Project no. FP6-034442

GridCOMP

Grid programming with COMPonents : an advanced component platform for an effective invisible grid

STREP Project

Advanced Grid Technologies, Systems and Services

D.GIDE.02 – Grid IDE Early Prototype

Due date of deliverable: 31st May, 2007

Actual submission date: 3rd July, 2007

Start date of project: 1 June 2006 **Duration**: 30 months

Organisation name of lead contractor for this deliverable: University of Westminster (UoW)

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU

Keyword List: Grid IDE, Software Development for Grids, Component-oriented applications, Grid Deployment and Monitoring, Runtime Steering
Responsible Partner: Vladimir Getov, UoW

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
0	15-06-2007	Draft 1	Artie Basukoski, Jeyan Thiyagalingam
1	31-06-2007	Draft 2	Artie Basukoski, Jeyan Thiyagalingam

Deliverable manager

- Vladimir Getov, UoW

List of Contributors

- Artie Basukoski, UoW
- Jeyan Thiyagalingam, UoW
- Yuan Yulai, TU
- Yongwei Wu, TU

List of Evaluators

- Thomas Weigold, IBM
- Jose Piquer, UNI CHILE

Summary

This document describes the overall aims and functionality of the Grid IDE from the perspective of two different communities: developers and data centre operators. First the document describes the overall integrated platform, and then it focuses on different aspects that the Grid IDE platform supports such as composition, monitoring, deployment, and steering. The document also points out the limitations of the current version.

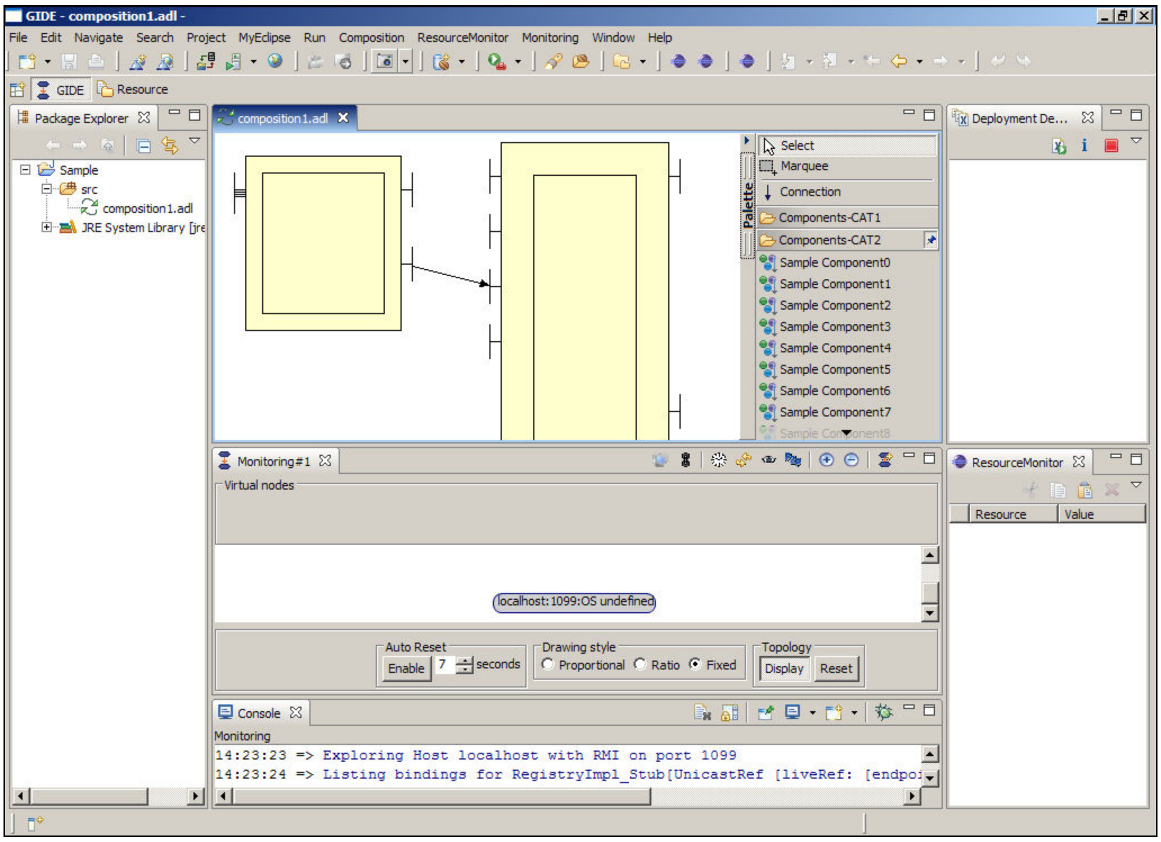
Table of Contents

1 Introduction

The GIDE provides an integrated environment to support both the software development process and the data centre operations. The GIDE is a plugin for Eclipse and this deliverable report assumes you, the user, are competent in using the Eclipse platform, configuring the platform for additional features and plugins. Full installation instructions are given in Appendix A. Additional information for managing plugins can be found in the Eclipse User Manual.

Figure 1 below shows the potential perspective for the integrated environment consisting of a number of views. The final version of the deliverable will consist of four perspectives as follows:

- Composition Perspective to enable the composition and application development.
- Deployment Perspective for deploying applications, components and for launching and stopping components.
- Resource Monitoring Perspective for monitoring resources including hosts and nodes
- Monitoring/Steering Perspective which is an integrated IC2D plugin



```
14:23:23 => Exploring Host localhost with RMI on port 1099
14:23:24 => Listing bindings for RegistryImpl_Stub[UnicastRef [liveRef: [endpoi
```

Figure 1: The Integrated Development Environment for Grid

Although the final version of the software will incorporate all requested features, this prototype version is limited in the scope of its functionalities. These limitations include :

- Dynamic re-configuration of the component gallery or palette when new components are added or deleted
- Support for composite components within the GIDE composition
- Appropriate and correct rendering of imported ADL files
- Support for generation of ADL files for compositions
- Full implementation of the deployment and resource monitor views
- Finalized implementation of perspectives within Eclipse

The following sub-sections highlight the means for using the features implemented in this version of prototype.

2 Composition

In a component-oriented development environment, applications are made up of several components. One potential advantage of using the GIDE for that process is the support for visual interaction for developing applications.

To begin with the process

- Launch eclipse and activate the supplied plugin. Restart Eclipse if necessary. This step will ensure that when you start Eclipse next time, it automatically loads the plugin and sets up the environment for you.

- Once activated, create a new project. The plugin supports the composition process within an existing project.

Figure 2 shows the overall user interface for the GIDE Composition Editor.

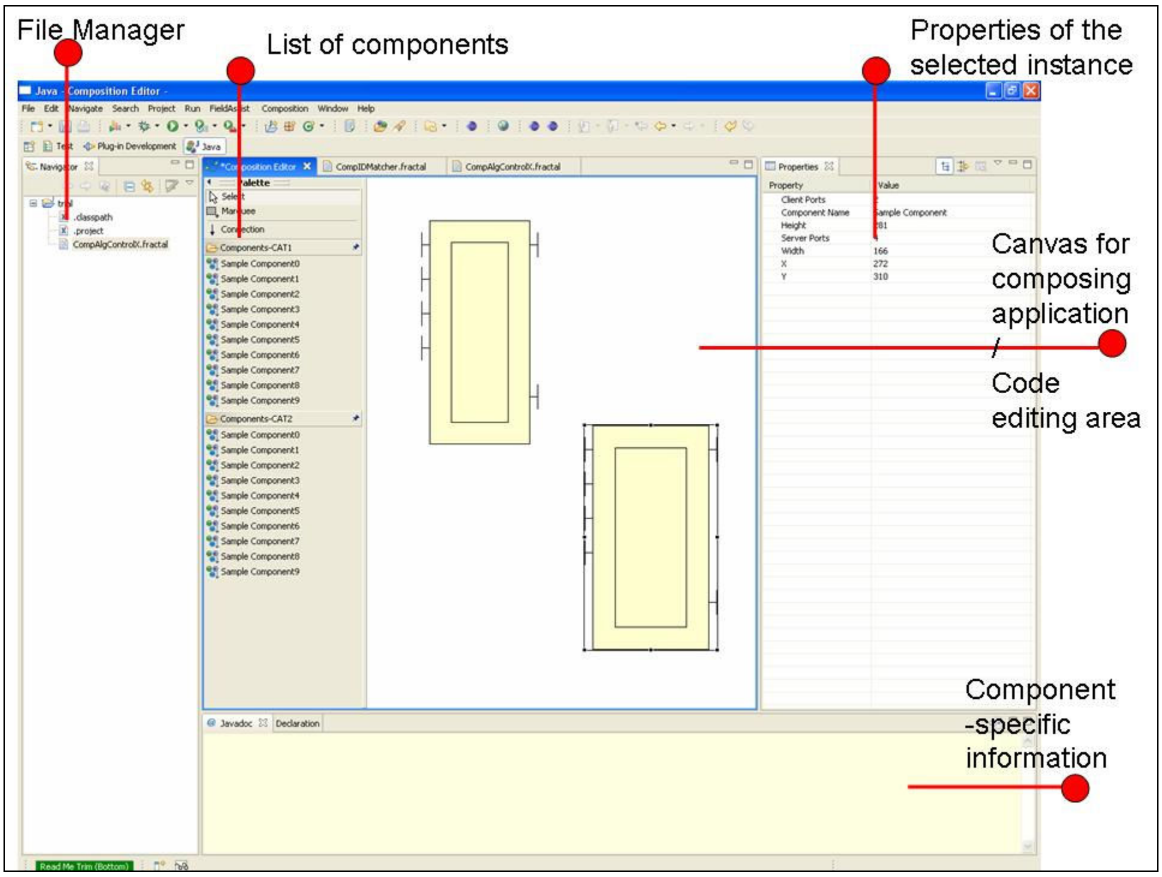

File Manager
List of components
Properties of the selected instance
Canvas for composing application / Code editing area
Component -specific information

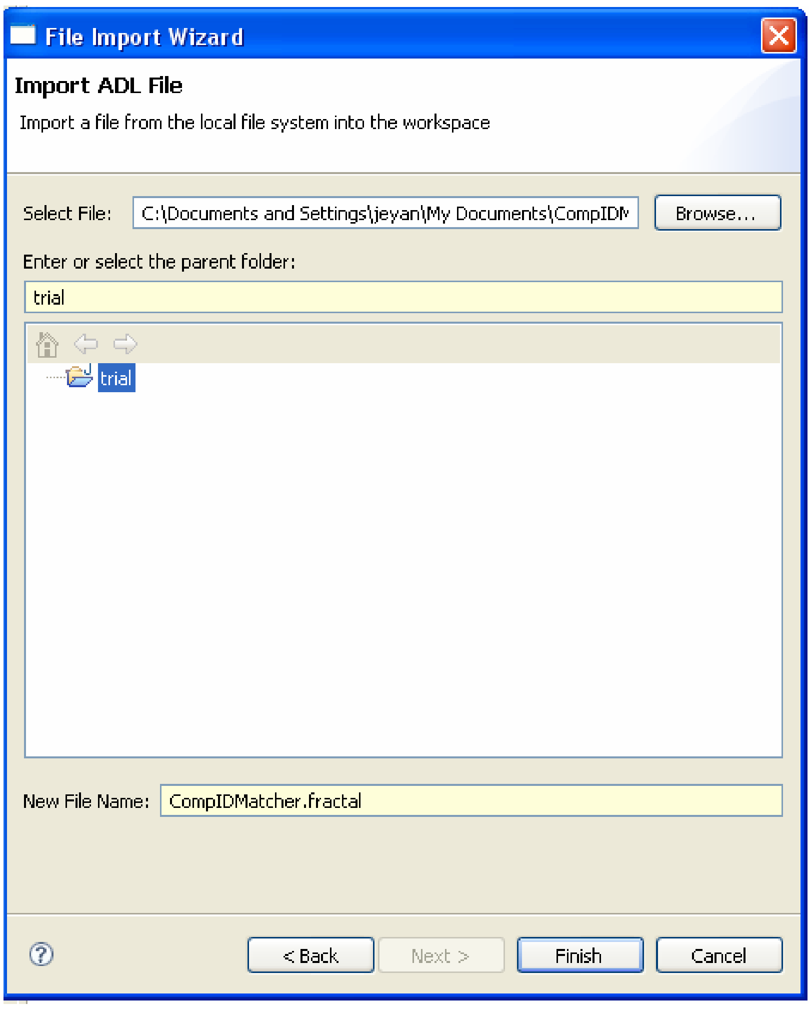
Figure 2 User Interface of the Composition Editor

When composing an application, users can drag and drop different components from the components gallery. Their properties could be changed using the properties tab. As they are dragged and dropped, necessary skeleton codes are generated in the back-end and they can be viewed or edited as necessary using the Solution/File Explorer. Additional information about each component is available in the metadata tab.

From time to time, it may be necessary to import ADL files into an existing project and then to modify the imported file. To import an existing ADL file

- Select File -> Import -> ADL File Import -> Import ADL File

Navigate to the file you want to import and provide a name for the imported file. The default name can be left intact. Figure 3 depicts this scenario.


File Import Wizard

Import ADL File

Import a file from the local file system into the workspace

Select File: C:\Documents and Settings\jeyan\My Documents\CompIDM

Enter or select the parent folder: trial

New File Name: CompIDMatcher.fractal

Figure 3: Importing an ADL file into an existing project

Once imported, the file will appear as part of the solution and can either be edited or rendered on screen for further editing through the context menu. This is shown in Figure 4 below

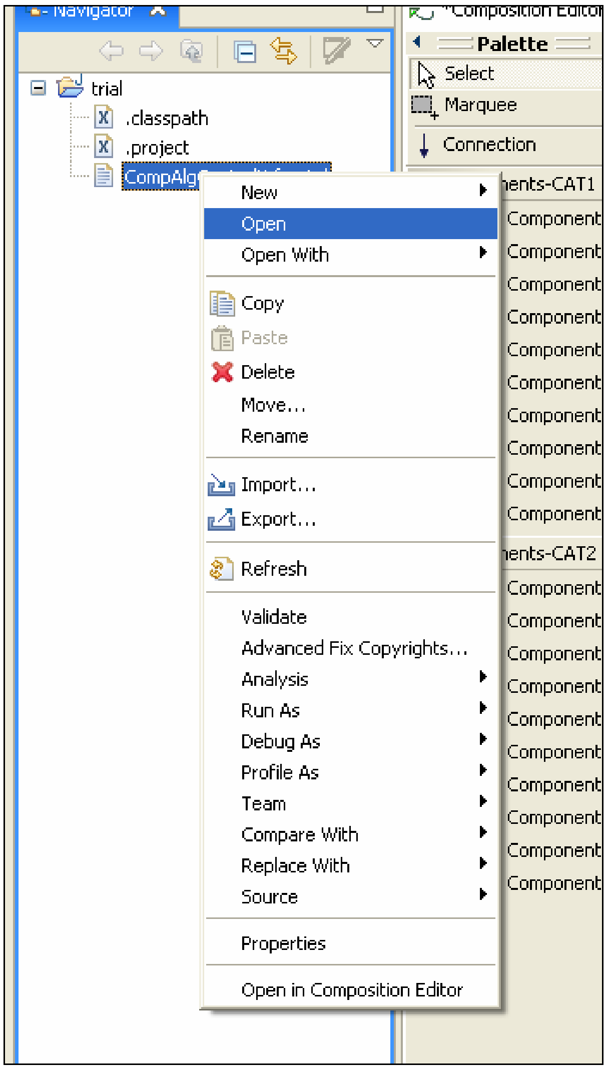
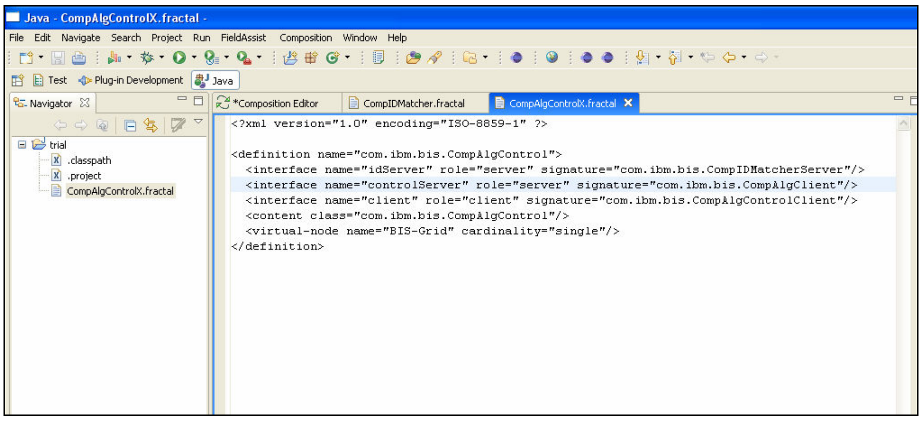


Figure 4: ADL File as part of a solution and associated operations through context menu

Selecting "Open" will open the ADL file in an editor for manual editing and selecting "Open in Composition Editor" will render the file for visual editing and this is shown in Figure 4.



```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<definition name="com.ibm.bis.CompAlgControl">
  <interface name="idServer" role="server" signature="com.ibm.bis.CompIDMatcherServer"/>
  <interface name="controlServer" role="server" signature="com.ibm.bis.CompAlgClient"/>
  <interface name="client" role="client" signature="com.ibm.bis.CompAlgControlClient"/>
  <content class="com.ibm.bis.CompAlgControl"/>
  <virtual-node name="BIS-Grid" cardinality="single"/>
</definition>
```

Figure 5: ADL File within an Editor

The IDE essentially has three different interconnected components viz. ADL Parser/Verifier, ADL Renderer and ADL/Source Code Generator: For full internal functionalities of the composition part which we explained in this Section, please consult the technical report.

3 Resource Monitoring

The Host and Resource monitor views will provide a list of all hosts, and enable the user to "zoom-in" to a particular host to monitor its resource utilization. The resource utilization is provided in the Resource Monitor View which is partly implemented and shown as a view in Figure 1. When complete it will provide a table view of CPU utilization, hard disk usage, threads and so on.

To start the resource monitor view

- Within eclipse, select Window → Show View → Other → Resource Monitor.

4 Monitoring and Steering

Monitoring and steering is currently supported by the IC2D plugin at the active object level. Given this functionality, the current version can be used to monitor a component which acts as a representative for an active object. Future versions of the GIDE will also support monitoring at the component level. The GIDE dynamically ensures that the plugged in IC2D offers all features as the standard IC2D version. Please consult the IC2D manual for detailed information.

5 Conclusions

This document has provided a quick overview of how to use the system from the perspective of two different user communities: developers and data centre operators. The full source code can be found in the following location:

http://perun.hscs.wmin.ac.uk/GridComp/D.GIDE.02-bundle03.zip

Appendix A

GIDE – Installation

Installing GIDE is fairly a straightforward process. However, following are the prerequisites for installing the system:

1. A computer system capable of running Windows XP or Linux
2. Eclipse Framework Version 3.2
3. Java Development Kit 1.5
4. ProActive Library 3.2

To begin the process of installation:

1. Download the code base from the location http://perun.hscs.wmin.ac.uk/GridComp/D.GIDE.02-bundle03.zip
2. Create a new directory (for example C:\GIDE, if the target platform is Windows or ./GIDE if Linux) and copy the downloaded ZIP file within that directory.
3. Create a new workspace in Eclipse under the working directory (C:\GIDE or ./GIDE). The location of the directory is optional.
4. Import the copied archive file into the workspace by using the "Import Existing Project Archive Wizard".
5. The installation includes necessary libraries referred by the project. However, the CLASSPATH and associated variables may need to be modified to suit your environment.

This concludes the installation.

To create a sample composition,

1. Restart Eclipse
2. Create a new Java Project
3. Select File -> New -> Example -> Composition. *This should provide you with a composition wizard.*
4. See Figure 3 for the illustration of completing the wizard pages
5. Once completed, this should open the composition perspective.
6. To create a composition, drag and drop components from the gallery.
7. Save and open the files to check.

One important issue here is that, in the current version, compositions are saved with the ADL extension and this will be fixed in the next release.

Appendix B

Node Resource Monitor Presentation & Specification

B.1 Introduction

This document is an assessed definition for a Node Resource Monitor which enables the monitoring of resource information of nodes. Its aim is to provide an independent tool that will be deployed on the nodes and report real-time resource information, CPU (total, used, idle), memory (total, used, free), disk (total, used, free), cache, system clock, OS release, etc. In order to fulfill those requirements, we first propose the platforms and metrics that this Node Monitor Component will cover, and then we specify the API and the XML & deployment process.

B.2 Platform

Node monitor component runs on Linux (i386, ia64, alpha, etc. to be added), FreeBSD.

B.3 Metrics

Metric Name	**Description**	**Unit**
cpu_speed	Speed of CPU	MHz
cpu_num	Number of CPUs or Cores	
cpu_idle	Percent CPU idle	%
cpu_system	Percent CPU system	%
cpu_user	Percent CPU user	%
load_fifteen	Fifteen minute load average	
load_five	Five minute load average	
load_one	One minute load average	
mem_total	Total amount of memory	kB
mem_free	Amount of available memory	kB
mem_buffers	Amount of buffered memory	kB
mem_cached	Amount of cached memory	kB
disk_total	Amount of total disk	kB
disk_free	Amount of available disk	kB
swap_total	Total amount of swap memory	kB
swap_free	Amount of available swap memory	kB
proc_total	Total number of processes	
proc_run	Total number of running processes	
sys_clock	Current time on host	ms
os_name	Operating system name	
os_release	Operating system release (version)	
boot_time	Boot time of the system	minute

B.4 API

The following API function calls can be used to get the Node Resource Information. There are two ways to get the resource information, one is to get the whole resource information once at a time, using the function `public NodeResourceInfo getNodeResource();`, and then get the resource information from

NodeResourceInfo. The other way is to get the resource information separately and directly, using the functions `public float getCpuSpeed(); public int getCpuNum();`, etc.

```
public class NodeMonitorClass {
   public                    NodeResourceInfo
getNodeResourceInfo();
      public float getCpuSpeed();
      public int getCpuNum();
      public float getCpuIdle();
      public float getCpuSystem();
      public float getCpuUser();
      public float getLoadFifteen();
      public float getLoadFive();
      public float getLoadOne();
      public int getMemTotal();
      public int getMemFree();
      public int getMemBuffers();
      public int getMemCached();
      public int getMemShared();
      public int getDiskTotal();
      public int getDiskFree();
      public int getSwapTotal();
      public int getSwapFree();
      public int getProcTotal();
      public int getProcRun();
      public long getSysClock();
      public String getOsName();
      public String getOsRelease();
    public long getBootTime();
}
```

```
public class NodeResourceInfo {
      public float getCpuSpeed();
      public int getCpuNum();
      public float getCpuIdle();
      public float getCpuSystem();
      public float getCpuUser();
      public float getLoadFifteen();
      public float getLoadFive();
      public float getLoadOne();
      public int getMemTotal();
      public int getMemFree();
      public int getMemBuffers();
      public int getMemCached();
      public int getMemShared();
      public int getDiskTotal();
      public int getDiskFree();
      public int getSwapTotal();
      public int getSwapFree();
      public int getProcTotal();
      public int getProcRun();
      public long getSysClock();
      public String getOsName();
      public String getOsRelease();
    public long getBootTime();
}
```

B. 5 Deployment & XML

The Virtual Node & Node mapping XML files are defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProActiveDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DescriptorSchema.xsd">
   <componentDefinition>
      <virtualNodesDefinition>
         <virtualNode name="testNode"/>
      </virtualNodesDefinition>
   </componentDefinition>
   <deployment>
      <mapping>
         <map virtualNode="testNode">
            <jvmSet>
               <vmName value="ssh_Jvm0"/>
            </jvmSet>
            <jvmSet>
               <vmName value="ssh_Jvm1"/>
            </jvmSet>
            <jvmSet>
               <vmName value="ssh_Jvm2"/>
            </jvmSet>
         </map>
      </mapping>
      <jvms>
         <jvm name="ssh_Jvm0">
            <creation>
               <processReference refid="sshProcess0"/>
            </creation>
```

```
          </jvm>
          <jvm name="ssh_Jvm1">
             <creation>
                <processReference refid="sshProcess1"/>
             </creation>
          </jvm>
          <jvm name="ssh_Jvm2">
             <creation>
                <processReference refid="sshProcess2"/>
             </creation>
          </jvm>
       </jvms>
    </deployment>
    <infrastructure>
       <processes>
             <!-- ssh_Jvm0 and ssh_Jvm1 has the same classpath, javapath ,etc -->
             <processDefinition id="localJVM0">
             <jvmProcess
class="org.objectweb.proactive.core.process.JVMNodeProcess">
             <!-- We might need to redefine classpathand other variables if files
are not shared -->
                <classpath>
                   <absolutePath value="/usr/ProActive/ProActive.jar"/>
                   ...............................................
                   <absolutePath value="/usr/ProActive/lib/jsch.jar"/>
                </classpath>
                <javaPath>
                   <absolutePath value="/usr/java/jdk1.5.0_08/bin/java"/>
                </javaPath>
                <policyFile>
                   <absolutePath
value="/usr/ProActive/scripts/proactive.java.policy"/>
                </policyFile>
                <log4jpropertiesFile>
                   <absolutePath value="/usr/ProActive/scripts/proactive-log4j"/>
                </log4jpropertiesFile>
                <jvmParameters>
                          <parameter value="-
Dproactive.communication.protocol=rmissh"/>
                     </jvmParameters>
             </jvmProcess>
             </processDefinition>

             <!-- ssh_Jvm2 has the classpath, javapath ,etc that are different from
ssh_Jvm0 and ssh_Jvm1-->
             <processDefinition id="localJVM1">
             <jvmProcess
class="org.objectweb.proactive.core.process.JVMNodeProcess">
             <!-- We might need to redefine classpathand other variables if files
are not shared -->
                <classpath>
                   <absolutePath value="/usr/Work/ProActive/ProActive.jar"/>
                   ...............................................
                   <absolutePath value="/usr/Work/ProActive/lib/jsch.jar"/>
                </classpath>
                <javaPath>
                   <absolutePath value="/usr/Work/java/jdk1.5.0_08/bin/java"/>
                </javaPath>
                <policyFile>
                   <absolutePath
value="/usr/Work/ProActive/scripts/proactive.java.policy"/>
                </policyFile>
                <log4jpropertiesFile>
                   <absolutePath value="/usr/Work/ProActive/scripts/proactive-
log4j"/>
                </log4jpropertiesFile>
                <jvmParameters>
                          <parameter value="-
```

```
Dproactive.communication.protocol=rmissh"/>
                    </jvmParameters>
            </jvmProcess>
            </processDefinition>

      <!-- Example to show how to use ssh. This process reference the one above, it
                means that once log on the remote machine with ssh the java process
                is started. If files are not shared, the local process might need to
                redefine some variables.-->
         <processDefinition id="sshProcess0">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.32" username="root">
               <processReference refid="localJVM0"/>
            </sshProcess>
         </processDefinition>

         <processDefinition id="sshProcess1">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.33" username="root">
               <processReference refid="localJVM0"/>
            </sshProcess>
         </processDefinition>

         <processDefinition id="sshProcess2">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.34" username="root">
               <processReference refid="localJVM1"/>
            </sshProcess>
         </processDefinition>

      </processes>
   </infrastructure>
</ProActiveDescriptor>
```

The Node Monitor Component Deployment process and how to call the functions are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProActiveDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DescriptorSchema.xsd">
   <componentDefinition>
      <virtualNodesDefinition>
         <virtualNode name="testNode"/>
      </virtualNodesDefinition>
   </componentDefinition>
   <deployment>
      <mapping>
         <map virtualNode="testNode">
            <jvmSet>
               <vmName value="ssh_Jvm0"/>
            </jvmSet>
            <jvmSet>
               <vmName value="ssh_Jvm1"/>
            </jvmSet>
            <jvmSet>
               <vmName value="ssh_Jvm2"/>
            </jvmSet>
         </map>
      </mapping>
      <jvms>
         <jvm name="ssh_Jvm0">
            <creation>
               <processReference refid="sshProcess0"/>
            </creation>
         </jvm>
```

```
          <jvm name="ssh_Jvm1">
             <creation>
                <processReference refid="sshProcess1"/>
             </creation>
          </jvm>
          <jvm name="ssh_Jvm2">
             <creation>
                <processReference refid="sshProcess2"/>
             </creation>
          </jvm>
       </jvms>
    </deployment>
    <infrastructure>
       <processes>
             <!-- ssh_Jvm0 and ssh_Jvm1 has the same classpath, javapath ,etc -->
             <processDefinition id="localJVM0">
             <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess">
             <!-- We might need to redefine classpathand other variables if files are
not shared -->
                <classpath>
                   <absolutePath value="/usr/ProActive/ProActive.jar"/>
                   ...............................................
                   <absolutePath value="/usr/ProActive/lib/jsch.jar"/>
                </classpath>
                <javaPath>
                   <absolutePath value="/usr/java/jdk1.5.0_08/bin/java"/>
                </javaPath>
                <policyFile>
                   <absolutePath
value="/usr/ProActive/scripts/proactive.java.policy"/>
                </policyFile>
                <log4jpropertiesFile>
                   <absolutePath value="/usr/ProActive/scripts/proactive-log4j"/>
                </log4jpropertiesFile>
                <jvmParameters>
                          <parameter value="-
Dproactive.communication.protocol=rmissh"/>
                     </jvmParameters>
             </jvmProcess>
             </processDefinition>

             <!-- ssh_Jvm2 has the classpath, javapath ,etc that are different from
ssh_Jvm0 and ssh_Jvm1-->
             <processDefinition id="localJVM1">
             <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess">
             <!-- We might need to redefine classpathand other variables if files are
not shared -->
                <classpath>
                   <absolutePath value="/usr/Work/ProActive/ProActive.jar"/>
                   ...............................................
                   <absolutePath value="/usr/Work/ProActive/lib/jsch.jar"/>
                </classpath>
                <javaPath>
                   <absolutePath value="/usr/Work/java/jdk1.5.0_08/bin/java"/>
                </javaPath>
                <policyFile>
                   <absolutePath
value="/usr/Work/ProActive/scripts/proactive.java.policy"/>
                </policyFile>
                <log4jpropertiesFile>
                   <absolutePath value="/usr/Work/ProActive/scripts/proactive-
log4j"/>
                </log4jpropertiesFile>
                <jvmParameters>
                          <parameter value="-
Dproactive.communication.protocol=rmissh"/>
                     </jvmParameters>
             </jvmProcess>
```

```
            </processDefinition>

      <!-- Example to show how to use ssh. This process reference the one above, it
                means that once log on the remote machine with ssh the java process
                is started. If files are not shared, the local process might need to
                redefine some variables.-->
         <processDefinition id="sshProcess0">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.32" username="root">
               <processReference refid="localJVM0"/>
            </sshProcess>
         </processDefinition>

         <processDefinition id="sshProcess1">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.33" username="root">
               <processReference refid="localJVM0"/>
            </sshProcess>
         </processDefinition>

         <processDefinition id="sshProcess2">
            <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
hostname="10.0.1.34" username="root">
               <processReference refid="localJVM1"/>
            </sshProcess>
         </processDefinition>

      </processes>
   </infrastructure>
</ProActiveDescriptor>
```