GT42 Adaptor

The Globus adaptor and the GT42 adaptor now work on different environment variables.

The system property GLOBUS_LOCATION is now called **GT42_LOCATION**. For to make work this change I modified also the file *ContainerConfig.java* in the package org.globus.wsrf.config. In the method *getGlobusLocation()* I changed the value of the property to be retrieved. In fact the old one was the GLOBUS_LOCATION, the new one is the GT42_LOCATION.

The system property axis.ClientConfigFile is now **axis.ClientConfigFileGT42**. For to make this works I modified also the file *EngineConfigurationFactorydefault.java* in the package org.apache.axis.configuration I changed the value of the field *OPTION_CLIENT_CONFIG_FILE* that now is axis.ClientConfigFileGT42.

To make the things more clear I also modified the name of the WSDD file in the GT42 folder that now is called ***client-configGT42.wsdd***. To match this other change I had to modify also the value of the string *CLIENT_CONFIG_FILE* and the value of the field fileName inside the class *ClientConfigUtil.java* in the package org.globus.axis.configuration (obviously the new value is the string client-configGT42.wsdd for both of them).

This isn't still enough to avoid the code of the two adaptors mix up. They need two separated InitialContext. A context allows all the naming operations between variables and classes. In fact all naming operations are relative to a context.

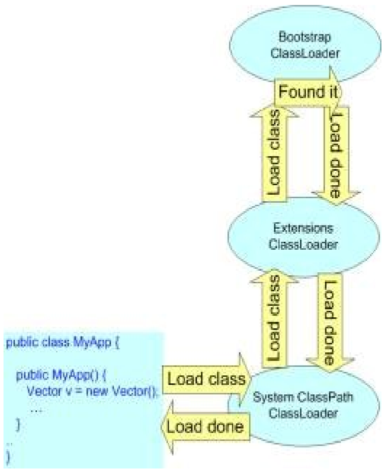
In the Globus Toolkit, the context is created by a *ContextFactory* and the GT 4.0 and the GT 4.2 have two different ones. The GT 4.2 needs the *ContextFactory* created by the class *`javaURLContextFactory`* (package org.globus.wsrf.jndi) that extends the interface *`InitialContextFactory`*. Conversly, the GT 4.0 requires a *ContextFactory* created by the class *javaURLContextFactory* in the package org.apache.naming.java.

The name of the ContextFactory to be loaded is into the system property javax.naming.Context.INITIAL_CONTEXT_FACTORY which value is `java.naming.factory.initial`.

All the problems begin in the class *JNDIUtils* (package org.globus.wsrf.jndi), in fact there is a piece of code that checks if the system property `java.naming.factory.initial` is already set. If it is set, it doesn't change its value anymore.

It is now clear that if somebody runs two jobs sequentially or concurrently the system property is set only once with the value requested by the first GT 4.x adaptor launched and this will throw an exception because the GT4.0 can't work with a context created by the GT4.2 ContextFactory and vice-versa.

To fix this problem, I got to understand better how the java classloader works, because the class InitialContext is inside the package javax.naming and is one of the libraries loaded by the java BootstrapClassLoader. Before to continue, let's take a look inside the java classloader hierarchy.



```
public class MyApp {

  public MyApp() {
    Vector v = new Vector();
    ...
  }
..
}
```

Fig 1 Java Classloader Hierarchy

Has shown in the figure Java applications running from command line involve three classloaders – Bootstrap, Extensions and System-Classpath classloaders. The three class loaders have a parent child relationship among themselves. The most important principle about java classloader is Delegation Principle. According to this principle, if a particular class is not loaded already, the classloaders delegate the requests to load that class to their parent classloaders, even if they are able to load that class. This delegation continues until the top of the hierarchy is reached and the primordial classloader loads the class.

Once I knew that, the first approach I tried to use to fix this problem was trying to "force" the java ClassLoader, so the context loaded by my adaptor would be always the right one. To do this I should be able to load my version of the class *NamingManager* (javax.naming.spi) with a modified version of the method getInitialContext because it is the method that retrieves the value of the system variable `java.naming.factory.initial` and invokes the ContextFactory.

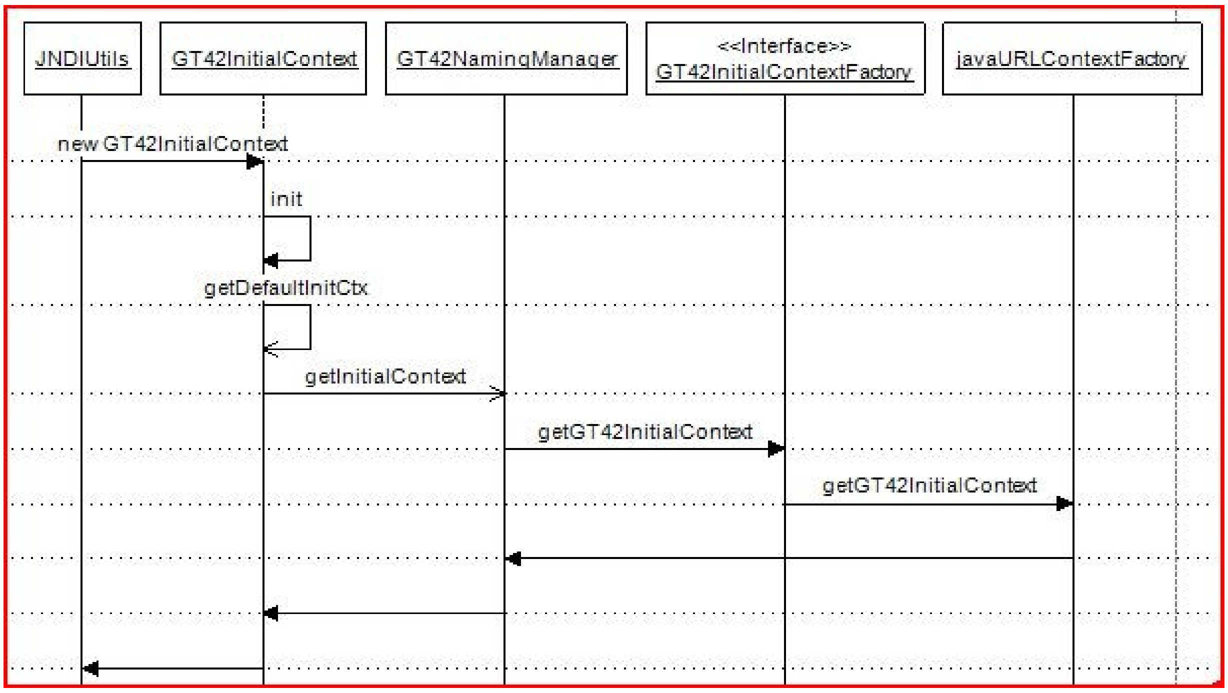
The change made in the class NamingManager was to retrieve a different system property; instead of `java.naming.factory.initial` the `java.naming.factory.initialGT42` previously set to the right value (org.globus.wsrf.jndi.`javaURLContextFactory`) in the initJNDI method inside the class JNDIUtils and invoke the right ContextFactory.

Unfortunatly, I didn't find a solution to "force" the javaclassloader to load my modified NamingManager because as we know it is loaded by the bootstrapClassloader and this means that the “children” of it delegates always him to load the class *NamingManager*. I tried also to create my own classLoader, but the constructor requires the name of a parents like parameter and if you invoke the constructor without parameter the parents will be the bootstrapClassLoader. So I thought that it was better to find another solution.

The solution adopted by me was to “intercept” the first invocation that brought the program into the code in the direction of the *NamingManager* class, and brought it in the direction of my before any interaction with the code from the javax.naming package.

The GT42ADAPTOR code starts to separate from the GLOBUS Adaptor in the method initJNDI. Like I wrote before, after changing the value and the name of the system property to be set (`java.naming.factory.initialGT42`) instead of invoke the costructor of InitialContext class from the javax.naming package I invoke my new class *GT42InitialContext* that like that has the same methods and extends the same interface of the original one but in the method *getDefaultInitCtx* invokes the static method *getInitialContext* from the GT42NamingManager, that also is the modified version of the class *NamingManager* inside javax.naming.spi. At this point the getInitialContext method just retrieves the `java.naming.factory.initialGT42` environment variable and calls the right ContextFactory (org.globus.wsrf.jndi.`javaURLContextFactory`). Finally the from the *GT42InitialContextFactory* object is possible to invoke the method ***`getGT42InitialContext`*** that creates the new NamingContext.

This can sound a little confusing, so maybe the sequence diagram below can help to understand what happens:



RFTGT42 and GridFTPGT42 File Adaptors

Most of the APIs of the RFT GT 4.0 are compatible with the new ones of the RFT GT 4.2.

Obviously, also here I changed the name of the system properties to retrieve and to set (they are GT42_LOCATION and axis.ClientConfigFileGT42).

I made the most important code's changing in the *getCredentialEPR()* and in the *setRequest()* methods because the class BaseRFTClient does't exist anymore in the GT 4.2.

I made no changes in the GridFTP: the new libraries are totaly compatible.

I tested the new Adaptors with all the possible combinations between a local machine, a machine with the GT 4.0 and another with the GT 4.2. In the table are shown the results of the remote copy.

	RFT File Adaptor	GridFTP File Adaptor	GT 4.2 RFT File Adaptor	Grid FTP File Adaptor
LRZ to DAS 3	Version Mismatch	Works	Works	Works
DAS 3 to LRZ	Works	Works	Version Mismatch	Works
LRZ to Local	Version Mismatch	Works	Fails (1)	Works
DAS 3 to Local	Fails (1)	Works	Version Mismatch	Works
Local to LRZ	Fails (2)	Works	Fails (2)	Works
Local to DAS 3	Fails (2)	Works	Fails (2)	Works

Table 1: Supported Operations Table

(1) setRequest fails.(Because there isn't a GridFTP Server on the local host)

(2) getMultipleResourceProperties, java.net.ConnectionException: Connection refused. .(Because there isn't a GridFTP Server on the local host)

GT42 Resource Broker

The GT42 Resource Broker is similar to the WSGT4 Resource Broker. Also in this case I had to to some changes due to the new GT42 APIs: for example one of the key changes in the specification is that Reference Parameters are used now rather than Reference Properties. I made all the required changes inside the method *submitJob*.

Also here I changed the value of the system properties.