Spagic 3 - Deployments Models

Author: Andrea Zoppello
Gianfranco Boccalon

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)		
Version/Release n° :	2.0	**Date**	July 5, 2010
Description/Modifications:	Updates for the SpagicInvoker utility		
Version/Release n° :	3.0	**Date**	July 5, 2010
Description/Modifications:	Update to ebpm version and improvments for some concepts		

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).

- **Web Application Mode:** Within this new mode Spagic 3 is deployed as a Web application and the spagic service manager has the same lifecycle of a web application context. The web application mode is useful also for all the scenario where you already have an application server and you would not to install spagic as separate product.

  This is mode is useful if you've a webapplication and you want to be able to call spagic services without the usage of connectors but **using co-located calls**.

  When Spagic is used in Web Application mode you could use differente scenario:

  - **Embbeded Mode (Direct ClientAP Usage).** In that case Spagic is deployed within your business web application. This mean that you have **only one webapplication with boh spagic and your business logic**. (Described in paragraph…)

  - **InVM Mode (InVM Client API Usage) :** In that case **the Spagic Web Application is installed as alone web application** and separated from other web applications that we'll call Spagic WebClient Applications. (Described in paragraph…)

  With regards to API usage client application will be provided with a ClientAPI that will hide all the complexity of the deployment models giving the programmer the same model.

  The choice of which mode to use depends from specific needs and what you want to achieve. In particular as a general rule if you're going **to develop multiple web applications that need to use spagic the InVM mode is the preferred one**, instead if you're going to develop **one web application that need spagic** you woud like to prefer the Embedded Mode. In the chapter 3 of these documents we'rte going to explain these configurations in details.

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:

- **Spagic3 –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
```

2.1 Remote Access to OSGi Console

When spagic service manager will be deployed in production environment probably it would (and should) not be possible to have direct access to osgi console directly within the machine that runs the spagic service manager.

This is basically because sysadmin could chose to install and run spagic within automatic os scripts and probably they're not granting to you direct access to production machine.

So to enable **spagic administrators (that could be someone different from the system admin)** we need to run Spagic and instrumenting it so we could have access to osgi console accessing it within a remote connection.

To do this we need to choose a port and saying spagic service manager to enable the osgi console to listen on this port.
To enable this on spagic is necessary to specify a port number when running spagic in the following way:

- **Spagic3 –console <OSGI_CONSOLE_PORT_NUMBER> (-debug)**

At this point spagic administrator could simply use the following command to acces remotely the OSGi console:

- **telnet <SPAGIC_HOST_NAME> <OSGI_CONSOLE_PORT_NUMBER>**

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 three alternatives:

1. Download a **preconfigured webapp called Spagic All-In-One** 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. Within Spagic Studio, right click on a Web Application project and choose the command "**Spagic3->Embed Service Manager**". Some configurations have to be defined within the Spagic Preferences.

3.1 Commons Steps to Create a Spagic Enabled 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 from `SPAGIC_DISTRIBUTION/embedded-files/lib` in `WEBAPP/WEB-INF/lib` folder

   ```
   - org.eclipse.equinox.servletbridge.jar
   - org.eclipse.ebpm.servletbridge_{SPAGIC-VERSION}.jar
   - org.eclipse.ebpm.invm.server_{SPAGIC-VERSION}.jar
   - org.eclipse.ebpm.client.api_{SPAGIC-VERSION}.jar
   ```

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` all the folders related to native lauchers so all the file that starts with the prefix `org.eclipse.equinox.launcher.<ANY_SUFFIX>`

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

7. In the folder WEBAPP/WEB-INF/eclipse/plugins copy the following bundle from `SPAGIC_DISTRIBUTION/embedded-files/plugins`

   `org.eclipse.ebpm.client.osgi_{SPAGIC-VERSION}.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>
        <servlet-class>org.eclipse.ebpm.servletbridge.SpagicServlet</servlet-class>
        <init-param>
                <!--  This will enable remote access to the console -->
                <param-name>commandline</param-name>
                <param-value>-console 9898</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.eclipse.ebpm.servletbridge,org.eclipse.ebpm.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.eclipse.ebpm.client.api.Client**
- Invoke your spagic service using the method offered by **org.eclipse.ebpm.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());
        if (clientAPI != null) {
               clientAPI.fireAndForget(serviceId, requestMessage);
        }
    } 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.eclipse.ebpm.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.

3.3 SpagicInvoker utility

To simplify the development of services, allowing the developer to invoke the Spagic services in a way that is independent from their physical location and deployment model, the class *org.eclipse.ebpm.clientproxy.**SpagicInvoker*** is provided. Following there is a sample of usage of the SpagicInvoker: as you can see it's a complete replacement for the ClientAPI.

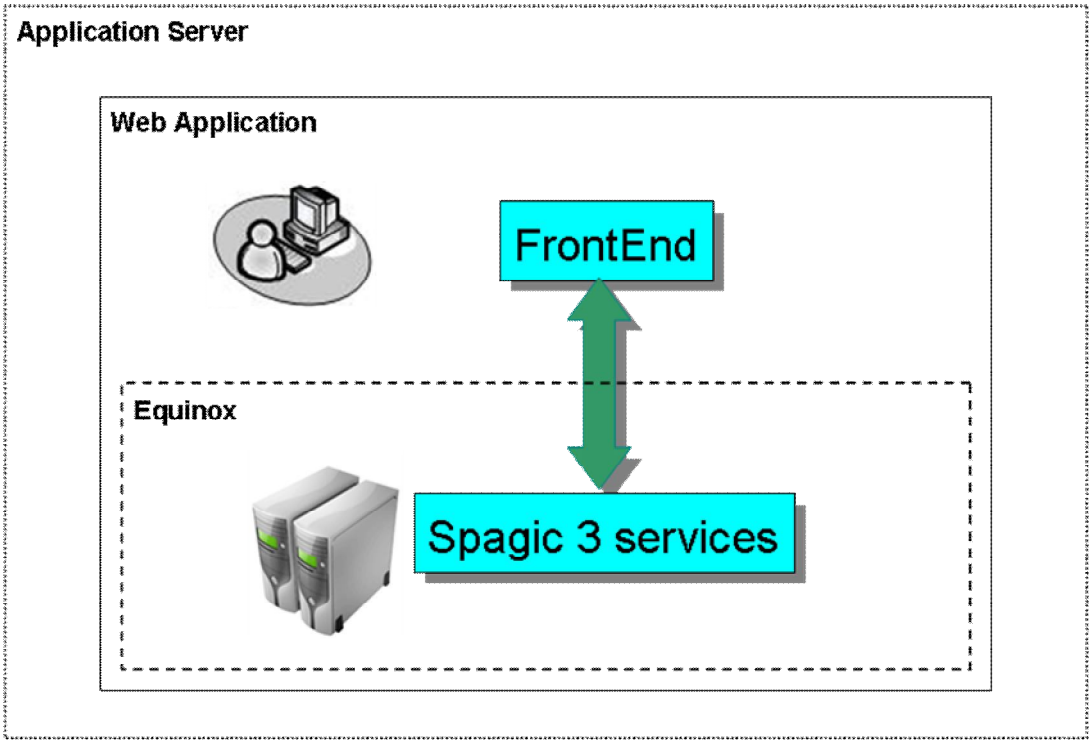
```
SpagicInvoker invoker = new SpagicInvoker();
ClientMessage message = new ClientMessage(idMessage, …..);
ClientMessage responseFromSpagic = invoker.invokeAndWait(spagicServiceID, message);
```

Let see the usage of the SpagicInvoker with the different deployment models and the different configurations to setup.

3.3.1 WAR with Equinox embedded (direct ClientAPI usage)

In this configuration Equinox is embedded with the Web Application that uses the Spagic Services.
In addition to the steps described in “Commons Steps to Create a Spagic Enabled Webapplication” you have to perform the following steps:

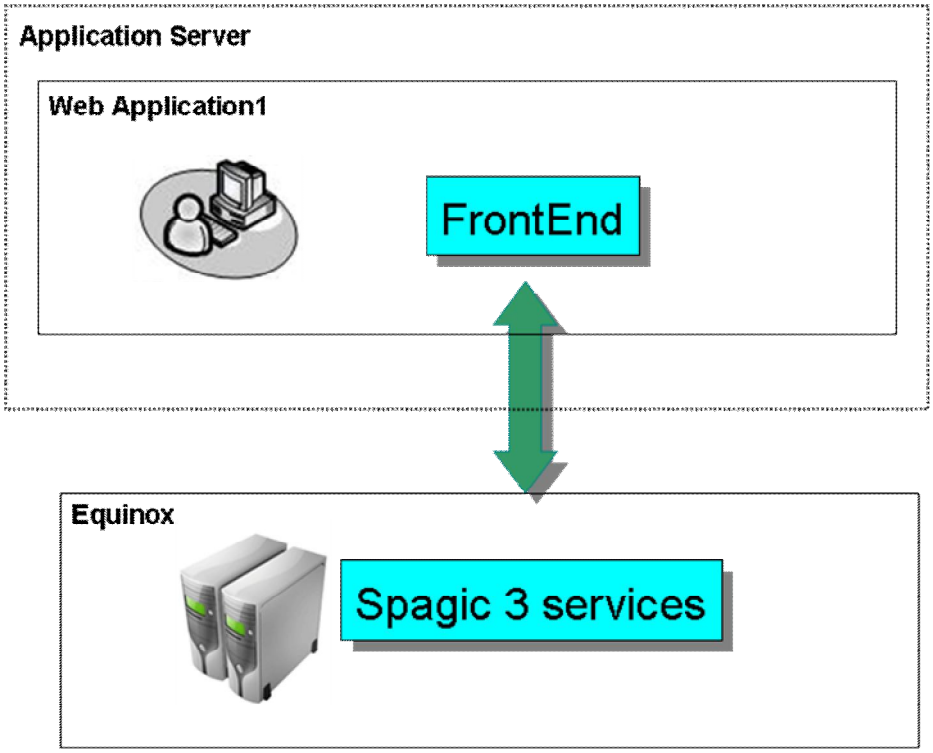
- Add the library “*org.eclipse.ebpm.client.api.jar*” to the WEB-INF\lib folder of the WebApp.



With this configuration the SpagicInvoker will perform direct calls using the ClientAPI.

3.3.2 WAR and Equinox standalone (remote ClientAPI usage)

In this configuration the Spagic services are deployed on a standalone Equinox, so to use them the Web Application should use a connector. This connector is created automatically by Spagic, so you don't have to configure it.



To use this configuration you have to change your Web Application following these steps:

- Add the library "*org.eclipse.ebpm.client.proxy_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.equinox.servletbridge.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.ebpm.servletbridge_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.ebpm.invm.server_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.ebpm.client.api_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add to the web.xml the URL the Spagic connector that exposes the services (you have only to adjust the IP address, changing *localhost* with the right address):

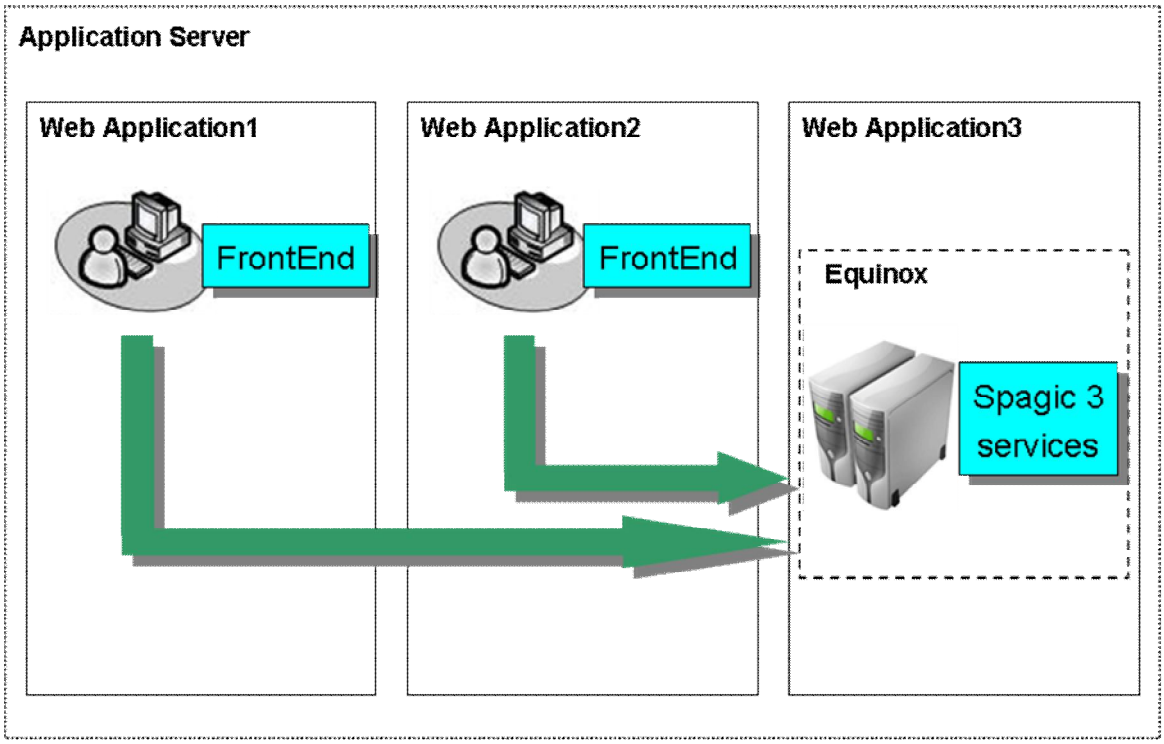
```
<context-param>
        <param-name>spagic.remote.proxy.url</param-name>
        <param-value>http://localhost:9090/client_remote/</param-value>
</context-param>
```

- Add to the web.xml the declaration of the servlet bridge as explained at page 9, paragraph 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").

With this configuration, the calls performed by the SpagicInvoker will use a preconfigured HTTP connector able to invoke all the services deployed on the Spagic Service Manager.

3.3.3 WARs on the same application server (InVM ClientAPI usage)

In this configuration there is one (or more) Web Application that uses the Spagic services deployed on another Web Application (deployed on the same application server).
The SpagicInvoker exploit the fact that all the Web applications are deployed on the same application server, and thus they run within the same JVM: no message serialization is performed and no connectors are used.
This is the configuration most similar, from the performance point of view, to the direct ClientAPI usage.



The first step is to put some Spagic libraries within the libraries of your application server (folder *lib* for Tomcat 6.X, *server\default\lib* for JBoss 5.X):

- *org.eclipse.ebpm.client.api_{SPAGIC-VERSION}.jar*

The second step is to change your Web Application client (Web Application1 and 2) following these steps:

- Add the library "*org.eclipse.equinox.servletbridge.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.ebpm.client.servletbridge_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "*org.eclipse.ebpm.invm.server_{SPAGIC-VERSION}.jar*" to the WEB-INF\lib folder of the WebApp.
- Add to the web.xml the context parameter:

```
<context-param>
        <param-name>spagic.invm.proxy.enabled</param-name>
        <param-value>true</param-value>
</context-param>
```

- Add to the web.xml the declaration of the servlet bridge as explained at page 9, paragraph 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").

The third step is to deploy a Web Application hosting all the Spagic services (Web Application3 in the previous figure).
In addition to the steps described in "Commons Steps to Create a Spagic Enabled Webapplication" you have to perform the following steps of this Web Application:

- Remove the library "*org.eclipse.ebpm.client.api_{SPAGIC-VERSION}.jar*" from the WEB-INF\lib folder of the WebApp.
- Add the library "*log4j-1.2.8.jar*" to the WEB-INF\lib folder of the WebApp.
- Add the library "slf4j-log4j12.jar" to the WEB-INF\lib folder of the WebApp.
- Add, in the web.xml, the following parameter to the SpagicServlet servlet:

```
<init-param>
        <param-name>spagic.invm.proxy.server</param-name>
        <param-value>true</param-value>
</init-param>
```

- Optional: configure the following parameters in SpagicServlet servlet to check the InVM server behaviour:

```
<init-param>
        <!—Thread Pool size of the Client request consumers -->
        <param-name>spagic.invm.proxy.server.poolsize</param-name>
        <param-value>20</param-value>
</init-param>
<init-param>
        <!—Polling sleep interval for the response availability check -->
        <param-name>spagic.invm.proxy.sleepInterval</param-name>
        <param-value>1</param-value>
</init-param>
<init-param>
        <!—Timeout of the SpagicInvoker for the availability of the response -->
        <param-name>spagic.invm.proxy.invocationTimeout</param-name>
        <param-value>30000</param-value>
</init-param>
```

3.3.4 Libraries summary for all deployment options

In the following table we summarize the location of the core libraries for the several deployment options.

SERVER\lib is the libraries folder of the application server (usually changes on several application servers).

✓ means that the library should be copied in the folder.

✗ means that the library should be deleted from the folder.

Library/Deployment Model	**Direct usage**		**Remote usage**		**InVM usage**	
Client Application	**WEB-INF\ lib**	**SERVER\ lib**	**WEB-INF\ lib**	**SERVER\ lib**	**WEB-INF\ lib**	**SERVER\ lib**
org.eclipse.ebpm.client.proxy_{SPAGIC_VERSION}.jar	✓		✓		✓	
org.eclipse.equinox.servletbridge.jar	✓		✓		✓	
org.eclipse.ebpm.servletbridge_{SPAGIC-VERSION}.jar	✓		✓		✓	
org.eclipse.ebpm.invm.server_{SPAGIC-VERSION}.jar	✓		✓		✓	
org.eclipse.ebpm.client.api_{SPAGIC-VERSION}.jar	✓		✓			✓
Library/Deployment Model					**InVM usage**	
Server Application					**WEB-INF\ lib**	**SERVER\ lib**
org.eclipse.ebpm.client.proxy_{SPAGIC_VERSION}.jar						
org.eclipse.equinox.servletbridge.jar					✓	
org.eclipse.ebpm.client.servletbridge_{SPAGIC-VERSION}.jar					✓	
org.eclipse.ebpm.client invm.server_{SPAGIC-VERSION}.jar					✓	
org.eclipse.ebpm.client.client.api_{SPAGIC-VERSION}.jar					✗	✓
log4j-1.2.8.jar					✓	
slf4j-log4j12.jar					✓	

3.3.5 Time comparison between the different options

We report the execution times of some tests that we made with the different deployment models.
We tried to call a simple Groovy service for 1000 times, in the following table we report the means results that we got.
The time in the table is the **total time for the 1000 executions**.

Service Manager Embedded HTTP Proxy	**Service Manager Standalone HTTP Proxy**	**Service Manager Embedded In-VM Proxy EventAdmin using 1 thread**	**Service Manager Embedded In-VM Proxy EventAdmin using 20 threads**
Paragraph 3.3.3	Paragraph 3.3.2	Paragraph 3.3.3	Paragraph 3.3.3
19520 msec	19939 msec	1462 msec	1484 msec

As you can see from the results, using the HTTP Proxy with the Service Manager embedded (with the services on an application, and their clients on another application) and the Service Manager standalone is absolutely equivalent, and there is a factor of ten of performance difference with the In-VM Proxy usage.
In this test there is no difference of performance changing the number of threads used by the EventAdmin.
These tests were executed with the Wait time of the In-VM connector configured to 1ms (see the internal documentation of the connector for the details).

3.4 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.

3.5 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();
```

4 Monitoring with JMX

To enable JMX for monitoring with JConsole or other dedicated applications, add the following lines to the Spagic.ini (for standalone configuration):

```
-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.port=9999
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.local.only=false
-Dcom.sun.management.jmxremote.ssl=false
```

For Linux environments we suggest also to add the following option:

```
-Djava.rmi.server.hostname=<host ip address>
```

This configuration is only a sample, with JMX security disabled. For further options, please read the JMX options for the JVM.