “organization X” classified as

Organization and taxonomies need to be defined at the beginning of project and often are stable in time, so in Spagic Studio we offer two utilities to preload organizations and taxomonies text files.

To **"Load organizations"** in Business registry use the following button:

Spagic Studio will ask you for a ".org" file. It's a simple text file like the example:

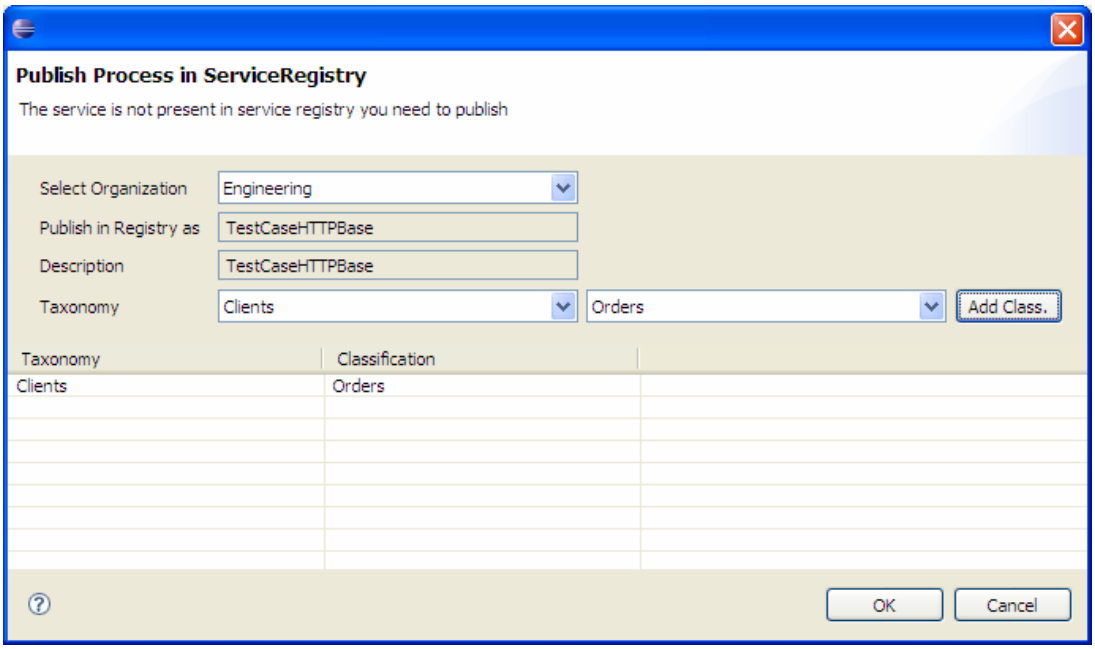
```
#OrganizationName;OrganizationDescription
Engineering;Engineering Ing. Informatica
Bank XYZ;Description for Bank
Institute ABC;Description for Institute ABC
```

To **"Load taxonomies"** in Business registry use the following button:

Spagic Studio will ask you for a ".tax" file. It's a simple text file like the example:

```
#TechnologyTaxonomy
->Java Services
->Web Services
->CICS Services
->Cobol Serivices
```

After you've loadedthe data in the registry, you can publish your services in the business registry with the context menu entry available on Spagic files:



1.10 Configure Rules for Relevant Data Extraction

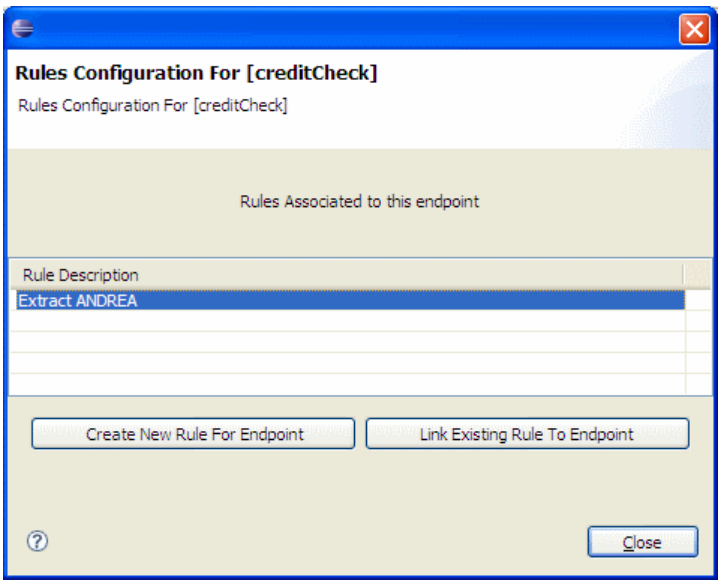
As explained in section 1.2 the concepts of process, process instance and of relevant data of a particular process instances are central for Spagic.

Support of process and process instances concepts in Spagic requires only the publication of the process in metadatabase. Extracting relevant data require the configuration of rules inside Spagic Studio.

The important things to keep in mind about rules for extracting relevant data:

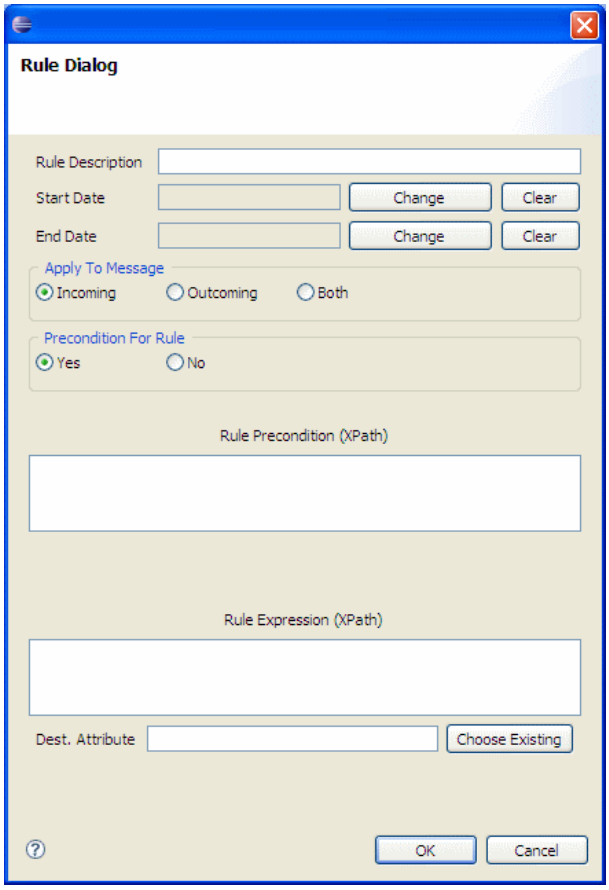
1. ***Rules*** are used to extract relevant data relative to an ***Attribute***.
2. ***Relevant data*** is the value that an Attribute has in a particular process instance.
3. Rule must be an XPath ***Expression***. This because the payload of normalized message must be XML.
4. Rules can be applied only if a precondition has been verified. Precondition for rules are expressed with ***Boolean XPath Expression***. If the precondition is configured the rule is always evaluated.
5. ***Rules are configured on endpoints***. It's possible to choose to apply only to incoming message, only to outcoming message from the endpoint or both.
6. The same rule can be reused in different endpoints.
7. In two different processes the same attribute can be produced by different rules.

To configure rules for endpoint the publication of the process in database is required.
Then it's possible to select and endpoint in the editing area, and select "Configure Rules for Endpoint" entry in the context menu. The following dialog will be opened:



Here the rules already associated with this endpoint are shown.
It's possible to:

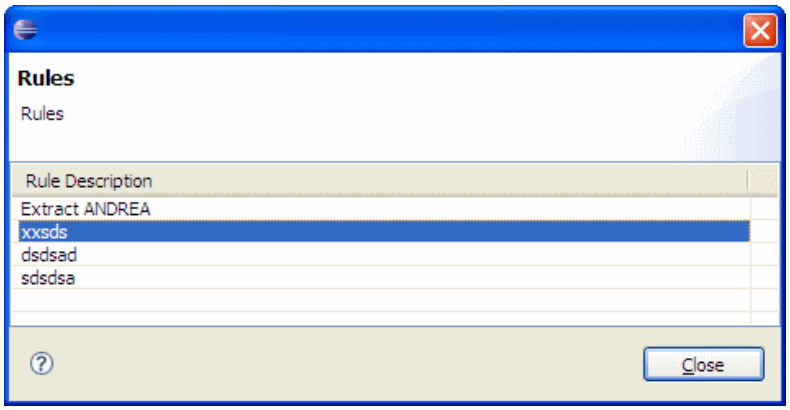
- ***Create a new rule and automatically link to this endpoint*** with *"Create New"* button. The New Rule Creation Dialog will be opened:



Particular attention must be made to the field "Dest. Attribute":

1. *If you want that this rule should produce a **new attribute** you need only to define the attribute name in the text field.*
2. *If you want that this rule should produce relevant data for an **existing attribute** you can choose it with "Choose Existing Attribute" button.*

- ***Associate an existing rule to this endpoint*** with "Associate Existing" button. The following dialog will be open Here you must select an existing rule with a double click.

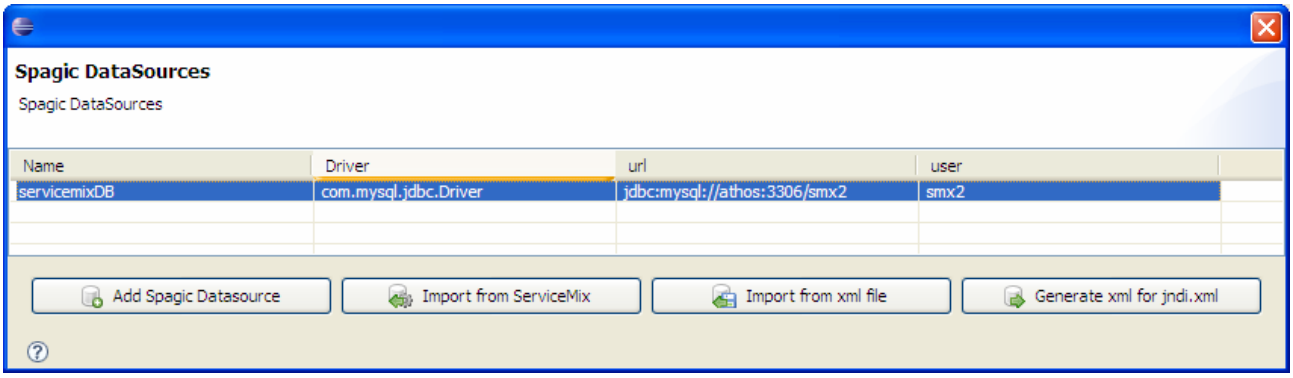


1.11 Working with Datasources

Some components that can be used during the modelling of an integration process are related to database activity and they need to be configured with datasources to work correctly.
Typically the datasources are defined in the ESB by its own configuration file.
To avoid "missing datasource problems" during the deployment phase we need a way to provide bidirectional synchronization between datasource defined in Spagic Studio and datasources defined in the ESB.

In Spagic Studio a database configuration utilities is provided in the toolbar by the following button:

If you click on it a dialog like this will open:



Here you can see the datasource defined in Spagic Studio environment. From here you can:

1. Add a new datasource to Spagic Studio with "Add Spagic datasource button". Pay attention that this feature will add a datasource to Spagic Studio environment not to ServiceMix, so if you want to have the configuration for ServiceMix you need to generate with the "Generate xml for jndi.xml" button.
2. Import datasource definition from a running ServiceMix, this will concat the services "Get Resources" at the url defined in the preference page
3. Import in Spagic Studio datasources selecting directly the configuration file (SMX_HOME\conf\jndi.xml) if ServiceMix is not running.
4. Generate the configuration for ServiceMix for the datasources defined in Spagic Studio with the button "Generate xml for jndi.xml", this service will generate the xml configuration part that you only need to copy to jndi.xml file.

If during import operations, a conflict is detected (for example we can have different configuration in Spagic Studio and ServiceMix for the same datasource), Spagic Studio will ask you to decide what you want to do.

1.12 Iter Types and Iters

Spagic support the concept of ***Iter Types*** and ***Iter***. The link between the concept of ***Iter Type*** and the concept of ***Iter*** is the same that there's between Process and Process Instance.
Iter type is at a higher level of abstraction than the concept of Process. With Iter Type we can logically correlate two or more Process that share a set of common attributes. This is the reason why the same attribute can be produced by different rules.
To define an iter type we need to:

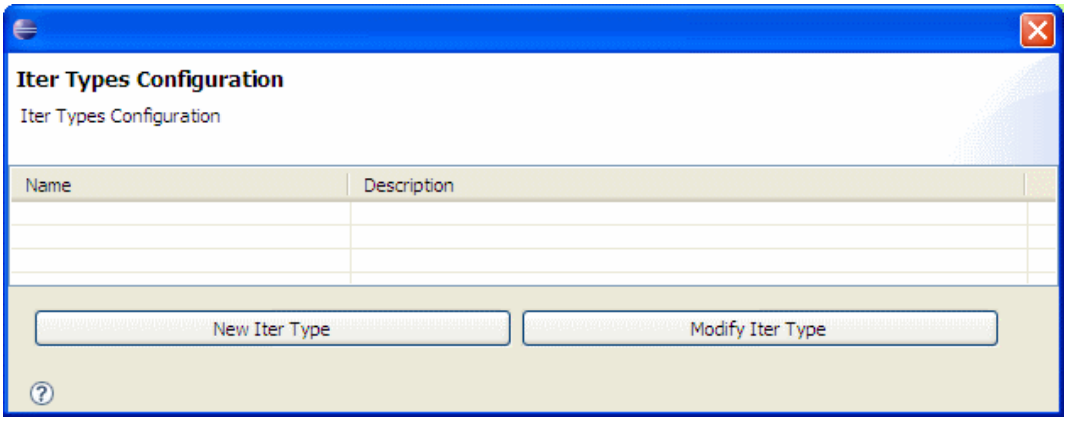
- ***Define the Processes that are part of the Iter.*** The order of the processes is important to allow Spagic creating new Iter instances when the processes are executed.

- ***Define a rule for Iter Type determination***. Because a process can be part of more Iter Types, we need a rule to know how to determine in which Iter to correlate the running process. ***A new iter instance is created when an instance of the process, that is the first in an iter type definition, verify the rule for iter type determination. Rules for iter type determination are Boolean XPath expressions.***

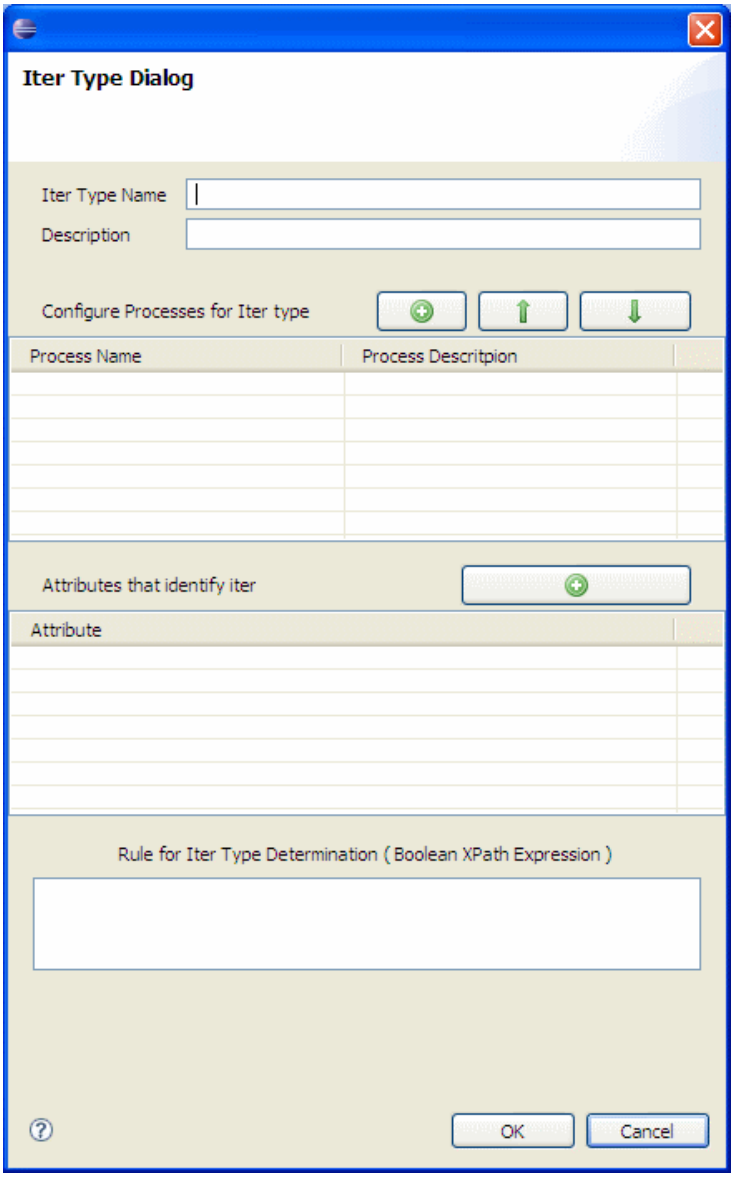
- ***Define the set of attribute of the Iter.*** The correlation of process instances to iter instances is done matching the relevant data of process instances***. If two process instances share the same set of relevant data, and their process definitions are in the same Iter Type definition, then they're part of the same Iter Instance.***

In Spagic Studio it's possible to define Iter Types, using the process already published in the database using the button showed in the following image:

If you press the button the following dialog will be opened:



The dialog show all the iter defined in database. It's possible to delete (canc/del button) an iter type, modify an existing iter or create a new one:



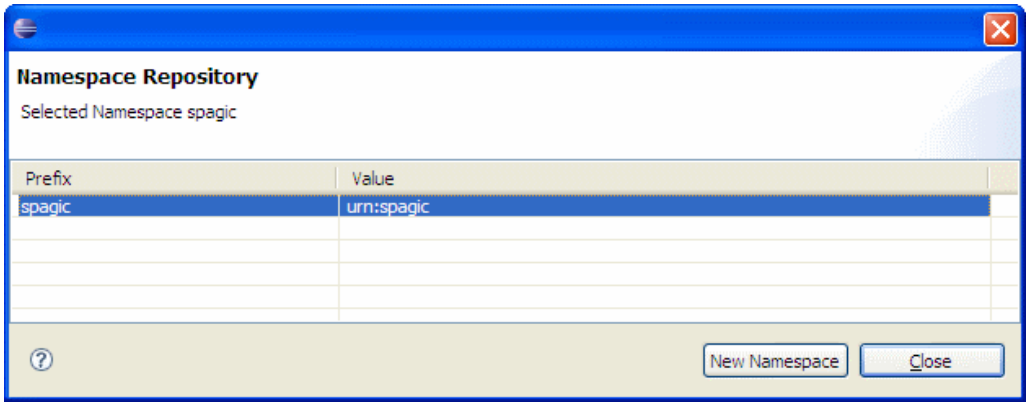
1.13 XPath and Namespace configuration

All the messages exchanged in ESB are Normalized Message, where the most important part (the payload) is XML content. In this context XML and related technologies are very important. One of the most important is the XPath technology. A lot of components and concept in Spagic are related to XPath.
To work correctly in Spagic you need an XPath skill. This document doesn't cover XPath.

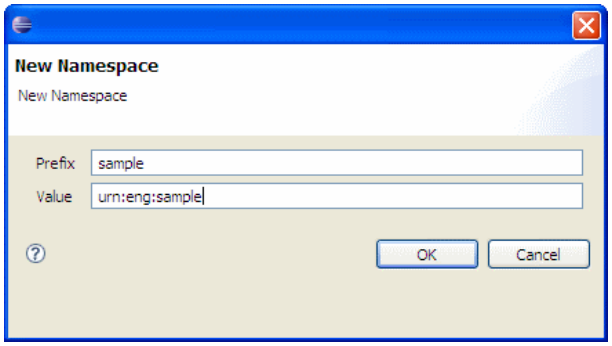
XPath is a quite simple technology to learn but some problems could arise when we're using XPath expression against xml content with namespaces.
It's very important to know that if we want to write an XPath expression using a namespace prefix, the XPath engine must know the namespace value associated to this prefix. Briefly the XPath engine must be aware of all namespace prefix used in XPath expressions. If XPath engine is not configured properly we can have wrong evaluation of expressions.
Spagic components that use XPath allow declaring in their configuration all the namespaces used. To avoid specifying this information for each endpoint, in Spagic Studio the following strategies are defined:

- ***All Namespaces are declared globally.*** Using the button showed by the image:

A typical master detail dialog will be opened:



Where it's possible to see, add, and delete namespaces.



- ***The namespace configuration will be automatically generated for each component that requires it, freeing the process developer to do that manually.***

1.14 Catalogs Configuration for Console

Once rules have been configured for a process, a set of attributes is defined in database.

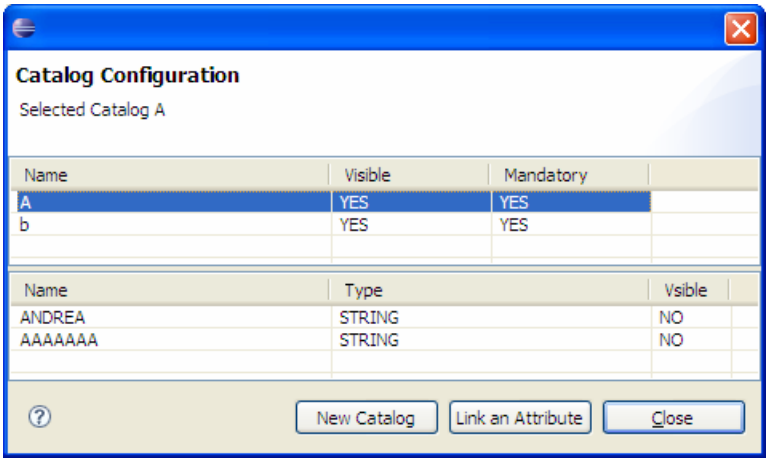
To better organize presentation in the console, attributes can be grouped in logical catalogs.

Within a catalog it's possible:

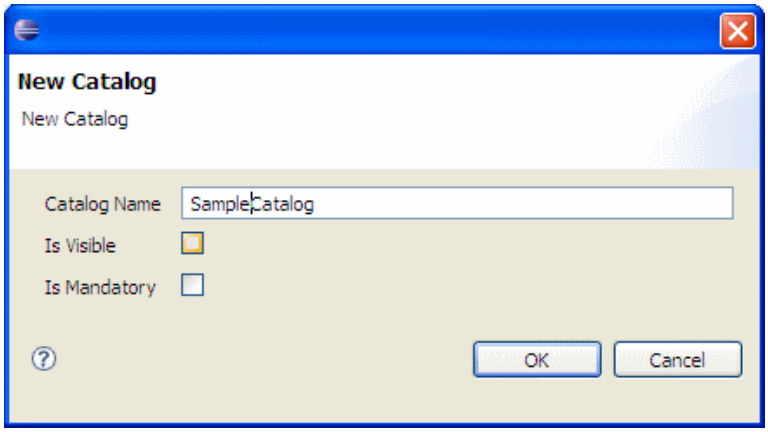
1. Define a presentation orders for attributes
2. Make an attribute visible or hidden.

To open the catalogs dialog hit the buttons highlighted in the image from the Spagic global buttons group:

A two tables dialog will be opened:



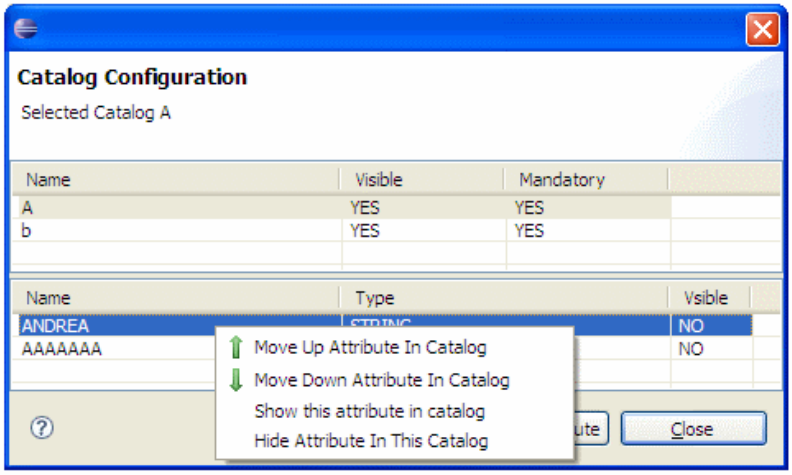
The master table list catalogs in the database, and if we make a selection on it, the table below will be refreshed with the attributes linked to catalog. To create a new catalog just hit the button "*New Catalog*":



With the other button we can link an attribute to a selected catalog; the new attribute will be positioned as the last attribute.

The attributes table has a context menu where after selecting an attribute it's possible to:

1. Move up or down in presentation order
2. Hide or make visible an attribute within the catalog



2 Components

2.1 Binding Components

2.1.1 HTTP Component

HTTP Binding Component	
Component family	Standard, Binding Component
Palette Group	Binding Components
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)
MEP	Message exchange type used by the component when has to create the exchange.
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

Interesting is the use of this binding component in conjunction with the Synchronizer component; 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.

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 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”. 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
Palette Group	Binding Components
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 componect 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 intput 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.
MEP	message exchange type used by the component when

	has to create the exchange.
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
Trusstore 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.

[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”)

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

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 retring 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 riched 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="it.eng.spagosoa.smx.components.tcp.ConsoleTCPReceiver">
            </tcp:tcpBCConfig>

      </tcp:config>

</tcp:consumer>
```

2.1.3 JDBC Poller

JDBC Poller	
Component family	Standard, Binding Component
Palette Group	
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 initialise 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="it.eng.spagosoa.smx.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="it.eng.spagosoa.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="it.eng.spagosoa.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="it.eng.spagosoa.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="it.eng.spagosoa.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="it.eng.spagosoa.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
Palette Group	Binding Components
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
Palette Group	Data Integration
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 BC

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 BC

Sender	
Directory(*)	The name of the *directory* where the files will be written
Append	To write the file in append mode. Valide 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
Palette Group	Binding Components
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
Palette Group	Binding Component
Function	A Standard binding component that handle input or output of the flows from and to JMS queues.

2.1.8 FTP Binding Component

FTP Binding Component	
Component family	Lightweight, Binding Component
Palette Group	Binding Component
Function	A Lightweight, Binding component that handle input or output from and to a JMS server.

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

2.1.9 RSS Binding Component

RSS Binding Component	
Component family	Lightweight, Binding Component
Palette Group	Binding Component
Function	A Lightweigth binding component that handle input or output of the flows from and to rss feeds

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

2.1.10 Timer (Quartz) Binding Component

Timer (Quartz) Binding Component	
Component family	Lightweight, Binding Component
Palette Group	Binding Component
Function	A Lightweigth binding component that allow to start process at predefined interval of time. It use quartz

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
Palette Group	Data Integration
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
Palette Group	Data Integration
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="it.eng.spagosoa.smx.components.jdbcquery.JDBCAdvancedQueryComponent">
       <property name="connConfig">
               <bean class="it.eng.spagosoa.smx.components.jdbcquery.JDBCConnectionConfig">
                      <property name="datasource" ref="jforum"/>
                </bean>
               </property>
               <property name="queryConfig">
               <bean class="it.eng.spagosoa.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="it.eng.spagosoa.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="it.eng.spagosoa.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:

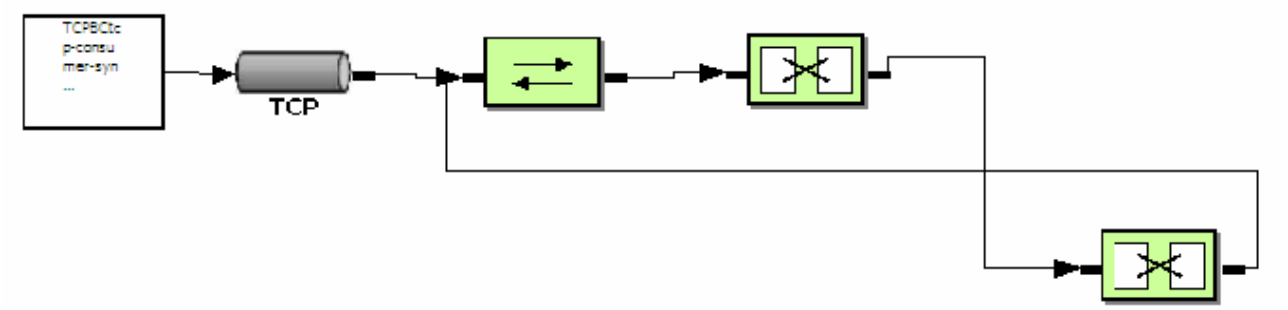
```
<!-- ###################### myJdbcQuery 1 ######################## -->
<sm:activationSpec componentName="myJdbcQuery 1" service="foo:myJdbcQuery 1"
        destinationService="foo:myScreenOutputStoreProcedureTest">
                                    <sm:component>
                              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

Synchronizer	
Component family	Lightweight, Service Engine
Palette Group	Extended
Function	This component allows using In-Only components to create an In-Out ServiceAssembly.
Prerequisites	All the components used in the Service Assembly should

	propagate (or create) the correlation id.

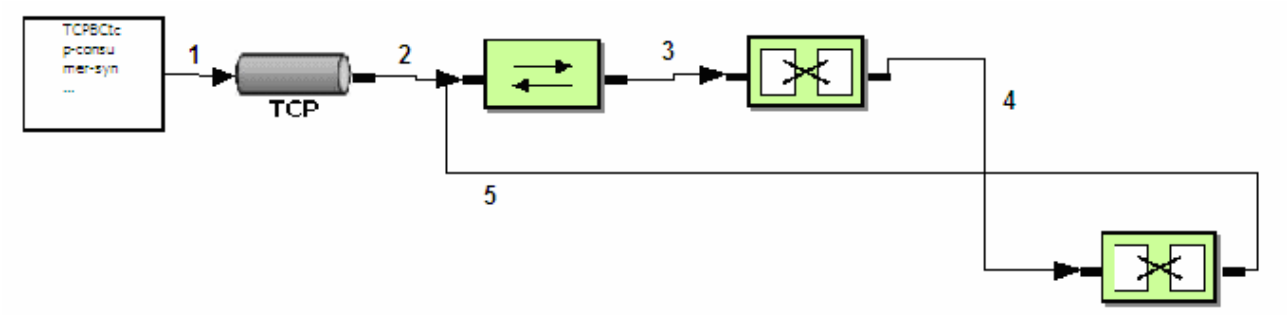
Suppose that you need to create a Service Assembly using a binding component that supports synchronous request/response pattern, like for example the TCP binding component (or HTTP binding component).
Suppose also that the components that you need to use support only the In-Only MEP, which means that they accepts In-Only exchanges and forward In-Only exchanges.
This sample case is shown in the following diagram, where the TCP binding component is used together two transform components.



With the Synchronizer component it is possible to manage this case: the component should be inserted after the binding component that supports the In-Out MEP and before the components that support the In-Only MEP.
The result of the last component (the second transformer component) should be provided to the Synchronizer component, that recognizes that this exchange is related to the request sent by the binding component, and provides the Transformer In-Only message as the response to the binding component.

2.2.3.1 Error management

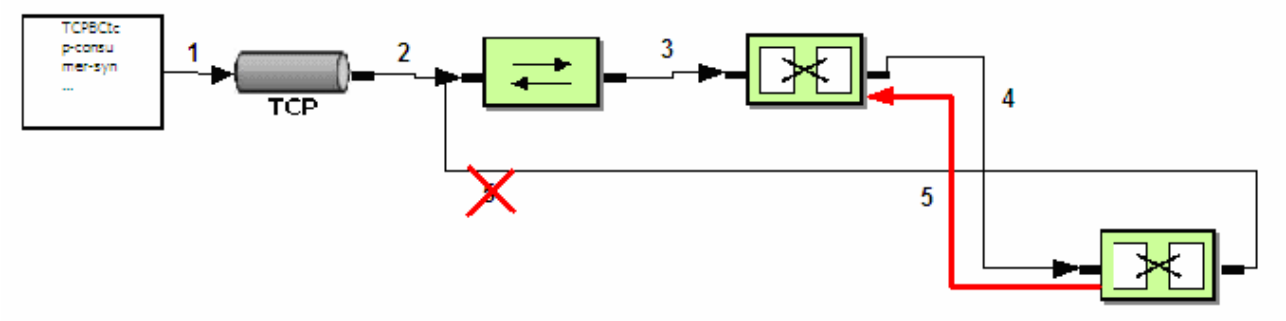
Using the Synchronizer can cause problems when an error arises: to understand the problem, consider first the case when no errors are generated.



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 (*it.eng.spagosoa.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.

2.2.4 Syntax Validator

Syntax Validator Component	
Component family	Lightweight, Service Engine
Palette Group	Extended
Function	This component is used to validate the message content againt xsd file

2.2.4.1 Configuration Parameters

Schema Resource(*)	The path of the xsd file used to validate the message content. The format is: ***<path_to_xsd_relative_to_syntax_rules_folder>*** For example if under the Syntax Rule folder the xsd file sample.xsd under xsd folder the path will be: ***xsd/sample.xsd*** The xsd file must be put in Syntax Rules folder of the Spagic Process. *This file **will not be included** when the service assembly will be generated.*
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.

2.2.4.2 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>
```

2.2.4.3 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 jars on classpath. Schema files are accessed as spring classpath resource so the preferred way to work with xsd is:

1. Organize Syntax Rules Folder in subfolder.
2. Create a jar of xsd resources for each subfolder, and deploy each jar on a service mix directory loaded by classpath. The best place to do that is to put in the <ServiceMix_home>/lib directory.
3. In the Syntax Validator endpoints set in the schema resources properties with the 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>
```

2.2.5 Semantic Validator

Semantic Validator Component	
Component family	Standard, Service Engine
Palette Group	Extended
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

2.2.5.1 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: **project: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. *This file will be included when the service assembly will be generated.*

2.2.5.2 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"/>
```

2.2.5.3 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>" );
```

2.2.5.4 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
```

2.2.6 Groovy Scripting Component

Groovy Scripting Component	
Component family	Lightweight, Service Engine
Palette Group	Extended
Function	This component enables the use of scripting code within an Integration Process. Scripts are expressed in groovy language.

2.2.6.1 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: **project: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. *This file will be included when the service assembly will be generated.*

2.2.6.2 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>
```

2.2.6.3 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.

2.2.6.4 Example of groovy script

2.2.7 Talend Job Caller

Talend Job Caller	
Component family	Lightweight, Service Engine
Palette Group	Data Integration
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. At the moment we support the jobs designed with Talend v2.0.0

2.2.7.1 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.

2.2.7.2 Configuration Example

```
<!-- ###################### myTalendTalendTest ###################### -->
    <sm:activationSpec componentName="myTalendTalendTest" service="foo:myTalendTalendTest"
         destinationService="foo:myScreenOutputTalendTest">
         <sm:component>
          <bean class="it.eng.spagosoa.smx.components.talend.TalendCaller">
```

```
        <property name="talendJobClassName"  value="corso_talend.talendprova.TalendProva" />
      </bean>
     </sm:component>
   </sm:activationSpec>
```

2.2.7.3 Current Limitations

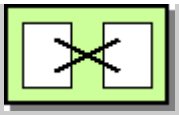
At the moment, from Spagic it's not possible to pass parameters to Talend Job classes.

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.

2.2.8 Transformer (XSLT Mapper) Component

Transformer XSLT Mapper Component	
Component family	Lightweight, Service Engine
Palette Group	Service Engines
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.

2.2.8.1 Configuration Parameters

XSLT File(*)	The path of the xslt file (.xslt) used for the transformation. The xslt file path must be expressed as: **project: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. *This file will be included when the service assembly will be generated.*

2.2.8.2 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>
```

2.2.8.3 Tools for Developing XSLT Transformation

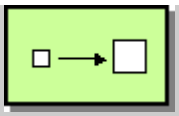
As we've just seen the Transformer xslt mapper component allow us to insert xslt transformation inside the integration process flow. That componet expect an xslt file just ready to use. XSLT is a quite complex language and write manually an xslt file can be quite complex so we need a tools where we can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

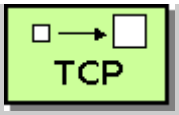
2.2.9 WS Pipeline

The Pipeline components permit to contact an external Web Service.

WS Pipeline	
Component family	Standard, Service Engine
Palette Group	Service Engines
Function	Pipeline on WS socket connection

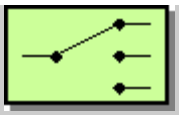
The configuration is similar to the HTTP component configured as provider.

2.2.10 TCP-IP Pipeline

TCP-IP Pipeline	
Component family	Standard, Service Engine
Palette Group	Service Engines
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).

2.2.11 Router

Router	
Component family	Standard, Service Engine
Palette Group	Service Engines
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.

2.2.11.1 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.

2.2.11.2 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:

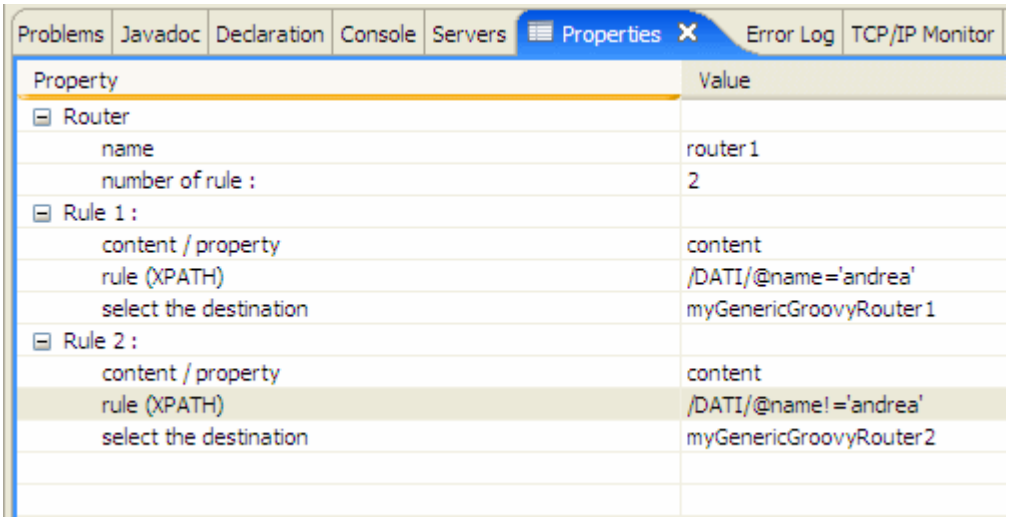
```
when header[<header_name>] <op> <value>
then route to <service>
```

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

2.2.11.3 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.

For example for the the following router configuration that has two rules based on message content:



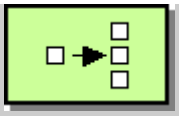
We have a configuration of a drools endpoint in drools service unit:

```
<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
```

2.2.12 XPath Splitter

Splitter (or XPath Splitter)	
Component family	Standard, Service Engine
Palette Group	Service Engines
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.

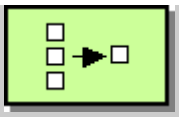
2.2.12.1 Configuration Parameters

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.

2.2.12.2 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>
```

2.2.13 SplitAggregator

SplitAggregator	
Component family	Standard, Service Engine
Palette Group	Service Engines
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).

2.2.13.1 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.

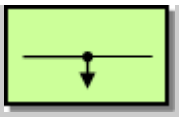
2.2.13.2 Configuration Example

```
<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>
```

2.2.14 Message Filter

2.2.15 Wire Trap component (Tracer)

WireTrap (Tracer)	
Component family	Standard, Service Engine
Palette Group	Service Engines
Function	A WireTap component can be used to forward a copy of the input message to a listener in a proxy fashion.

2.2.15.1 Configuration Parameters

Destination copy(*)	The service to wich a copy of the incoming message will be forwarded.

2.2.15.2 Configuration Example

```
<eip:wire-tap service="foo:myTracertestWireTrap" endpoint="myTracertestWireTrap">
      <eip:target>
            <eip:exchange-target service="foo:myFileBC 1"/>
      </eip:target>
      <eip:inListener>
            <eip:exchange-target service="foo:myScreenOutputtestWireTrap"/>
      </eip:inListener>
</eip:wire-tap>
```

2.2.16 AttachmentSplitter

Attachments Splitter	
Component family	Lightweight, Service Engine
Palette Group	Extended
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:

	<ATTACHMENT name=\""+attName+"\"> </ATTACHMENT>

2.2.16.1 Configuration Example

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

        <sm:component>
           <bean class="it.eng.spagosoa.smx.components.attachments.AttachmentsSplitter">
            </bean>
        </sm:component>

</sm:activationSpec>
```

2.2.17 Attachment To Base64

Attachments to Base64	
Component family	Lightweight, Service Engine
Palette Group	Extended
Function	This component move the attachment contained in the normalized message in the message-content part, encoded in base64 form.

2.2.17.1 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.

2.2.17.2 Configuration Example

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

2.2.18 Base64 To Attachment

Base64 To Attachment	
Component family	Lightweight, Service Engine
Palette Group	Extended
Function	This component move the base64 text content of some nodes from the message content as attachment of the normalized message.

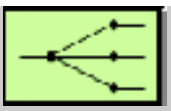
2.2.18.1 Configuration Parameters

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.

2.2.18.2 Configuration Example

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

2.2.19 StaticRecipientList

StaticRecipientList	
Component family	Standard, Service Engine
Palette Group	Service Engines
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.

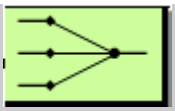
2.2.19.1 Configuration Parameters

Destination list	The list of the destination endpoints.

2.2.19.2 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>
```

2.2.20 AggregatorRecipientList

AggregatorRecipientList	
Component family	Standard, Service Engine
Palette Group	Service Engines
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).

2.2.20.1 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.

2.2.20.2 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>
```

3 Roadmap and evolutions

In next release of Spagic Studio we're going to support the following features:

- ***Export and Import from xml format***. The current Spagic Studio use java serialization to persist the process model in Spagic file. This produce compatibility problems if, for example, we add some properties to the existing components.

- ***Wizard to create Lightweight ServiceMix component and publish them on Spagic Studio Palette***. This feature will allow creating new "business logic components" and using them in the Spagic platform.
- ***Separate the modelling tool from code generation:*** The idea is to evolve the modelling and generation part in separate plugin. In this way it will be possible:
  - ***To use Spagic Studio as a generic modelling environment, and to have different generation plugins. (The modelling plugin could define for example an extension point)***
  - ***To use the code generation plugin with a different tool than Spagic Studio (for example eclipse stp BPMN modeller)***
- ***Support other ESB runtimes***: At the moment Spagic Studio supports the generation of deployable artefacts for ServiceMix 3.1.1.