FIND RESOURCES IMPLEMENTATION

Introduction

The method aims to retrieve a list of resources that have some features desired by the user. The features are the ones provided by Ganglia, that is the information provider running on the DAS3.

To implement the method it needs to be queried the Index Service with an Xpath query. Once received the query result it has to be parserized, because we need a more readable response and, most of all, a response that fits the javaGAT directives about the metrics.

To parserize the result, the JDOM utility was very helpful because it allows to create and read XML document in a very easy way. But let's take a look to all the the classes created or modified.

XMLUtil

It has four methods. They are:

CreateXMLDocument: is the method that creates the XML document that will be stored in the memory. During the creation there is also the “translation” of the elements in a more readable and “javaGAT compliant” way. What it does can be easily understood in the next two figures:


```
− <ns11:AggregatorData>
  − <ns1:GLUECE>
    − <ns1:Cluster ns1:Name="ClusterVisionCluster" ns1:UniqueID="ClusterVisionCluster">
      − <ns1:SubCluster ns1:Name="main" ns1:UniqueID="main">
        − <ns1:Host ns1:Name="node001.beowulf.cluster" ns1:UniqueID="node001.beowulf.cluster">
            <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0" ns1:CacheL1I="0" ns1:CacheL2="0" ns1:ClockSpeed="2412" ns1:InstructionSet="x86_64"/>
            <ns1:MainMemory ns1:RAMAvailable="3183" ns1:RAMSize="3948" ns1:VirtualAvailable="7166" ns1:VirtualSize="7949"/>
            <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="2.6.18-clustervision-136.1_cvos"/>
            <ns1:Architecture ns1:SMPSize="4"/>
            <ns1:FileSystem ns1:AvailableSpace="138575" ns1:Name="entire-system" ns1:ReadOnly="false" ns1:Root="/" ns1:Size="240875"/>
            <ns1:NetworkAdapter ns1:IPAddress="10.141.0.1" ns1:InboundIP="true" ns1:MTU="0" ns1:Name="node001.beowulf.cluster" ns1:OutboundIP="true"/>
            <ns1:ProcessorLoad ns1:Last15Min="0" ns1:Last1Min="1" ns1:Last5Min="0"/>
          </ns1:Host>
        − <ns1:Host ns1:Name="node002.beowulf.cluster" ns1:UniqueID="node002.beowulf.cluster">
            <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0" ns1:CacheL1I="0" ns1:CacheL2="0" ns1:ClockSpeed="2412" ns1:InstructionSet="x86_64"/>
            <ns1:MainMemory ns1:RAMAvailable="3142" ns1:RAMSize="3948" ns1:VirtualAvailable="7090" ns1:VirtualSize="7949"/>
            <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="2.6.18-clustervision-136.1_cvos"/>
            <ns1:Architecture ns1:SMPSize="4"/>
            <ns1:FileSystem ns1:AvailableSpace="163830" ns1:Name="entire-system" ns1:ReadOnly="false" ns1:Root="/" ns1:Size="240875"/>
            <ns1:NetworkAdapter ns1:IPAddress="10.141.0.2" ns1:InboundIP="true" ns1:MTU="0" ns1:Name="node002.beowulf.cluster" ns1:OutboundIP="true"/>
            <ns1:ProcessorLoad ns1:Last15Min="0" ns1:Last1Min="1" ns1:Last5Min="1"/>
          </ns1:Host>
```


Figure 1

Instead, the figure 2 shows how the informations are stored inside the javaGAT document:

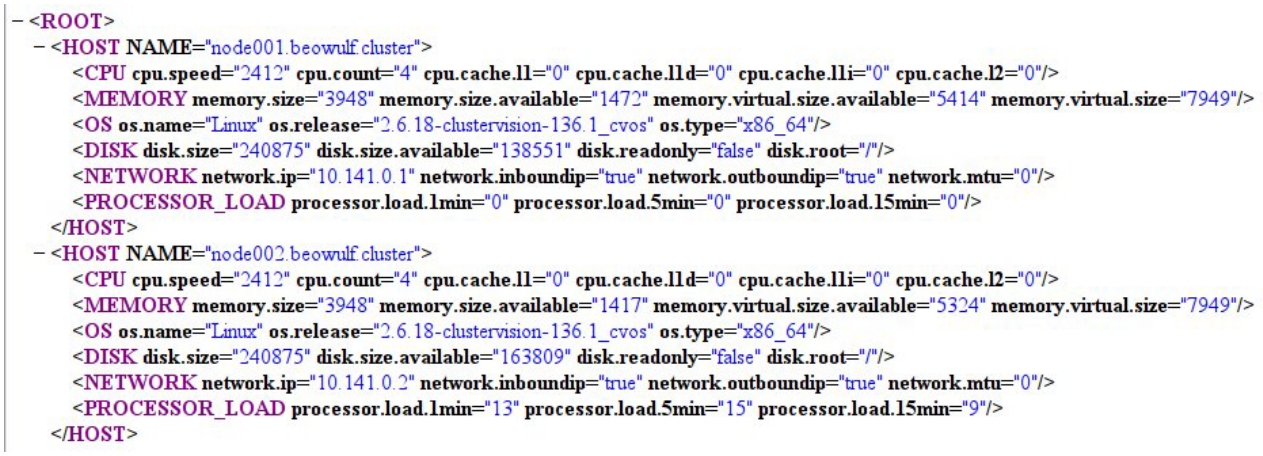

Figure 2

From the figure 2 is possible to see that the XML document memorized inside javaGAT is easier to read and to manage than the other one.

MatchResources: the aim of this method is to check if the resource received like parameter matches the user desiderated features. It's important to know that a resource matches the requested features "all" off then are matched, because a user can request, for example, a resource with a CPU clock faster than 2000 Mhz and a memory amount more than 4000 MB.

getDocument: it's an utility method that returns the XML document stored in the memory.

getResourceByName: it's an utility method that allows to obtain a resource searching it by his name

From a JDOM element of this class is possible to build a GlobusHardwareResource.

WSGT4newResourceBrokerAdaptor

This class has been modified, in fact there are some new methods. The most important new implemented method is *findResources*. The aim of this method is to find all the resources that match the ResorceDescription passed like parameter and return them as a list of HardwareResource.

This method first checks if there is an XML document stored in the memory, if there is not an XML document stored then it invokes the method *queryIndexService*.

Moreover, the method checks if the resource attributes of the resource description are all static. A static attribute is a one that can never change while an host is connected to its network. Examples of static attributes are the CPU speed or the disk size. So, if the method noticed that all the resource attributes of the ResourceDescription passed like parameter in a *findResource* invocation are static, and there is already an XML document stored in the memory, then it doesn't execute another query to the Index Service but it just retrieves the values stored in the XML document because it is sure

that the values of the resource attributes haven't changed.

One of the most important new method created is *queryDefaultIndexService (see appendix for details)*. Is goal is to query the Index Service from source code using an Xpath query. Most of the code is template but to connect to the DAS3 we must have a valid certificate. For to know how have a valid certificate take a look to the appendix; to know how fix this problem is very important, especially if you want to connect to another cluster on the port 8443.

QueryingThread

The aim of this thread is to make periodically query to the Default Index Service using the *queryIndexService* method provided by the ResourceBroker. Once obtained the result, using the *updateXML* method provided by the resource broker also, it updates the xml document stored in the memory. After this it goes to sleep for almost 5 minutes. The sleeping time is not exactly 5 minutes because the notifier threads wants fresh informations when they are awake, and if they sleep for the same time of the QueryingThread, they will retrieve the old informations because the query to the Index Service needs some seconds to retrieve the result.

This class has also a field variable called numberOfListeners that counts the number of Notifier threads listening. If this number after a removeMetricListener invocation reaches 0, the querying thread is killed because it becomes useless.

NotifierThread

The aim of this thread is to keep listening on a resource for some metrics and check if it's the case to fire an event or not, after have checked the values of the XML document stored in the memory (and updated every 5 minutes by the QueryingThread). It has a list of *org.gridlab.gat.monitoring.MetricListener* and *org.gridlab.gat.monitoring.Listener* like field variables because one NotifierThread can listen on several metrics. In addition, it has also an array of long called updates that stores the updating time of every metric. Retrieving that values is possible to decide the sleeping time of every metric added. The thread is also able to kill himself if the list of listeners becomes empty after a remove.

The more interesting method in this class is the *checkKeys* method. This is the method that decides if to fire or not a metric. Its implementation is really simple: the user can choose to have notification about a metric if its value is *lessThan* a value, *moreThan* a value or both *lessThen* and *moreThan* a value. For example the user can choose to be receive an event when the memory available on an host is less than 1500 MB, or to e notified when the memory available is more than 1600 but less than 2000 MB.

The implementation of these class required that the methods add, remove and a part of the run

should be managed by a lock to avoid concurrency problems.

GlobusHardwareResource

This class extends *org.gridlab.gat.resources.cpi.HardwareResourceCpi*. A *GlobusHardwareResource* is created from a JDOM Element. Its most important methods are:

createReturnDefinitions: this method creates the return definition that are needed for to implement the metric listeners.

addMetricListeners: this method starts the *QueryingThread* if it is not started already, it creates and starts the *NotifierThread* on the resource and finally it updates the number of threads that are listening on the *QueryingThread*.

removeMetricListeners: removes the MetricListeners from the *NotifierThread* and updates the number of threads that are listening on the *QueryingThread*. If there are no threads listening on the *QueryingThread* then it is killed.

Execution Example

The class *QueryIndexService* contains the main that allows to test the new classes. It contains an invocation to the method findResources that receives the desired HardwareResourceDescription. Like said before, the method returns a list of HardwareResource and for each of them is possible to add one or more metric with a frequency that must be a multiple of 5 minutes. After that the main thread goes to sleep for a little more than 15 minutes (for to give the time to the Notifier Threads to retrieve the informations from the memory for 3 times) and when he wakes up it removes the metric previously added. Now that the NotifierThreads are with no metrics they will die like the *QueryingThread* that will not have NotifierThreads listening on him anymore. After this he will sleep for few second and it is gonna die.

Appendix

SSLHandshakeException

If we don't have a valid SSL certificate we will have this exception:

javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderExc eption: unable to find valid certification path to requested target

This simply means that the web server or the URL you are connecting to does not have a valid certificate from an authorized CA.

What it needs to do is to import the server certificate and install it in the JDK's keystore. The steps to get rid of that error are:

1. First of all we need to copy the URL that you are connecting to and paste it in a browser. Internet Explorer is the best one to do these steps. So, paste the url in the address bar and press enter.
2. A dialog box warning you about the certificate will appear. Now click on the 'View Certificate' and install the certificate. Ignore any warning messages.
3. Now that the server certificate is installed in the computer, the browser will not show a warning when the same site is visited again. But however the JRE does not know about this certificate's existence yet until it is added it to its keystore.
4. Now we have to add the previously installed certificate to the keystore. To add, begin by exporting the CA Root certificate as a DER-encoded binary file and save it as **C:\root.cer**. (it is possible to see the installed certificates under *Tools->'Internet Options' ->Content->Certificates*. Once opened the certificates, locate the needed one under 'Trusted Root Certification Authorities". Select the right one and click on 'export'. It can be saved (DER encoded binary) under your c: drive.
5. Then whit the *keytool -import* command it is possible to import the file into the cacerts keystore. *For example:-alias myprivateroot -keystore ..\lib\security\cacerts -file c:\root.cer*
6. Now run *keytool -list* again to verify that the new private root certificate was added: *C:\Program Files\Citrix\Citrix Extranet Server\SGJC\jre\bin>keytool -list -keystore ..\lib\security\cacerts* You will now see a list of all the certificates including the one just added.

Inside the javaGAT source code it has been necessary to set the System property “javax.net.ssl.trustStore” to the new cacert file, and the value of “javax.net.ssl.trustStorePassword” to a standard string.

QueryDefaultIndexService method

The query to the Index Service returns an array of 3 MessageElement. The first 2 are the same but the 3th one has some little differences in the values (maybe it has the older values). It has been decided to took the first one because it has been used a voting approach (2 of them have the same values so it can be possible that they have the correct values).