Spagic 3 - Deployments Models

Author: Andrea Zoppello

Document Goal

In this document we will focus on all deployments models supported by Spagic 3.

Version History

Version/Release n° :	1.0	**Date**	October 5, 2009
Description/Modifications:	First Release (English version)		

1 Introduction

One of the most important feature of Spagic 3 (compared to Spagic 2) is the ability to be deployed in different modes. Within Spagic 2 the only way we've to deploy Spagic was to have the Spagic Server Manager and to invoke process within connectors (ESB Mode).

In Spagic 3 we've change this vision and we're going to support two different deployment modes:

- **ESB Mode (Standard Mode)**: That mode is very similar to the one present on the older version of Spagic; the only difference is that we switch from a JBI based container (servicemix) to an OSGi one (Equinox), but despite this technical details the concept is the same. You've one or more spagic node and your external application are going to invoke services and processes using input connectors (HTTP/SOAP, File/FTP).

- **WAR Embedded Mode:** Within this new mode Spagic 3 is deployed within a Web application and so the spagic service manager has the same lifecycle of the web application context. This is mode is useful if you've a webapplication where you want to embed spagic and you want to be able to call spagic services without the usage of connectors but using co-located calls. In this particular mode a **Spagic Client API** is available to the webapplication to invoke spagic services.

Another important change we've introduced in spagic 3 is a more and clean separation between services and processes. In Spagic 2 the only way you've to invoke a service was to deploy a process.
Now with Spagic 3 if you need to invoke a single service you could:

- **Create a connector and set the service as it's target in (ESB Mode).** This is quite similar to the case where in spagic 2 you're going to create a single process, the main difference is that in spagic 3 you don't need to create a bmn file but you could simply use the service editor to configure a connector and a service.

- **Call the service directly using the Spagic Client API** (Embedded Mode). This feature is very important for all application that need to invoke business service without the overhead of a remote communication.

2 ESB Mode (Standalone Mode)

This is the normal way in which spagic is executed.

Starting spagic in this mode is very simple from the SPAGIC-DISTRIBUTION go to SPAGIC_DISTRIBUTION/service-manager and from the command line type:

- **spagic –console (-debug)**

This **is the fast way to start spagic**, but take in mind that spagic needs to start specifying a **SPAGIC_INSTANCE_HOME** that is the folder that maintains the data of a parctiluar spagic instance in term of services, connectors, datasource and resource.

If not specified the SPAGIC_INSTANCE_HOME is set to **SPAGIC_DISTRIBUTION/service-manager/instance.**

If you want you can specify a SPAGIC_**INSTANCE_HOME** location in **SPAGIC_DISTRIBUTION/service-manager/Spagic.ini** file using the property ***spagic.home.*** An example of the ini file is the following one:

```
--launcher.XXMaxPermSize
256M
-vmargs
-Dosgi.requiredJavaVersion=1.6
-Declipse.ignoreApp=true
-Dosgi.noShutdown=true
-Dspagic.home=C:\Scrappy
-Xms40m
-Xmx512m
```

3 WAR Embedded Mode

The basic idea behind this deployment mode is that the (Equinox) OSGi container is launched and managed within the lifecycle of the webapplication context.

To do this we've used and extended the concept of the servlet bridge that leverage the launching of the Equinox OSGi container within a webapplication.

To run spagic in embedded mode you have two ways:

1. Download a **preconfigured webapp called Spagic Embedded** and deploy on your tomcat application server. This is the way to go if you can't wait and you want to test some simple examples.
2. **Follow some steps, to install spagic embedded in your application.** This is the suggested way to go when you've existing webapplication or you want to start a new webapplication with spagic embedded.

3.1 Steps to install spagic Embedded in Webapplication

Obtain the spagic3 distribution file and unzip it in a folder that we call **SPAGIC_DISTRIBUTION.**

In the following steps we assume that WEBAPP identify the webapplication where you're going to install spagic in embedded mode.

1. Copy the following files in WEBAPP/WEB-INF/lib folder

```
- servletbridge.jar        ( located in SPAGIC_DISTRIBUTION/embedded-files )
- spagic-client-api.jar    ( located in SPAGIC_DISTRIBUTION/embedded-files )
```

2. Create a folder called "eclipse" under WEB-INF so to have the following structure:

```
- WEB-INF/eclipse
---------/lib
```

3. From SPAGIC_DISTRIBUTION copy the file `SPAGIC_DISTRIBUTION/embedded-files/launch.ini` in `WEBAPP/WEB-INF/eclipse` folder

4. Copy the folder `SPAGIC_DISTRIBUTION/service-manager/eclipse/plugins` to the folder `WEBAPP/WEB-INF/eclipse/plugins`.

5. Remove from `WEB-INF/eclipse/plugins the following` folders if they're present:

```
org.eclipse.equinox.launcher.win32.win32.x86_1.0.200.v20090519
```

6. Create a folder called configuration under `WEBAPP/WEB-INF/eclipse.`

7. In the folder WEBAPP/WEB-INF/eclipse/plugins copy the following bundles

```
org.spagic3.client.osgi_1.0.0.jar ( located in embedded-files )
org.eclipse.update.configurator_3.2.100.v20070322.jar.
```

8. Use **the ConfigIniGenerator** Utilities in SPAGIC_DISTRIBUTION/tools/ to generate a config.ini file starting from the "bundles.info" file that is usually located in:

```
SPAGIC_DISTRIBUTION/service-manager/configuration/org.eclipse.equinox.simpleconfigurator/
```

9. Copy the generate config.ini file in from `WEBAPP/WEB-INF/eclipse/configuration`

10. At the end your final WEBAPP sctructure must be something similar to:

```
WEBAPP
   ---WEB-INF
   ------eclipse
   ---------configuration
   --------------config.ini
   ---------plugins
   ---------launch.ini
```

11. The last step is necessary to say your webapp to launch equinox when it's activated so add the following servlet declaration in your web.xml file

```
<servlet id="bridge">
        <servlet-name>equinoxbridgeservlet</servlet-name>
        <display-name>Equinox Bridge Servlet</display-name>
        <description>Equinox Bridge Servlet</description>
        <servlet-class>org.eclipse.equinox.servletbridge.SpagicServlet</servlet-class>
        <init-param>
                <param-name>commandline</param-name>
                <param-value>-console</param-value>
        </init-param>
        <init-param>
                <param-name>enableFrameworkControls</param-name>
                <param-value>true</param-value>
        </init-param>
        <!--
          org.eclipse.equinox.servletbridge and the Servlet API are exported automatically to
          the underlying OSGi framework.
          The extendedFrameworkExports parameter allows the specification of additional java
          package exports.
          The format is a comma separated list of exports as specified by the "Export-Package"
          bundle manifest header.
          For example: com.mycompany.exports; version=1.0.0, com.mycompany.otherexports;
          version=1.0.0
          -->
        <init-param>
                <param-name>extendedFrameworkExports</param-name>
                <param-value>org.spagic3.client.api</param-value>
        </init-param>

        <init-param>
                <param-name>spagic.home</param-name>
                <param-value>C:\Scrappy</param-value>
        </init-param>
        <!--
          You can specify your own framework launcher here.
          The default is: org.eclipse.equinox.servletbridge.FrameworkLauncher
        <init-param>
                <param-name>frameworkLauncherClass</param-name>
                <param-value>org.eclipse.equinox.servletbridge.FrameworkLauncher</param-value>
        </init-param>
        -->
        <load-on-startup>1</load-on-startup>
</servlet>
```

3.2 The Spagic Client API

Once Spagic has been installed in embedded mode within the webapplication, this one could retrieve from its context an object to call spagic services.

To be able to invoke spagic service use the following steps:

- Obtain an instance of **org.spagic.client.api.Client**
- Invoke your spagic service using the method offered by **org.spagic.client.api.Client**

The following code shows three utility methods, the first one to obtain the spagic client the second one to invoke the spagic service with a particular id and to obtain a response, the third one to fire a service invocation without waiting for a response:

```
public Client getSpagicClient(){

        Client client = (Client) getServletContext().getAttribute(SpagicServlet.SPAGIC_DELEGATE);
        return client;

}
```

```
public ClientMessage invokeService(Client clientAPI, String serviceId, ClientMessage requestMessage ){

  ClassLoader original = Thread.currentThread().getContextClassLoader();
  try {
        Thread.currentThread().setContextClassLoader(SpagicServlet.getFrameworkContextClassLoader());
        ClientMessage response = null;
        if (clientAPI != null) {
               response = clientAPI.invokeAndWait(serviceId, requestMessage);
               System.out.println("Message As Body" + response.getBody());
        }
        return response;
  } finally {
        Thread.currentThread().setContextClassLoader(original);
  }
}


public void fireAndForget(Client clientAPI, String serviceId, ClientMessage requestMessage ){

   ClassLoader original = Thread.currentThread().getContextClassLoader();
   try {
        Thread.currentThread().setContextClassLoader(SpagicServlet.getFrameworkContextClassLoader());
        ClientMessage response = null;
        if (clientAPI != null) {
               clientAPI.fireAndForget(serviceId, requestMessage);
               System.out.println("Message As Body" + response.getBody());
        }
    } finally {
        Thread.currentThread().setContextClassLoader(original);
    }
}
```

As you coud easy notice from the code above to call a spagic service you need to prepare an object of type **org.spagic.client.api.ClientMessage** that is simply composed of

- A message Id
- An XML Payload
- A set of properties
- A set of attachments

The same object is used by spagic for the reponse in the case you're going to use the invokeAndWait method.

4 Spagic BPM Gateway

One of the most important and powerful feature of Spagic 3 is it's Business Process Management Engine called BPM Gateway a particular service (yes the BPM Gateway is a service itself in OSGi container) that is able to orchestrate other service and/or human tasks.

Before two proceed it's important to clear some points.

When you're going to deploy a process in BPM Gateway you're really going to do a two phase deploy:

1. First you're going to deploy the process in BPM Gateway Database.
2. Then you're going to deploy a spagic service (of type BPMGateway service) for the process.

In the following of this paragraph we're going to refer to the BPM Gateway Service as the **"Orchestration Service"** for the process.

In particular in Spagic 3 the following type of processes are supported:

- **Tasks driven business process**. In this type of processes all the step are associated to some task activities that could be simple accept/resume activities or task driven by forms to be filled. This task are usually drive by
  - Business Logic of external applications
  - Human Activities performed in gui applications.

  To interface within BPM Gateway the external application must used a well definied api called **Spagic Workflow API**.

- **Automatic processes**. In this type of processes all the steps are associated to spagic service deployed in spagic service manager, this mean that during process execution the bpm gateway call the services interfacing the services container. In that case there's no human interaction within the process execution.

- **Mixed Processes.** In that case you've both humant-task and automatic tasks.

All this type of processes must have at least a **start** and an end **task**.

Regards to the start of a process all the type of the processes above could be:

- **Started Manually**. In that case the application responsible to start the process must obtain an instance of *org.spagic.workflow.api.IProcessEngine* and then use a code similar to the following one to start the process:

```
IProcessEngine engine = /* Obtain the ProcessEngine Impl */;
IControlAPI controlAPI = engine.getControlAPI();
IQueryAPI queryAPI = engine.getControlAPI();

// Start Without pass variable
controlAPI.startByProcessName(processName);
```

```
// Start With VARIABLES
Variable[] vars = /* Initialize Variables */;
long instanceId = controlAPI.startByProcessName(processName, vars);
```

In the following paragraph we're going to see how to obtain the *org.spagic.workflow.api.IProcessEngine* object, the great thing is that apart from the initialization of IProcessEngine object then you don't worry about details all the code that your application use to talk with the BPM Gateway is only dependent on the *IProcessEngine, IControlAPI*,and *IQueryAPI* interface.

- **Started Automatically by a Connector**. In that case no code is needed, it's spagic 3 bpm gateway that start a process instance when a message is received by the connector that has as target the **"Orchestration Service"** for the process.

- **Started Sending a XML Message directly to Orchestration Service (Only WAR Embedded Mode)**. When you use Spagic in embedded mode it's possible to use the Spagic Client API to invoke directly the "Orchestration Service" of the process, in that case the Spagic Client API plays the role of an input connector.

When a process instance ends we could have the following scenario:

- **If the process was started automatically by a Connector that require a response** at the end of the process the special variable called "XML_MESSAGE" is returned as response to the connector.

- **If the process was started automatically by a Connector that not require a response** at the end of the process instance the special variable called "XML_MESSAGE" is sent to the target of orchestration service if this have one.

- **If the process was started manually** it is the same as if the process was started by a fire and forget connector, if a destination connector is found in orchestration service it use otherwise nothing is done.

4.1 Deployment Models and Spagic Workflow API

As we shown in previous paragraph the SpagicWorflow API is the interface available to external application to interact within the spagic bpm gateway to:

- Start Business Process
- Manage Tasks (Accept/Refuse/Complete)

To work with the Spagic Workflow API an “external application” need to do the following steps:

1. Obtain an istance of an object implementing the **org.spagic.workflow.api.IProcessEngine** interface
2. Get the **org.spagic.workflow.api.IQueryAPI**, and the **org.spagic.workflow.api.IControlAPI** interface asking it to the IProcessEngine obtained.
3. Use api
4. Stop the process engine when the application has finished to use it.

```
/*                                  */
/* Spagic Workflow API Typical Usage */
/*                                  */

IProcessEngine engine = /* Obtain the ProcessEngine Impl */;

IControlAPI controlAPI = engine.getControlAPI();

IQueryAPI queryAPI = engine.getControlAPI();

/* Use ControlAPI and Query API Method */

queryAPI = engine.getQueryAPI();

engine.stop();
```

The good thing about the above code is that this code it’s **always the same** expect for the first line where you need to obtain the IProcessEngine object.

With regards to the deployment mode you’re using with spagic 3 you need to get different implementation of the IProcessEngine interface. In particular:

- If **you’re running in ESB Mode (standard)** the BPM gateway expose the api as webeservices. Fortunaltely in your application code you does not really need to generate clients stubs. You simply need to know the url of the webservices and use the **Spagic Workflow WS Client API implementation**, in the following way to get a valid IProcessEngine object:

```
/* Spagic Workflow API – Usage when SPAGIC is running in Standalone ESB Mode */
import org.spagic.workflow.api.ws.client.WSProcessEngine;
…
String url = http://spagichost:9090/wfengine/;

IProcessEngine engine = new WSProcessEngine();
IControlAPI controlAPI = engine.getControlAPI();
IQueryAPI controlAPI = engine.getQueryAPI();
```

- If you're running **Spagic in Embedded Mode,** it's not necessary to invoke webservices to access the BPM Gateway API, it's more useful to use the **Spagic Client API** to invoke services directly in the container. For this scenario too an implementation of the Spagic Workflow API called **Spagic Workflow Embedded API** is provided so you don't worry about the low level details.

```
/* Spagic Workflow API – Usage when SPAGIC is running in WAR Embedded MODE
   We suppose to have access to ServletContext object */

IProcessEngine embeddedEngine = EmbeddedProcessEngine.get(servletContext);
IControlAPI controlAPI = engine.getControlAPI();
IQueryAPI controlAPI = engine.getQueryAPI();
```