University College London
MSc Data Communications, Networks & Distributed Systems

Deployment in Computational Distributed Grids
Developer Section

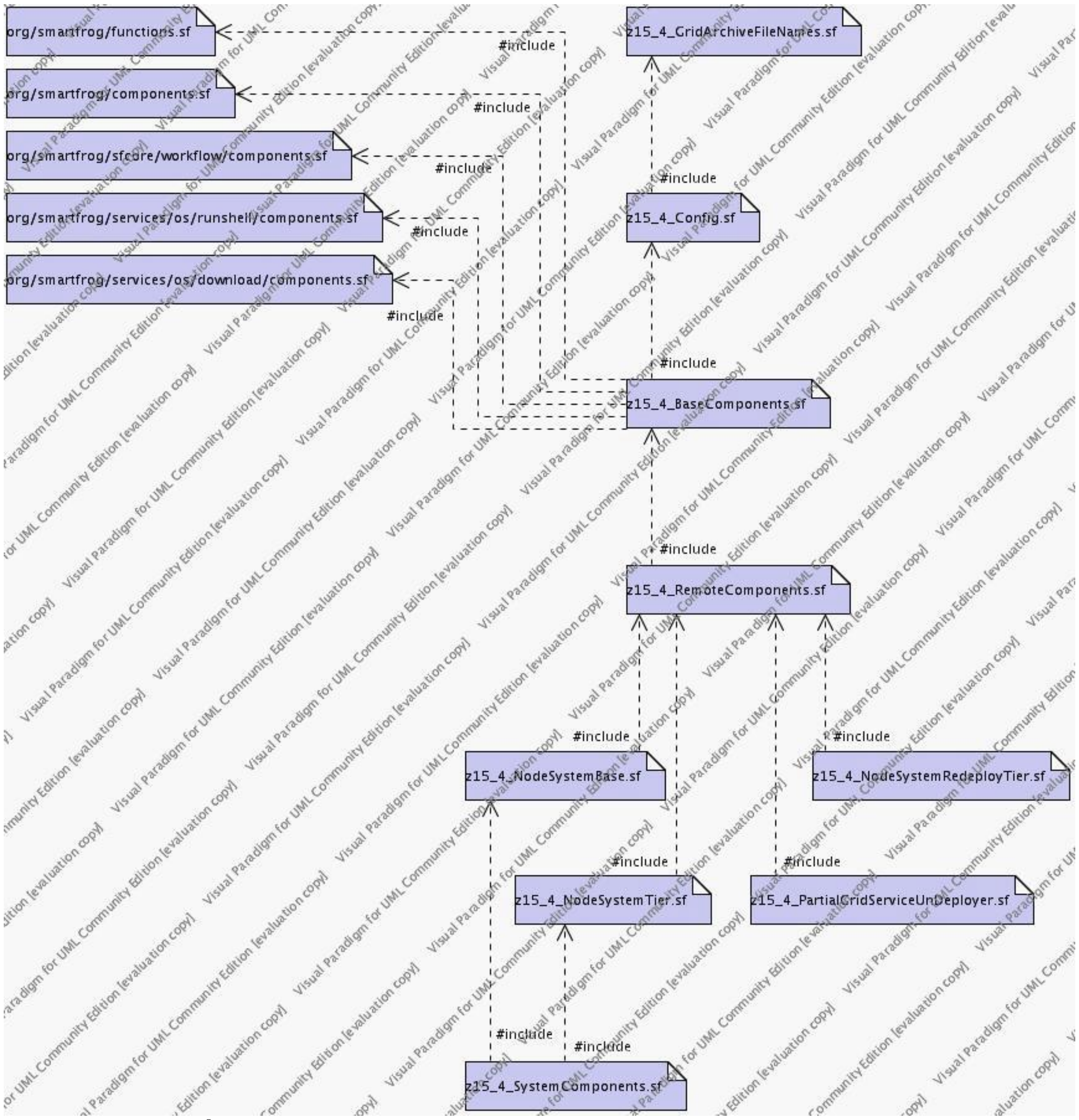
supervisor
Wolfgang Emmerich, Ph. D.

group members

Daisy Kong
Vesso Novov
Dimitrios Tsalikis
Stefanos Koukoulas
Thomas Karampaxoglou

06 September 2004

1. SmartFrog Components


org/smartfrog/functions.sf
org/smartfrog/components.sf
org/smartfrog/sfcore/workflow/components.sf
org/smartfrog/services/os/runshell/components.sf
org/smartfrog/services/os/download/components.sf
z15_4_GridArchiveFileNames.sf
z15_4_Config.sf
z15_4_BaseComponents.sf
z15_4_RemoteComponents.sf
z15_4_NodeSystemBase.sf
z15_4_NodeSystemRedeployTier.sf
z15_4_NodeSystemTier.sf
z15_4_PartialGridServiceUnDeployer.sf
z15_4_SystemComponents.sf
#include
Visual Paradigm for UML Community Edition [evaluation copy]

FileSystemDiag-01

The above *FileSystemDiag-01* illustrates the interdependence of the deployment application and SmartFrog-provided files containing the definitions of all components used. For details of components in the following files refer to the SmartFrog documentation:

org/smartfrog/functions.sf
org/smartfrog/components.sf
org/smartfrog/sfcore/workflow/components.sf
org/smartfrog/services/os/runshell/components.sf
org/smartfrog/services/os/download/components.sf

The following files contain project specific components and their description:

z15_4_GridArchiveFileNames.sf
z15_4_Config.sf
z15_4_BaseComponents.sf
z15_4_RemoteComponents.sf
z15_4_NodeSystemBase.sf
z15_4_NodeSystemTier.sf
z15_4_SystemComponents.sf
z15_4_NodeSystemRedeployTier.sf
z15_4_PartialGridServiceUnDeployer.sf
z15_4_FullInstallationUnDeployer.sf

For specific details of these components and files not included in the descriptions below, refer to the source code section of this report.

file: *z15_4GridArchiveFileNames.sf*

This file contains some basic configuration parameters and their values. These parameters are kept in this file separate from the parameters in the configuration file *z15_4_Config.sf* as this one is read and overwritten during run-time, i.e. - the values are changed dynamically by certain application components.

baseGarUrlAddress
baseGarFileName
baseGarId
sfProcessHost

The values of these parameters specify the URL address to the Grid Service Archive (GAR) file for the Grid Service to be deployed, the name of this file (including '.gar' extension), the ID of this file (file name excluding '.gar' extension) and the name/IP of the host where the deployment is to take place.

file: *z15_4_Config.sf*

This file contains the basic configuration parameters and their values
accessed by individual components in the application's hierarchy. Unlike the parameters in the previous file these values are predefined and static for the duration of the application execution.

baseHost
baseHostWebSrvr

These parameters define the attribute name(s) and the value(s) for the Web server hosting required installation, configuration, description etc. files and resources.

baseDir

baseProjectDirName
baseProjectDir

These parameters define the name and the location of the project directory where all installation files and resources are down-loaded and stored locally on each deployment host.

baseProjectGarDirName
baseProjectGarDir

These parameters define the name of the subdirectory of the above defined project directory where grid GAR files are down-loaded and stored locally on each deployment host.

baseRetryCounter

These parameters define the number of times SmartFrog Retry components attempt to execute their action sub-component. The parameter is used primarily by components downloading files from the Internet, thus accounting for possible cross-network file download problems.

gzTarFileName
gzTarUrlAddress
tarFileName
tarDirName

These parameters define the name and 'from' location the GNU Tar installation files are to be copied and the name and 'to' location to where they are to be down-loaded and installed. The 'to' location is a subdirectory of the project directory defined above.

gzAntFileName
gzAntUrlAddress
antDirName

These parameters define the name and 'from' location the Apache Ant installation files are to be copied and the name and 'to' location to where they are to be down-loaded and installed. The 'to' location is a subdirectory of the project directory defined above.

gzTomcatFileName
gzTomcatUrlAddress
tomcatDirName
tomcatUsersFileName

These parameters define the name and 'from' location the Tomcat installation files are to be copied and the name and 'to' location to where they are to be down-loaded and installed. The 'to' location is a subdirectory of the project directory defined above.

gzTomcatDeployerFileName
gzTomcatDeployerUrlAddress
tomcatDeployerDirName

These parameters define the name and 'from' location the Tomcat Deployer Package installation files are to be copied and the name and 'to' location to where they are to be down-loaded and installed. The 'to' location is a subdirectory of the project directory defined above.

gzOgsaFileName
gzOgsaUrlAddress
ogsaDirName
ogsaWebAppName

These parameters define the name and 'from' location the Open Grid Services Architecture (OGSA) Globus Toolkit version 3.2 installation files are to be copied and the name and 'to' location to where they are to be down-loaded and installed. The 'to' location is a subdirectory of the project directory defined above.

file: *z15_4_BaseComponents.sf*

This file contains the base application component descriptions. The components behind these descriptions provide basic functionality which is inherited and enhanced by subsequent components in the application hierarchy.

component: *z15_4BashRunCommand extends RunCommand {}*

This component inherits its basic functionality for executing Unix commands/shell scripts from SmartFrog-provided *RunCommand* component. It re-defines an inherited attribute specifying the shell interpreter - 'shell', 'bash', 'korn', 'c-shell' etc., to be used for all executions.

component: *z15_4BashUnTar extends Try {}*

This component inherits its basic functionality from the SmartFrog-provided *Try* component. Its 'action' sub-component inherits basic functionality for executing Unix commands and defines specific functionality to execute 'tar' command with options on a given compressed archive file.

component: *z15_4BaseDeployer extends Try {}*

This component inherits its basic functionality from the SmartFrog-provided *Try* component. It defines the basic down-load-install steps for compressed files/resources. Its 'action' sub-component is a sequence of components providing functionality for down-loading, decompressing, installing files. After successful execution of this sequence the original compressed file is deleted.

component: *z15_4BaseInstallationUnDeployer extends Try {}*

This component inherits its basic functionality from the SmartFrog-provided *Try* component. Its 'action' sub-component is a sequence of components, the second of which removes the directory on the deployment host containing all required files, programs and resources. The first action sub-component is left undefined to allow for addition (by being overwritten) of functionality, by inheriting components, before the final deployment directory clean up.

component: *z15_4BaseGarDeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* defined above. This component overwrites some of the inherited attributes and methods to narrow its task to deploy GAR files inside OGSA Globus Toolkit directory structure by using the provided for this purpose 'ant' scripts (included in OGSA GlobusToolkit3.2 installation).

component: *z15_4BaseGarUnDeployer extends z15_4BashRunCommand {}*

This component inherits basic Unix command/shell scripts execution functionality from the z15_4BashRunCommand defined above. This component overwrites some of the inherited attributes and methods to narrow its task to un-deploy GAR files inside OGSA Globus Toolkit directory structure by using the provided for this purpose 'ant' scripts (included in OGSA GlobusToolkit3.2 installation).

file: *z15_4_RemoteComponents.sf*

This file contains the component descriptions for components executed remotely on each deployment host. In addition to inheriting basic attributes and functionality from SmartFrog-provided components, these components are built on the functionality already defined in component descriptions in z15_4_BaseComponents.sf file.

component: *z15_4ProjectDirDeployer extends Try {}*

This component's purpose is to create an installation directory where all files, programs and resources are down-loaded, decompressed and installed. This is done in a sequence of action each executing base components already defined in z15_4_BaseCompoents.sf file. The first sub-action deletes any pre-existing installation directory, the seconds re-creates an empty one and the third creates an empty sub-directory where all GARs are down-loaded.

component: *z15_4GNUTarDeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* component, but overwrites attributes and methods to tailor its task specifically for deploying GNU Tar program. The sequence of steps includes down-loading a compressed install file, decompressing it and running the GNU Tar installation scripts.

component: *z15_4AntDeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* component, but overwrites attributes and methods to tailor its task specifically for deploying Apache Ant program. The sequence of steps includes down-loading a compressed install file, decompressing it using the previously installed GNU Tar program.

component: *z15_4TomcatDeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* component, but overwrites attributes and methods to tailor its task specifically for deploying Tomcat Web Server. The sequence of steps includes down-loading a compressed install file, decompressing it using the previously installed GNU Tar program and overwriting a configuration file with values used later in the deployment process.

component: *z15_4TomcatDeployerDeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* component, but overwrites attributes and methods to tailor its task specifically for deploying Tomcat Deployer package. The sequence of steps includes down-loading a compressed install file, decompressing it using the previously installed GNU Tar program.

component: *z15_4OGSADeployer extends z15_4BaseDeployer {}*

This component inherits basic resource deploying functionality from the *z15_4BaseDeployer* component, but overwrites attributes and methods to tailor its task specifically for deploying OGSA GlobusToolkit3.2. The sequence of steps includes down-loading a compressed install file, decompressing it using the previously installed GNU Tar program and executing 'ant' setup scripts using the previously installed Apache Ant program.

component: *z15_4GridServiceDeployer extends Sequence {}*

This component just defines a sequence of actions, each one deploying a GAR file using the previously defined *z15_4BaseGarDeployer* component. This sequential structure allows for future enhancements by providing the ability to add more *z15_4BaseGarDeployer*-s for sequential deployment of multiple GAR files.

component: *z15_4GridServiceUnDeployer extends Sequence {}*

This component just defines a sequence of actions, each one un-deploying a GAR file using the previously defined *z15_4BaseGarUnDeployer* component. This sequential structure allows for future enhancements by providing the ability to add more *z15_4BaseGarUnDeployer*-s for sequential un-deployment of multiple GAR files.

component: *z15_4WebServicesDeployer extends z15_4BashRunCommand {}*

This component inherits basic Unix command/shell execution functionality from the previously defined *z15_4BashRunCommand* component, but overwrites attributes and methods to make its task specific for deploying OGSA(and deployed Grid Services) directory structure as a Web application under Tomcat Web Server. This is done by using the provided 'ant' scripts included with OGSA GlobusToolkit3.2 installation package. It is assumed that Tomcat Web Server is NOT running so it is started as a final step.

component: *z15_4WebServicesReDeployer extends Sequence {}*

This component inherits basic Unix command/shell execution functionality from the previously defined *z15_4BashRunCommand* component, but overwrites attributes and methods to make its task specific for re-deploying OGSA(and deployed Grid Services) directory structure as a Web application under Tomcat Web Server. This is done by using the provided 'ant' scripts included with OGSA GlobusToolkit3.2 installation package. It is assumed that Tomcat Web Server IS running so this is a 'hot' deployment avoiding the need of shutting down and re-starting the Web Server.

component: *z15_4WebServicesReLoader extends z15_4BashRunCommand {}*

This component inherits basic Unix command/shell execution functionality from the previously defined z15_4BashRunCommand component, but overwrites attributes and methods to make its task specific for re-loading OGSA(and deployed Grid Services) directory structure as a Web application under Tomcat Web Server. This is done by using the provided 'ant' scripts included with Tomcat Deployer installation package. It is assumed that Tomcat Web Server IS running so this is a 'hot' re-loading avoiding the need of shutting down and re-starting the Web Server.

file: *z15_4_NodeSystemBase.sf*

This file contains the description of a component consisting of sequence of actions. The actions are performed on a deployment host by components executing on that host.

component: *z15_4NodeSystemBase extends Sequence {}*

Each action executes a previously defined component making this component just an aggregate one. The sequential execution of each action guaranteed by the inherited SmartFrog-provided components follows a step-by-step deploying pattern in which each step deploys a file/program/resource required for the subsequent steps and expects a file/program/resource is deployed by the previous step. The successful execution of all components of the sequence results in a deployed base of files, programs and resources required for the subsequent deployment of Grid Services. The sequential actions deploy Local Project Directory, GNU Tar, Apache Ant, Tomcat Web Server, Tomcat Deployer Package, and OGSA GlobusToolkit3.2. After the execution of each individual component a status signal is sent to the central management host (the host where the system was started from).

file: *z15_4_NodeSystemTier.sf*

This file contains the description of a component consisting of sequence of actions. The actions are performed on a deployment host by components executing on that host.

component: *z15_4NodeSystemTier extends Sequence {}*

Each action executes a previously defined component making this component just an aggregate one. The sequential execution of each action guaranteed by the inherited SmartFrog-provided components follows a step-by-step deploying pattern in which each step deploys a file/program/resource required for the subsequent steps and expects a file/program/resource is deployed by the previous step. This sequence as a whole expects the component defined in *z15_4_NodeSystemBase.sf* be successfully executed which results in the installation of a base of files, programs and resources required for the successful execution of this sequence. Successful execution of all components of this sequence results in a deployed OGSA directory structure as a new Web Application under Tomcat Web Server. After the execution of each individual component a status signal is sent to the central management host (the host where the system was started from).

file: *z15_4_SystemComponents.sf*

This file contains the description of a component consisting of sequence of actions. The actions are performed on a deployment host by components executing on that host.

component: *z15_4RemoteNodeSystemDeployer extends Sequence {}*

Each action executes a previously defined component making this component just an aggregate one - see *z15_4_NodeSystemBase.sf* and *z15_4_NodeSystemTier.sf*. The sequential execution of each action guaranteed by the inherited SmartFrog-provided components follows a step-by-step deploying pattern in which each step deploys a file/program/resource required for the subsequent steps and expects a file/program/resource is deployed by the previous step. In case the sequence of components defined in *z15_4_NodeSystemBase.sf* fails at any stage all deployed files , resources, directories etc. up to that point are automatically removed enforcing the 'all – or nothing' rule for the presence of all required basic resources before any further action is undertaken. Such automatic removal is not performed in case the sequence of components defined in *z15_4_NodeSystemTier.sf* fails as it is assumed that corrective actions are possible by the user at that point without the need of re-starting the system deployment from the beginning. This sequence as a whole is the first action of deployment on a target host. The successful execution results in the installation of a base of files, programs and resources along with running Tomcat Web Server with deployed one Grid Service.

file: *z15_4_NodeSystemRedeployTier.sf*

This file contains the description of a component consisting of sequence of actions. The actions are performed on a deployment host by components executing on that host.

component: *z15_4NodeSystemRedeployTier extends Sequence {}*

Each action executes a previously defined component making this component just an aggregate one. The sequential execution of each action guaranteed by the inherited SmartFrog-provided components follows a step-by-step deploying pattern in which each step deploys a file/program/resource required for the subsequent steps and expects a file/program/resource is deployed by the previous step. This sequence as a whole expects the component in *z15_4_NodeSystemBase.sf* be successfully executed which results in the installation of a base of files, programs and resources required for the successful execution of this sequence. Successful execution of all components of this sequence results in a 'hot' RE-deployment of OGSA directory structure as a Web Application on running Tomcat Web Server. The sequence is used for deployment of additional Grid Services on already deployed OGSA base and running Web Server.

file: z15_4_PartialGridServiceUnDeployer.sf

This file contains the description of a component consisting of a sequence of actions. The actions are performed on a deployment host by components executing on that host.

component: *z15_4PartialGridServiceUnDeployer extends Sequence {}*

Each action executes a previously defined component making this component just an aggregate one. The sequential execution of each action guaranteed by the inherited SmartFrog-provided components follows a step-by-step deploying pattern in which the first step results in a Grid Service being un-deployed from OGSA and the second results of 'hot' RE-loading of OGSA as a Web Application on running Tomcat Web Server so the changes in OGSA i.e. the removed Grid Service is reflected in Tomcat's list of available services. The sequence is used for removing of a Grid Service without disruption to the work of other Grid Services.

file: *z15_4_FullInstallationUnDeployer.sf*

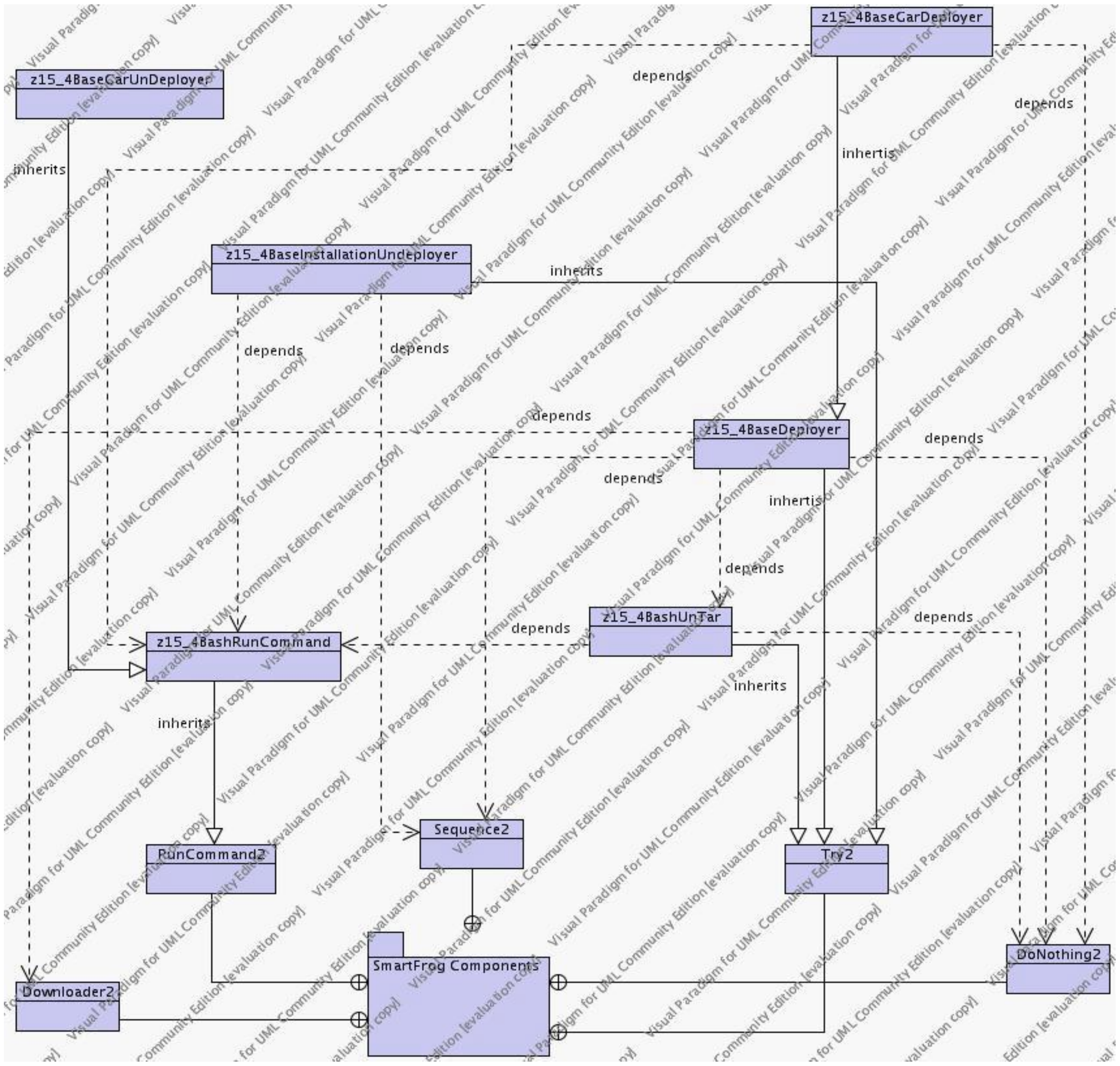
This file contains the description of a component that inherits installation un-deployment from the previously defined *z15_4BaseInstallationUnDeployer*.

component: *z15_4FullInstallationUnDeployer extends z15_4BaseInstallationUnDeployer {}*

This component overwrites attributes and methods to enhance its functionality i.e. it prepends a new action before the inherited action of removing completely the installation directory(see *z15_4BaseComponents.sf*). The new action results in a clean shut-down of Tomcat Web Server before the full installation un-deployment.

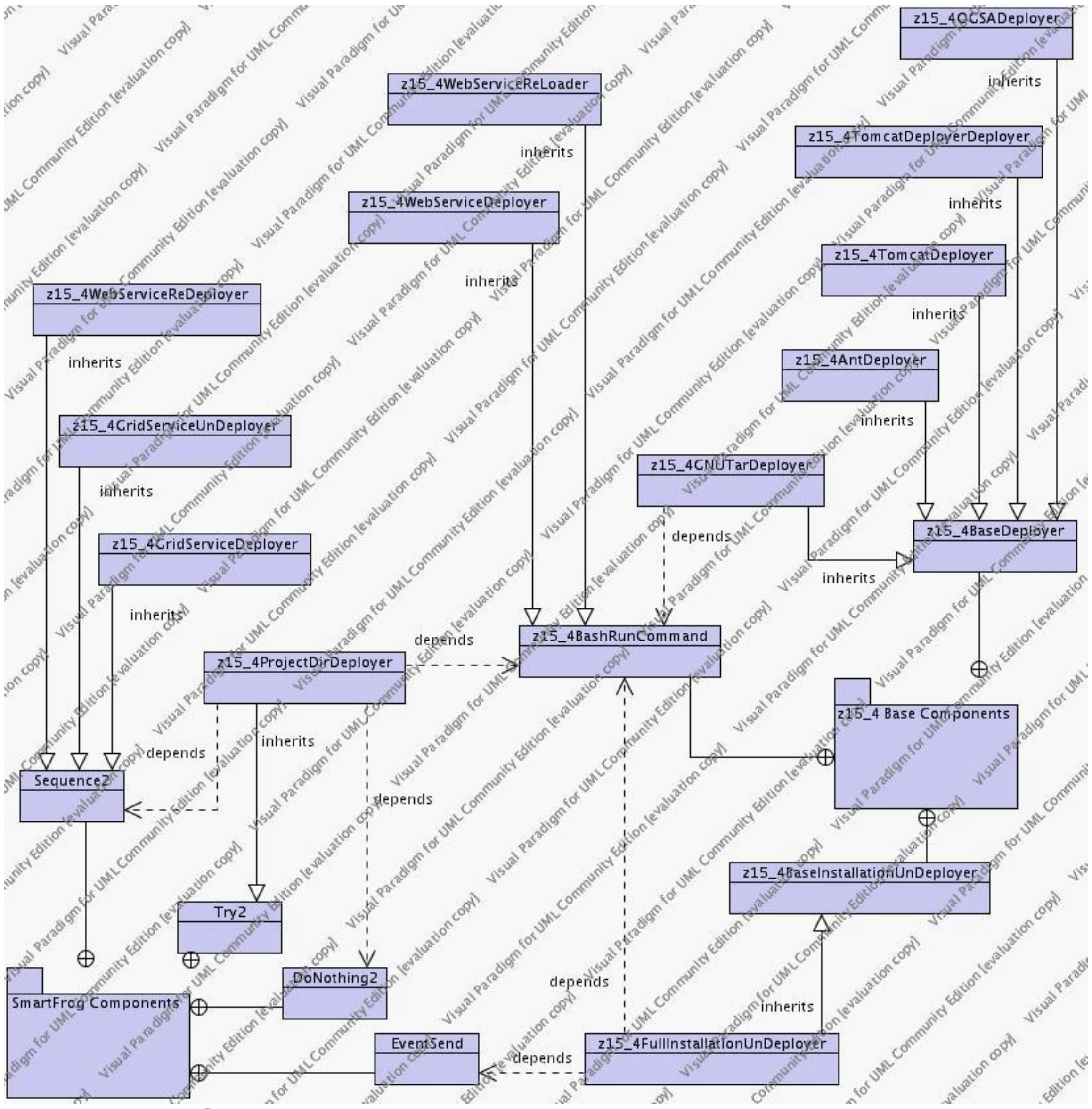
SmartFrog Class Diagrams

The following three class diagrams illustrate the individual components and their inter-relations built to deliver the functionality described in the activity diagrams in the preceding pages. All components were grouped in a hierarchical framework in which SmartFrog-provided components form the base and each new component in each successive layer of components inherits core functionality and expands its characteristics by customizing them for a specific task. Structuring the design in this way attempts to take advantage of all the benefits derived from an Object-Oriented, 'black-box' type, design where different components could be 'plugged' in and out as required in later evolutions of the architecture.



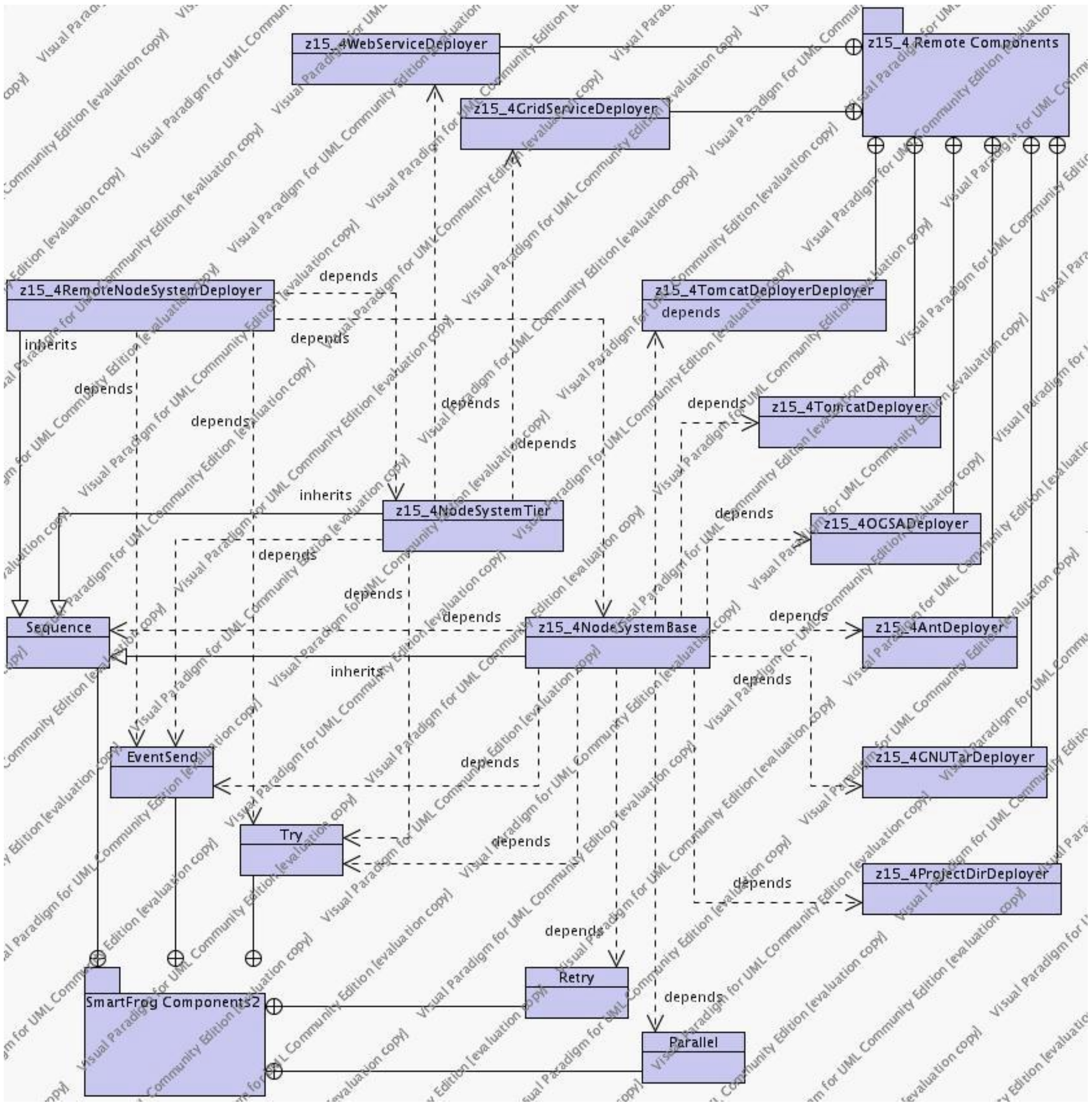
ClassDiag-01

ClassDiag-01 shows the components of the first level of the component hierarchy. These components (all starting with z15_4*) derived directly from SmartFrog-provided components which as explained earlier form the base of the inheritance hierarchy. This group of components provide the core functionality required for the deployment of Grid Services as explained at the beginning of the chapter; downloading files/resources, decompressing them, deploying and installing files/resources and executing Operating System (OS) - specific commands in a sequential or repetitive manor. The bulk of their behavior is derived unchanged from their SmartFrog 'super'-components taking advantage of one of the fundamental principles of object-oriented design and programming – the encapsulation of a particular functionality in re-usable piece of code.



ClassDiag-02

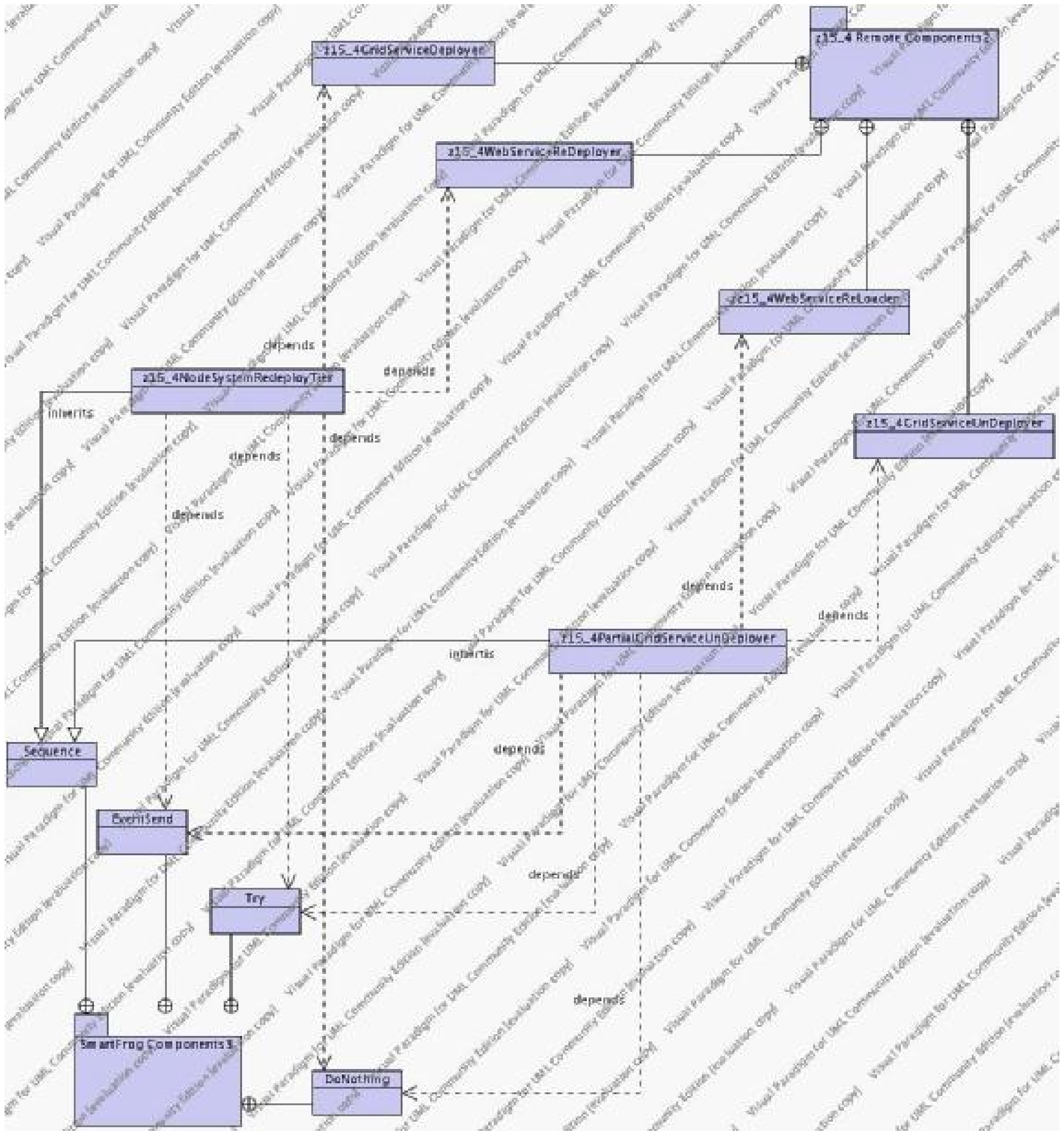
ClassDiag-02 shows the second level of the hierarchy. As it is illustrated by the graph, this group of components derive their functionality not only from the basic SmartFrog components but also from the group of components at the first level of the *z15_4** component framework. At this level, however, the new components tailor their individual behavior to suit a more narrow purpose than the one inherited from their 'super'-components. For example the component *z15_4GNUTarDeployer* derives core deploying and OS command execution behavior from *z15_4BaseDeployer* and *z15_4BashRunCommand* respectively, however, it is customized to perform these actions for the GNU version of the Unix Tar utility only. In a similar way the rest of the components in this group guarantee the deployment of all required resources involved in Grid Service deployment – Apache Ant, Tomcat Web Server, Tomcat Deployer Package, OGSA infrastructure.



ClassDiag-03

ClassDiag-03 shows the third level of the hierarchy. As it is illustrated by the graph, this group of components derive their functionality again from the basic SmartFrog

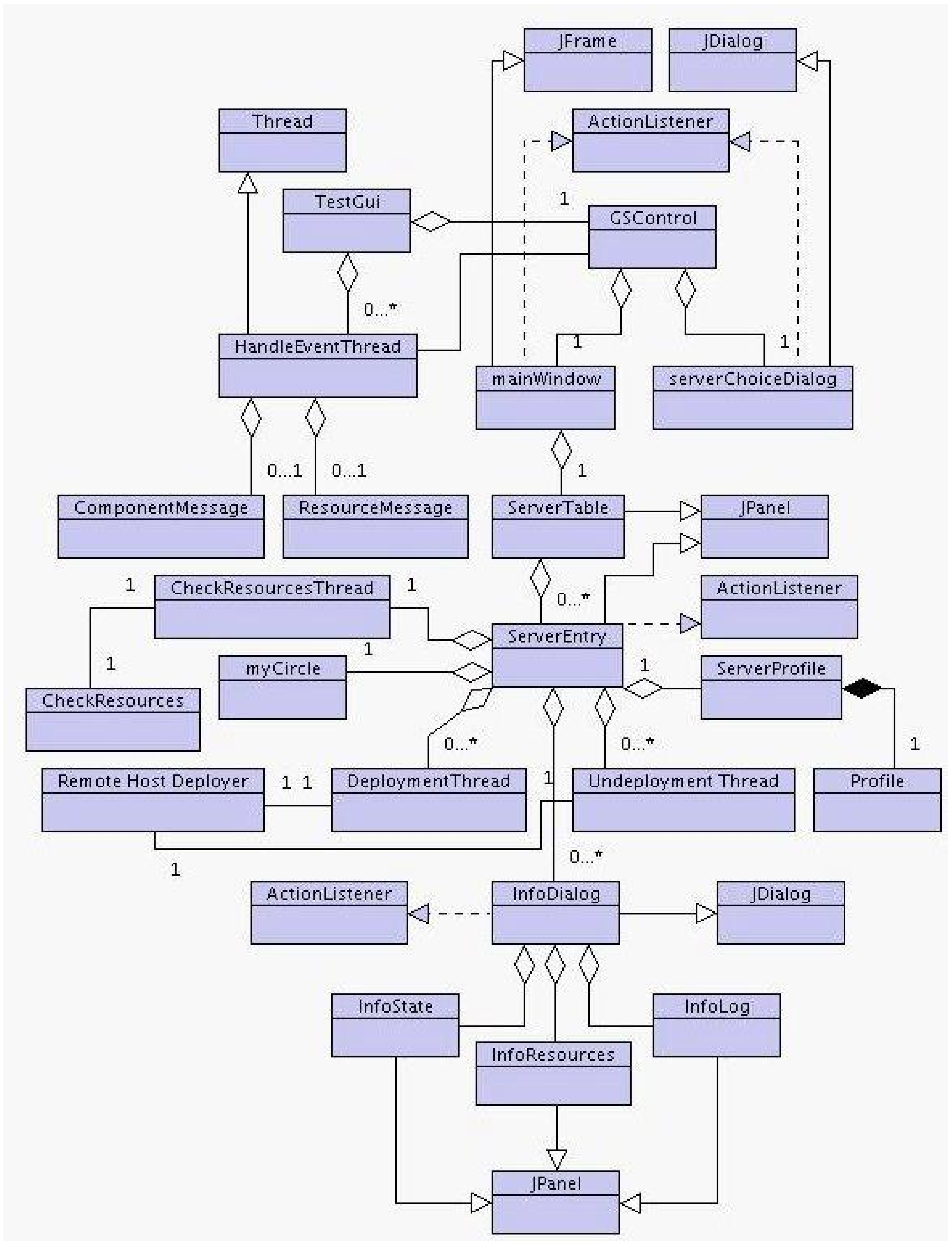
components and along with this they are build using the group of components at the second level of the *z15_4** component framework. At this level, however, the components are more of aggregate types for the second-tier components. The three unifying 'top'-components - *z15_4NodeSystemBase*, *z15_4NodeSystemTier*, *z15_4RemoteNodeSystemDeployer*, defined here combine the functionality of multiple components defined in *ClassDiag-02*. As their names suggest, they are the underlying modules providing all the functionality described in the preceding pages covering *ActivityDiag-02*, *ActivityDiag-03* and *ActivityDiag-04*. As a whole these three components encompass the bulk of the process required to deploy a Grid Service on a given deployment host.
The following class diagram illustrates the individual components and their inter-relations built to deliver the functionality described in the two activity diagrams in the preceding pages. These components are part of the hierarchical framework of SmartFrog-based components designed in the Elaboration phase. Just like the previous group of components they are constructed by inheriting behavior from the SmartFrog base as well as aggregating some of the *z15_4** first-tier components.



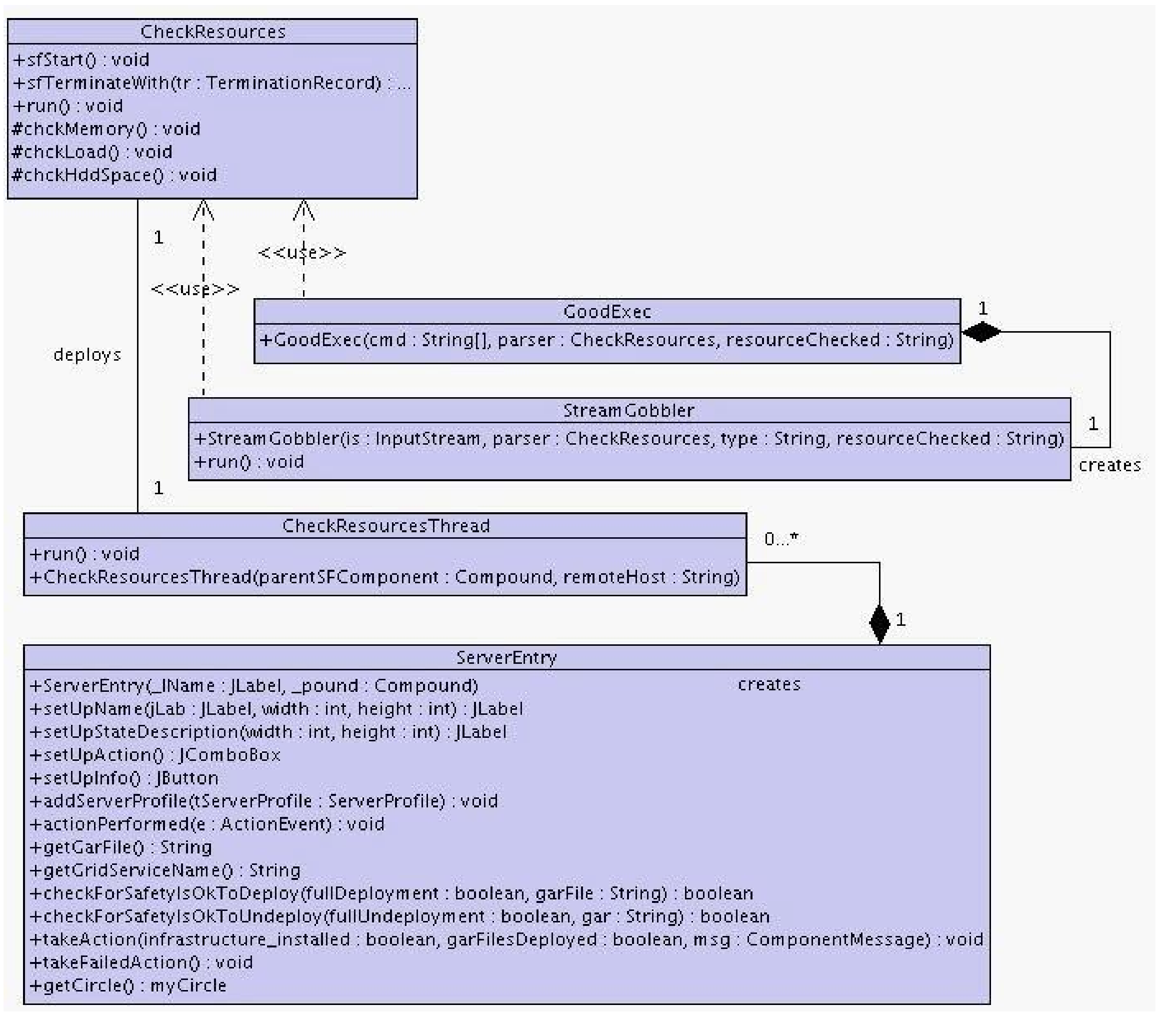
ClassDiag-04

ClassDiag-04 shows the fourth level of the hierarchy designed in the previous development phase. As it is illustrated by the graph, this group of components derive their functionality again from the basic SmartFrom components and along with this they are build using the group of components at the third level of the *z15_4** component framework (see *ClassDiag-02*). At this level, again, the components are aggregate types for the third-tier components. The two unifying 'top'-components - *z15_4NodeSystemRedeployTier*, *z15_4PartialGridServiceUnDeployer*, defined here combine the functionality of multiple components defined in *ClassDiag-02*. They are the underlying components providing all the functionality described in the preceding pages covering *ActivityDiag-05* and *ActivityDiag-06*.

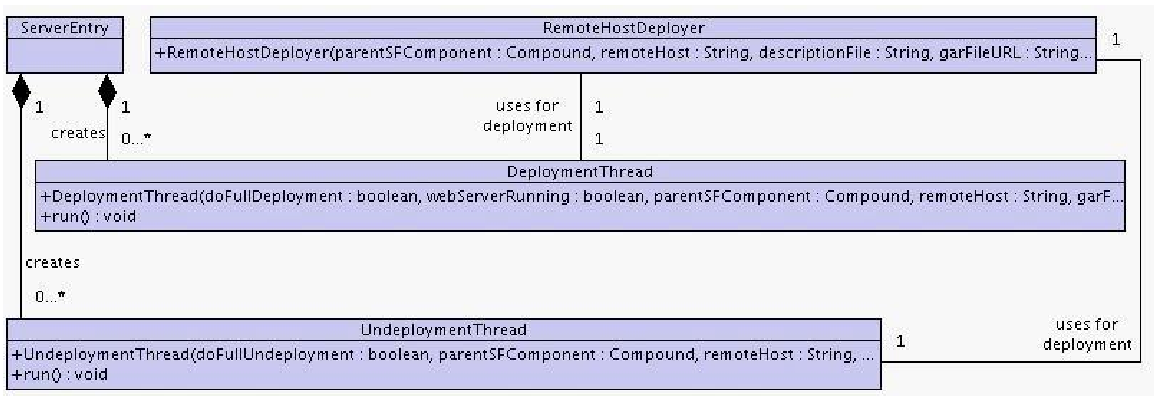
2. Java Components



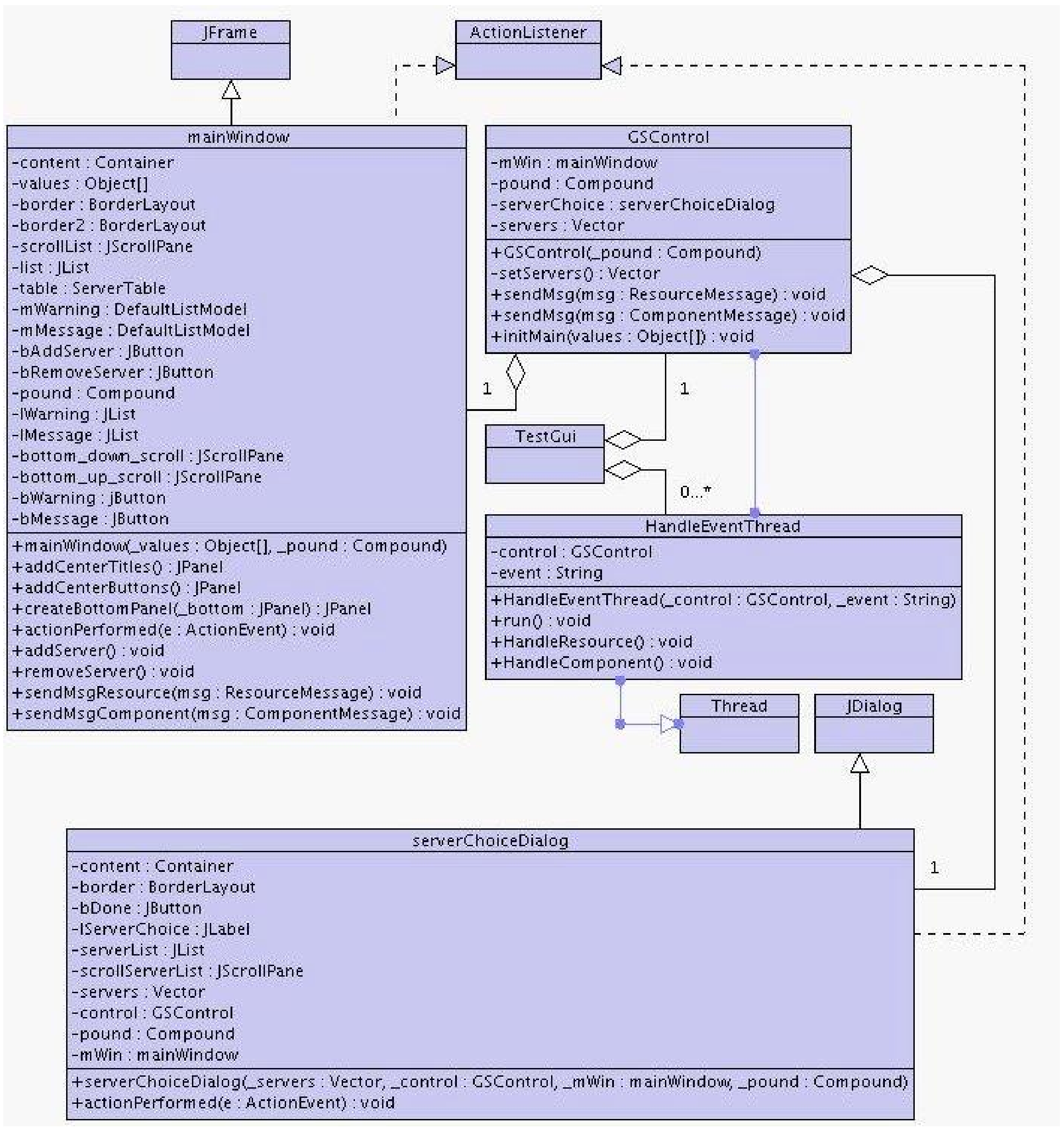
This is a class diagram of the associations between the classes that make up the GUI. In the following diagrams you can see more details regarding specific relationships between some of the above classes.


CheckResources
+sfStart() : void
+sfTerminateWith(tr : TerminationRecord) : ...
+run() : void
#chckMemory() : void
#chckLoad() : void
#chckHddSpace() : void
1
<<use>>
<<use>>
deploys
GoodExec
+GoodExec(cmd : String[], parser : CheckResources, resourceChecked : String)
1
StreamGobbler
+StreamGobbler(is : InputStream, parser : CheckResources, type : String, resourceChecked : String)
+run() : void
1
creates
1
CheckResourcesThread
+run() : void
+CheckResourcesThread(parentSFComponent : Compound, remoteHost : String)
0...*
1
ServerEntry
+ServerEntry(_lName : JLabel, _pound : Compound)
creates
+setUpName(jLab : JLabel, width : int, height : int) : JLabel
+setUpStateDescription(width : int, height : int) : JLabel
+setUpAction() : JComboBox
+setUpInfo() : JButton
+addServerProfile(tServerProfile : ServerProfile) : void
+actionPerformed(e : ActionEvent) : void
+getGarFile() : String
+getGridServiceName() : String
+checkForSafetyIsOkToDeploy(fullDeployment : boolean, garFile : String) : boolean
+checkForSafetyIsOkToUndeploy(fullUndeployment : boolean, gar : String) : boolean
+takeAction(infrastructure_installed : boolean, garFilesDeployed : boolean, msg : ComponentMessage) : void
+takeFailedAction() : void
+getCircle() : myCircle

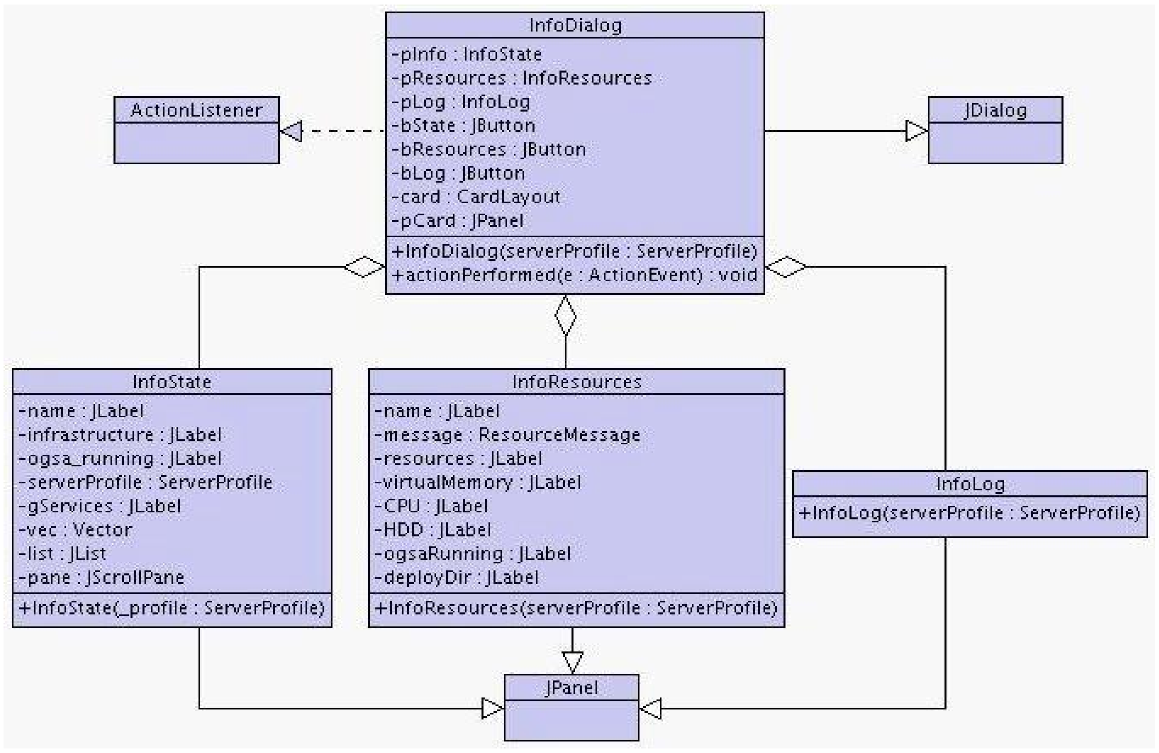
The above diagram shows the ServerEntry class which spawns threads in order to check the resources in each machine. The diagram shows the relationship between the classes that are used for that.


ServerEntry
RemoteHostDeployer
+RemoteHostDeployer(parentSFComponent : Compound, remoteHost : String, descriptionFile : String, garFileURL : String...
1
1
1
creates
0...*
uses for deployment
1
1
DeploymentThread
+DeploymentThread(doFullDeployment : boolean, webServerRunning : boolean, parentSFComponent : Compound, remoteHost : String, garF...
+run() : void
creates
0...*
UndeploymentThread
+UndeploymentThread(doFullUndeployment : boolean, parentSFComponent : Compound, remoteHost : String, ...
+run() : void
1
uses for deployment

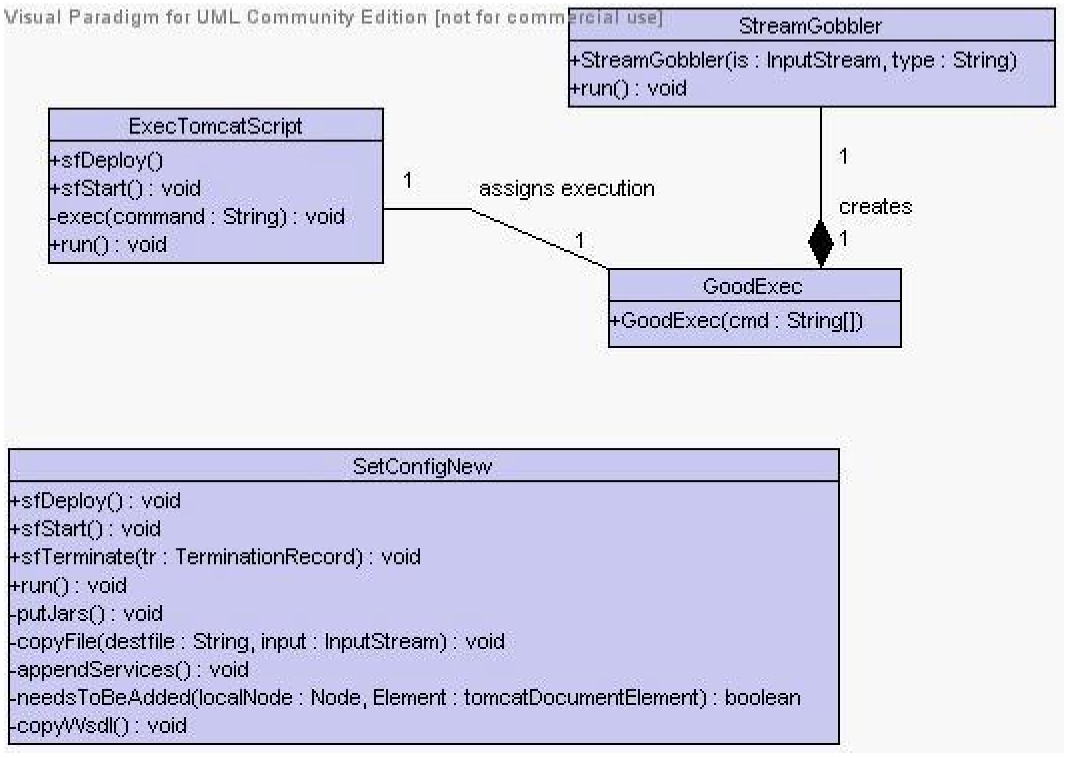
The above diagram shows the threads which are used for deployment and undeployment which are started from the ServerEntry class. It shows the relationship between the classes that are used for this purpose.


JFrame
ActionListener
mainWindow
-content : Container
-values : Object[]
-border : BorderLayout
-border2 : BorderLayout
-scrollList : JScrollPane
-list : JList
-table : ServerTable
-mWarning : DefaultListModel
-mMessage : DefaultListModel
-bAddServer : JButton
-bRemoveServer : JButton
-pound : Compound
-lWarning : JList
-lMessage : JList
-bottom_down_scroll : JScrollPane
-bottom_up_scroll : JScrollPane
-bWarning : JButton
-bMessage : JButton
+mainWindow(_values : Object[], _pound : Compound)
+addCenterTitles() : JPanel
+addCenterButtons() : JPanel
+createBottomPanel(_bottom : JPanel) : JPanel
+actionPerformed(e : ActionEvent) : void
+addServer() : void
+removeServer() : void
+sendMsgResource(msg : ResourceMessage) : void
+sendMsgComponent(msg : ComponentMessage) : void
GSControl
-mWin : mainWindow
-pound : Compound
-serverChoice : serverChoiceDialog
-servers : Vector
+GSControl(_pound : Compound)
-setServers() : Vector
+sendMsg(msg : ResourceMessage) : void
+sendMsg(msg : ComponentMessage) : void
+initMain(values : Object[]) : void
1
1
TestGui
0...*
HandleEventThread
-control : GSControl
-event : String
+HandleEventThread(_control : GSControl, _event : String)
+run() : void
+HandleResource() : void
+HandleComponent() : void
Thread
JDialog
serverChoiceDialog
-content : Container
-border : BorderLayout
-bDone : JButton
-lServerChoice : JLabel
-serverList : JList
-scrollServerList : JScrollPane
-servers : Vector
-control : GSControl
-pound : Compound
-mWin : mainWindow
+serverChoiceDialog(_servers : Vector, _control : GSControl, _mWin : mainWindow, _pound : Compound)
+actionPerformed(e : ActionEvent) : void
1

The above diagram shows the relationships between the classes that comprise an important part of the GUI's functionality. These classes are responsible for starting the GUI, allowing the user to choose servers and displaying the main window.



The above diagram shows the relationships between the classes that make up the info dialog of the GUI which provides information for each server.


Visual Paradigm for UML Community Edition [not for commercial use]
StreamGobbler
+StreamGobbler(is : InputStream, type : String)
+run() : void
ExecTomcatScript
+sfDeploy()
+sfStart() : void
-exec(command : String) : void
+run() : void
1
assigns execution
1
1
creates
1
GoodExec
+GoodExec(cmd : String[])
SetConfigNew
+sfDeploy() : void
+sfStart() : void
+sfTerminate(tr : TerminationRecord) : void
+run() : void
-putJars() : void
-copyFile(destfile : String, input : InputStream) : void
-appendServices() : void
-needsToBeAdded(localNode : Node, Element : tomcatDocumentElement) : boolean
-copyWsdl() : void

This is the class diagram for the elaboration phase.

Java Classes

Class: CheckResources

CheckResources

public **CheckResources**()
throws java.rmi.RemoteException

Constructs CheckResources (calls the constructor of the super class)

Throws:
java.rmi.RemoteException - in case of RMI or network failure.

sfDeploy

public void **sfDeploy**()
throws org.smartfrog.sfcore.common.SmartFrogException,
java.rmi.RemoteException

No Deployment requirements for this component. Deploy what is defined in the super class.

Specified by:
sfDeploy in interface org.smartfrog.sfcore.prim.Prim
Throws:
java.rmi.RemoteException - In case of network/rmi error
SmartFrogDeploymentException - In case of any error while deploying the component
org.smartfrog.sfcore.common.SmartFrogException

sfStart

public void **sfStart**()
throws org.smartfrog.sfcore.common.SmartFrogException,
java.rmi.RemoteException

Constructs and sends the message with the resources of the host.

Specified by:
sfStart in interface org.smartfrog.sfcore.prim.Prim
Throws:
org.smartfrog.sfcore.common.SmartFrogException - In case of SmartFrog system error
java.rmi.RemoteException - In case of network/rmi error

sfTerminateWith

public void **sfTerminateWith**(org.smartfrog.sfcore.prim.TerminationRecord tr)

Terminate the thread nicely if needed, can wait for it to awake to actually do so

run

public void **run**()

The body of the thread.It calls methods to check the available memory of the JVM,the CPU load,the available hard disk space (with this order).It also checks if the URL pointing to OGSA (as a web service in tomcat) is valid and whether the deployment directory that is created temporarily on each host.Then it waits for a period of time (default 5 mins) and repeats the whole process.

Specified by:
run in interface java.lang.Runnable

parseCPULoadResult

public void **parseCPULoadResult**()

Parses the result send as a string for the command line CPU check in order to define the outcome of the check

parseHddSpaceResult

public void **parseHddSpaceResult**()

Parses the result send as a string for the command line hard disk space check in order to define the outcome of the check

Class: CheckResourcesThread

run

public void **run**()

Creates a RemoteHostDeployer object with the appropriate parameters to deploy the the CheckResources component on the remote host

Class: DeploymentThread

run

public void **run**()

Depending on the type of the deployment, it creates a RemoteHostDeployer object with the appropriate parameters to deploy the grid service on the remote host

Class: ExecTomcatScript

ExecTomcatScript

public **ExecTomcatScript**()
    throws java.rmi.RemoteException

Constructor of the class Calls the constuctor of PrimImpl

sfDeploy

public void **sfDeploy**()
    throws java.rmi.RemoteException,
        org.smartfrog.sfcore.common.SmartFrogException

Overrides the sfDeploy() of PrimImpl. Resolves the references to the name of the script

Specified by:
sfDeploy in interface org.smartfrog.sfcore.prim.Prim

Throws:
java.rmi.RemoteException
org.smartfrog.sfcore.common.SmartFrogException

sfStart

public void **sfStart**()
    throws java.rmi.RemoteException,
        org.smartfrog.sfcore.common.SmartFrogException

Overrides the sfStart() of PrimImpl. It plays the role fo a main method in smartfrog deployment framework. In this case it sets the CATALINA_HOME environment variable and calls the exec() to run the script

Specified by:
sfStart in interface org.smartfrog.sfcore.prim.Prim

Throws:
java.rmi.RemoteException
org.smartfrog.sfcore.common.SmartFrogException

run

public void **run**()

The terminator thread that is created calls the sfTerminate() to terminate the component

Specified by:
run in interface java.lang.Runnable

Class: GoodExec

GoodExec

```
public GoodExec(java.lang.String[] cmd,
    CheckResources parser,
    java.lang.String resourceChecked)
  throws java.lang.Exception
```

Constructor of the class

Parameters:
cmd - the String array with the command arguments
parser - the CheckResources object that created the GoodExec object
resourceChecked - the type of resource to be checked with the command

Throws:
java.lang.Exception

Class: RemoteHostDeployer

RemoteHostDeployer

```
public RemoteHostDeployer(org.smartfrog.sfcore.compound.Compound parentSFComponent,
          java.lang.String remoteHost,
          java.lang.String descriptionFile,
          java.lang.String garFileURL,
          java.lang.String garId)
     throws java.rmi.RemoteException,
          org.smartfrog.sfcore.common.SmartFrogRuntimeException
```

Constructor of the class The attributes added in the component decription dynamically are the 'sfProcessHost', the 'mainGUIeventQueue', the 'baseGarFileName', the 'baseGarUrlAddress' and the 'baseGarId'

Parameters:
parentSFComponent - the Compound used to deploy the new component
remoteHost - the host on which the component will be deployed
descriptionFile - the file with Description
garFileURL - the URL of the gar file in case of deployment of a grid service

garId - the id of the gar file in case of undeployment of a grid service

Class: SetConfigNew

SetConfigNew

public **SetConfigNew**()
  throws java.rmi.RemoteException

Constructor of the class Calls the constuctor of PrimImpl

sfDeploy

public void **sfDeploy**()
  throws java.rmi.RemoteException,
    org.smartfrog.sfcore.common.SmartFrogException

Overrides the sfDeploy() of PrimImpl. Resolves all the references in the description file that contain the names of the files and the directories that have to be changed and updated

Specified by:
sfDeploy in interface org.smartfrog.sfcore.prim.Prim
Throws:
java.rmi.RemoteException
org.smartfrog.sfcore.common.SmartFrogException

sfStart

public void **sfStart**()
  throws java.rmi.RemoteException,
    org.smartfrog.sfcore.common.SmartFrogException

Overrides the sfStart() of PrimImpl. It plays the role fo a main method in smartfrog deployment framework. In this case it calls all the methods to implement the necessary transfers and configurations in the correct order

Specified by:
sfStart in interface org.smartfrog.sfcore.prim.Prim
Throws:
java.rmi.RemoteException
org.smartfrog.sfcore.common.SmartFrogException

sfTerminate

public void **sfTerminate**(org.smartfrog.sfcore.prim.TerminationRecord tr)

Overrides the sfTerminate() of PrimImpl. Used to terminate the smartfrog component

Specified by:
sfTerminate in interface org.smartfrog.sfcore.prim.Prim

run

public void **run**()

The terminator thread that is created calls the sfTerminate() to terminate the component

Specified by:
run in interface java.lang.Runnable

Class: UndeploymentThread

run

public void **run**()

Depending on the type of the undeployment, it creates a RemoteHostDeployer object with the appropriate parameters to undeploy the grid service on the remote host

Class: ComponentMessage

The ComponentMessage class represents a message which is sent as a result of an attempt to deploy a component. It contains information about the host, the date, and the type and kind of the message. Various static variables are used in order to simplify operations in this and other classes. The messageType refers to the attempt to deploy a component and the messageKind to the importance of that attempt. COMMON_CONSUMPTION components may fail without creating any problem to the server whereas INTRIGUING components' failure will have a stronger effect. The ComponentMessage exists as a matter of convenience. It is easier to hold all the information in one object, in a specific format which is easy to request and have available throughout the program.

ComponentMessage

public **ComponentMessage**(java.lang.String _host,
                java.lang.String _msg)

The constructor that creates a ComponentMessage. The msg is parsed in order to determine the messageType. According to the messageType the messageString is also created and a value is assigned to messageKind.

Parameters:

_host - the host from where this message originated.

_msg - the message which must be in the format ex. systemBase-SUCCEEDED.

getMessageKind

public int **getMessageKind**()

Returns the kind of message.

Returns:

the messageKind.

getMessageType

public int **getMessageType**()

Returns the type of messaage.

Returns:

the messageType.

getMessageString

public java.lang.String **getMessageString**()

Returns the message itself (without the indication for success or failure. If that is needed the getMessageResultString() should be used as well.

Returns:

the message.

getMessageResult

public int **getMessageResult**()

Returns the result of the deployment effort.

Returns:

the result of the deployment effort.

getMessageResultString

public java.lang.String **getMessageResultString**()

Calculates and returns the result of the deployment effort in the form of a string. It is determined based on the messageResult variable.

Returns:
the result of the deployment attempt.

getDate

public java.util.Date **getDate**()

Returns the date of the message.

Returns:
the date of the message.

getHost

public java.lang.String **getHost**()

Returns the host where the message originated from.

Returns:
the host.

Class: GSControl

This class reads the servers from the server file and creates the server choice dialog. It also acts as part of a channel where the two types of messages, resource and type, pass through.

GSControl

public **GSControl**(org.smartfrog.sfcore.compound.Compound _pound)

The constructor for the class GSControl. It reads the servers, stores them in a vector and starts the serverChoiceDialog. If no servers are found the mainWindow is initialized instead of the serverChoiceDialog.

sendMsg

public void **sendMsg**(ResourceMessage msg)

Invokes a method in mainWindow in order to pass along a resource message.

Parameters:
msg - a resource message.

sendMsg

public void **sendMsg**(ComponentMessage msg)

Invokes a method in mainWindow in order to pass along a component message.

Parameters:
msg - a component message.

initMain

public void **initMain**(java.lang.Object[] values)

Initializes the main window.

Parameters:
values - an array of server names.

Class: HandleEventThread

This class extends the Thread class. It is used to determine what a message is and processes it accordingly.

run

public void **run**()

This method is invoked automatically. It checks the first character of the string event and determines whether it is a Resource message or a Component message.

HandleResource

public void **HandleResource**()

This method creates a ResourceMessage by doing some simple parsing on the event string. It then invokes the sendMsg method at GSControl and passes along the message.

HandleComponent

public void **HandleComponent**()

This method creates a ComponentMessage by doing some simple parsing on the event string. It then invokes the sendMsg method at GSControl and passes along the message.

Class: InfoDialog

This is the main class for the info dialog window which contains information for each server. It loads the current information from the profile of each server. It contains two panels, one for the buttons and another one which is given a card layout and holds instances of InfoState, InfoResources and InfoLog. The buttons in the upper panel allow for navigation through the cards.

InfoDialog

`public InfoDialog(ServerProfile serverProfile)`

The constructor accepts a ServerProfile object which contains all the necessary information.

Parameters:

`serverProfile` - a server profile.

actionPerformed

`public void actionPerformed(java.awt.event.ActionEvent e)`

It is invoked when a button with an action listener is pushed. It is used in order to navigate through the cards.

Specified by:

`actionPerformed` in interface `java.awt.event.ActionListener`

Parameters:

`e` - the action event which has taken place.

Class: InfoLog

The class InfoLog is a panel which contains the log file of a server. It can be found in the third card in the InfoDialog.

InfoLog

public **InfoLog**(ServerProfile profile)

The constructor accepts the server profile, finds the name of the server, uses it to find the log fine, opens and reads it into a vector and then initializes a JList with that vector.

Parameters:

profile - a server profile object.

Class: InfoResources

This class is the second card in the InfoDialog and its role is to show the most recent report of resources that has been received. It will only show a report when it has one.

InfoResources

public **InfoResources**(ServerProfile profile)

The constructor for the class accepts a ServerProfile object and uses the information in it to construct the panel.

Parameters:

profile - a server profile object.

Class: InfoState

This class constructs objects for the first card of the InfoDialog. Its role is to present information regarding the state of the objects such as the name, whether infrastructure exists or not, whether OGSA is running or not and what Grid Services (if any) have been deployed.

InfoState

public **InfoState**(ServerProfile _profile)

The constructor receives a ServerProfile object as a parameter and uses the information in it in order to create the panel.

Parameters:

_profile - a server profile object.

Class: Profile

This class is only used for purposes of serialization into a file. It holds the basic information about the state of a server. This way, when for some reason GSControl closes the information will be kept in the file.

Profile

public **Profile**()

Profile

public **Profile**(boolean _ogsa_running,
            boolean _infrastructure_installed,
            java.util.Vector _gars)

Class: ResourceMessage

The class ResourceMessage is used in order to create objects which are meant to hold the information contained in a resource message. Resource messages are sent by the servers in frequent intervals and every time one arrives a ResourceMessage is created in order to parse the message, extract the information and hold it in its variables. Resource messages are important in order to monitor the state of servers and take action when necessary. Three different resources are checked and stored in the message, Virtual memory, CPU and HDD. Two additional pieces of information are also given, whether OGSA is running and whether the deployment directory exists. The ResourceMessage exists as a matter of convenience. It is easier to hold all the information in one object, in a specific format which is easy to request and have available throughout the program.

ResourceMessage

public **ResourceMessage**(java.lang.String _host,
                java.lang.String _msg)

The ResourceMessage's constructor accepts the host and the message. Then parses the message in order to look for the various possible messages that it may contain. There are three possible values for each resource, GOOD which means that everything is in order, BAD which means that there is a problem with the resource, the server may fail and a warning is shown, and UNPAIKTABLE, which means that the resource for some reason can not be checked. The constructor also calculates the general state of the server. That is done by checking all resources and GOOD is returned if each resource is deemed as GOOD or UNPAIKTABLE.

Parameters:
_host - the host where the message originated from.
_msg - the msg itself.

getMessage

public java.lang.String **getMessage**()

This method returns the state of the server as a string.

Returns:

the general state of the server

getState

public int **getState**()

This method returns the the state of the server as int.

Returns:

the state of the server.

getHost

public java.lang.String **getHost**()

This method returns the host that the message originated from.

getDate

public java.util.Date **getDate**()

This method returns the date of the message.

Returns:

the date.

getVirtualMemory

public int **getVirtualMemory**()

This method returns the Virtual memory of the server.

Returns:

the virtual memory.

getCPU

public int **getCPU**()

This method returns the state of the CPU of the server.

Returns:

the state of the CPU.

getHDD

public int **getHDD**()

This method returns the state of the HDD of the server.

Returns:
the state of the HDD.

getOGSARunning

public int **getOGSARunning**()

This method returns OGSA's state.

Returns:
OGSA's state.

getDeployDir

public int **getDeployDir**()

This method returns the state of the deployment directory.

Returns:
the state of the deployment directory.

getVirtualMemoryMessage

public java.lang.String **getVirtualMemoryMessage**()

This method returns the Virtual memory of the server.

Returns:
the virtual memory.

getCPUMessage

public java.lang.String **getCPUMessage**()

This method returns the state of the CPU of the server.

Returns:
the state of the CPU.

getHDDMessage

public java.lang.String **getHDDMessage**()

This method returns the state of the HDD of the server.

Returns:
the state of the HDD.

getOGSARunningMessage

public java.lang.String **getOGSARunningMessage**()

This method returns OGSA's state.

Returns:
OGSA's state..

getDeployDirMessage

public java.lang.String **getDeployDirMessage**()

This method returns the state of the deployment directory.

Returns:
the state of the deployement directory.

Class: ServerEntry

Each object of this class becomes an entry in the server list. Besides being a panel which holds all the information that is needed in order to accurately describe each server, it also contains all the components that are displayed in the window. It also contains a ServerProfile object which holds further information about the server and is actually used in order to render the components. By receiving user events it also facilitates the many choices that the user has. It is responsible for invoking the threads that will manage deployment and undeployment issues.

ServerEntry

```
public ServerEntry(javax.swing.JLabel _lName,
        org.smartfrog.sfcore.compound.Compound _pound)
```

The constructor initializes the JComponents by calling class methods and then simply adds the components to the panel. Finally, it also starts the CheckResources thread that must run on each server.

Parameters:
_lName - the label for the server.
_pound - the compound.

setUpName

```
public javax.swing.JLabel setUpName(javax.swing.JLabel jLab,
                                    int width,
                                    int height)
```

This method sets up the JLabel which holds the name of the server.

Parameters:
  jLab - the JLabel.
  width - the width of the label.
  height - the height of the label.
Returns:
  the label containing the name of the server.

setUpStateDescription

public javax.swing.JLabel **setUpStateDescription**(int width,
                                                  int height)

This method sets up the description of the state of the server. The initial default state is IDLE.

Parameters:
  width - the width of the state description label.
  height - the height of the state description label.
Returns:
  the label containing the description of the state of the server.

setUpAction

public javax.swing.JComboBox **setUpAction**()

This methods sets up the action combo box which the user can use in order to choose the action that he wants to take.

Returns:
  the combo box containing the possible actions.

setUpInfo

public javax.swing.JButton **setUpInfo**()

This methods sets up the info button. The user can press it to get more information about a particular server.

Returns:
  the info button.

addServerProfile

public void **addServerProfile**(ServerProfile tServerProfile)

This methods adds a server profile to the server entry.

Parameters:

tServerProfile - the server profile of the server entry.

actionPerformed

public void **actionPerformed**(java.awt.event.ActionEvent e)

This method is called whenever an action event takes place. It first checks to find the source of the event and then, if the event has come from the combo box, a deployment or undeployment action may be taken. If it has come from the info button, the info window will pop-up. These are the only two components that accept actions.

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Parameters:

e - the action event.

getGarFile

public java.lang.String **getGarFile**()

This method will pop up a window which is asking for the location of the gar file. It should be an http location.

Returns:

the location of the gar file.

getGridServiceName

public java.lang.String **getGridServiceName**()

This method will pop up a window asking for the name of the grid service. Note: This is the name of the gar file without the '.gar'.

Returns:

the name of the grid service.

checkForSafetyIsOkToDeploy

public boolean **checkForSafetyIsOkToDeploy**(boolean fullDeployment,
java.lang.String garFile)

This methods checks to make sure that deployment is indeed possible. According to the parameters that are passed, it checks for full or partial deployment, whether the gar file is already deployed, and whether its value is appropriate.

Parameters:

fullDeployment - is the deployment full (true) or partial (false).

garFile - the location of the gar file.

Returns:

is it safe to deploy (true) or not (false).

checkForSafetyIsOkToUndeploy

```
public boolean checkForSafetyIsOkToUndeploy(boolean fullUndeployment,
                                            java.lang.String gar)
```

This methods checks to make sure that undeployment is indeed possible. According to the parameters that are passed, it checks for full or partial undeployment, whether the infrastructure exists, etc.

Parameters:

fullUndeployment - is the undeployment full (true) or partial (false).

gar - the name of the grid service.

Returns:

is it safe to undeploy (true) or not (false).

takeAction

```
public void takeAction(boolean infrastructure_installed,
                       boolean garFilesDeployed,
                       ComponentMessage msg)
```

This method is responsible for altering the window components which reflect the state of the server. It is usually used after altering the state of the server.

Parameters:

infrastructure_installed - whether the infrastructure is installed or not.

garFilesDeployed - whether there are any grid services deployed.

msg - the component message.

takeFailedAction

```
public void takeFailedAction()
```

This method changes the label of the state of the server because of a failed action. It also changes the color of the server into red.

getCircle

```
public myCircle getCircle()
```

This method simply returns the component which holds the color of the server

Class: ServerProfile

This class is used to store information about the state of each individual server. It is also used to control the text files that are associated with each server. Two different kind of files are used. A log file, which is a text that holds the messages that are sent by the server and a state file which holds information about the state of the server. The information in the state file are serialized objects of the Profile class.

ServerProfile

public **ServerProfile**(java.lang.String _name,
    ServerEntry _entry)

The constructor of this class deals with the files that are related to the class by calling the appropriate functions. In the function that deals with the state information, if a file exists and a Profile object is read, the variables of the profile objeet are used to initiate the variables of the ServerProfile object. Otherwise, the variables are initiated as default. Finally the constructor also arranges the color which signifies the state of the server.

Parameters:
_name - the name of the server.
_entry - the ServerEntry object that is associated with the ServerProfile object.

checkState

public void **checkState**(java.lang.String name)

This method deals with the state file which is associated with the server. If the file does not exist it creates it. It then creates a Profile object using the default constructor and serializes it into the file. If the file exists, it deserializes it and stores it in a variable so that it can be read by the constructor of the ServerProfile class.

Parameters:
name - the server name.

getName

public java.lang.String **getName**()

This method returns the server's name.

Returns:
the server's name.

getServerEntry

public ServerEntry **getServerEntry**()

> This method returns the ServerEntry object that is associated with the current ServerProfile object.
>
> **Returns:**
> the server entry.

getState

public boolean **getState**()

> This method returns the general state of the server. The general state depends upon the state of the infrastructure and the state of OGSA.
>
> **Returns:**
> the server's general state.

getInfrastructure

public boolean **getInfrastructure**()

> This methods returns the state of the infrastructure.
>
> **Returns:**
> the state of the infrastructure.

getOgsaRunning

public boolean **getOgsaRunning**()

> This methods returns the state of OGSA.
>
> **Returns:**
> the state of OGSA.

isThereAnyGridService

public boolean **isThereAnyGridService**()

> This method checks the size of the garFiles vector in order to find out whether there is any grid service running at the server. It returns true if at least one grid service exists.
>
> **Returns:**
> whether there is any grid service deployed on the server.

addGarFile

public void **addGarFile**(java.lang.String garFile)

> This method adds a grid service to the server's profile. It is used when a new grid service has been succesfully deployed.
>
> **Parameters:**
> garFile - the grid service location that was deployed on the server.

garFileExists

public boolean **garFileExists**(java.lang.String garFile)

> This method checks to see if the gar file that was given as a parameter has been deployed on the server or not.
>
> **Parameters:**
> garFile - the gar file for which you want to check.
> **Returns:**
> whether the gar file is deployed on the server or not.

writeLogMessage

public void **writeLogMessage**(ResourceMessage msg)

> This method receives resoure messages from the servers and writes them in the log files. It uses methods of the ResourceMessage object to find all the necessary information.
>
> **Parameters:**
> msg - the resource message that you want written to the log file.

writeLogMessage

public void **writeLogMessage**(ComponentMessage msg)

> This method receives component messages from the servers and writes them in the log files. It uses methods of the ComponentMessage object to find all the necessary information.
>
> **Parameters:**
> msg - the component message that you want written to the log file.

alterState

public void **alterState**(ComponentMessage msg)

This method is responsible for altering the state of the server upon the reception of a component message, if that is necessary. The alterations that are made depend on the nature of the message. If the message signifies success a change may be made to the state of the server. If it signifies failure no change will be made to the state but an additional action will be taken.

setGarFileToBeDeployed

public void **setGarFileToBeDeployed**(java.lang.String gar)

This method sets the candidate for the gar file that may be deployed to this server.

Parameters:
gar - a gar file.

setGarFileToBeUndeployed

public void **setGarFileToBeUndeployed**(java.lang.String gar)

This method sets the candidate for the gar file that may be undeployed from this server.

Parameters:
gar - a gar file.

getGarFiles

public java.util.Vector **getGarFiles**()

This method returns the grid services that are deployed on the server.

Returns:
the grid services that are deployed on the server.

getMessage

public ResourceMessage **getMessage**()

This method returns the last resource message that was sent from this server.

Returns:
a resource message.

alterOgsa

public void **alterOgsa**(ResourceMessage msg)

This method alters the state of OGSA by receiving a resource message from the server. If OGSA's state in the message is marked as running, the state of the server is changed appropriately. The state of the server is subsequently serialized to the state file.

Parameters:
msg - the resource message.

Class: ServerTable

This class is used to hold ServerEntry objects and arrange them into a list. It contains methods which are used to add and remove servers. It is also used as a middle man and pushes messages to the corresponding server.

ServerTable

public **ServerTable**(java.lang.Object[] _servers,
    org.smartfrog.sfcore.compound.Compound _pound)

The constructor for this class creates and adds the server entries. It also creates the correspond ServerProfile objects and sets up the relationship between the two.

Parameters:
_servers - the names of the servers which will be created.
_pound - the compound.

addServerEntry

public void **addServerEntry**(java.lang.String name)

This method adds a new server entry to the server table. It also creates the corresponding ServerProfile objects and links the two.

Parameters:
name - the name of the server that is to be added.

removeServerEntry

public void **removeServerEntry**(java.lang.String name)

This method removes a server entry from the server table.

Parameters:
name - the name of the server that is to be removed.

pushResourceMessage

public void **pushResourceMessage**(ResourceMessage msg)

This method receives a resource message, finds the server which is the sender of the message, evokes a method from the ServerProfile class to write the message to the log and sends the message to the server. This message has the possibility to alter the OGSA's state.

Parameters:
msg - a resource message.

pushComponentMessage

public void **pushComponentMessage**(ComponentMessage msg)

This method receives a component message, finds the server which is the sender of the message, evokes a method from the ServerProfile class to write the message to the log and sends the message to the server. This message has the possibility, if it is deemed to be intriguing, to alter the state of the server.

Parameters:
msg - a resource message.

Class: mainWindow

This class creates the main window for the application. It takes care of all the components that the gui has.

mainWindow

public **mainWindow**(java.lang.Object[] _values,
org.smartfrog.sfcore.compound.Compound _pound)

The constructor accepts as arguments the servers that will be initiated and then initializes the ServerTable which sets into motion the initialization of all the servers.

Parameters:
_values - the servers
_pound - the compound

addCenterTitles

public javax.swing.JPanel **addCenterTitles**()

This method creates and adds the titles for the server list.

Returns:
a panel that includes the server titles.

addCenterButtons

public javax.swing.JPanel **addCenterButtons**()

It adds the two buttons, add and remove server.

Returns:
a panel that contains the two buttons.

createBottomPanel

public javax.swing.JPanel **createBottomPanel**(javax.swing.JPanel _bottom)

It creates the two panels, Warnings and Messages.

Parameters:
_bottom - the bottom jpanel.

actionPerformed

public void **actionPerformed**(java.awt.event.ActionEvent e)

It handles the event when the AddServer or RemoveServer buttons are pushed. It calls the appropriate method.

Specified by:
actionPerformed in interface java.awt.event.ActionListener
Parameters:
e - an action event.

addServer

public void **addServer**()

It adds a server to the server list. Creates all the necessary components.

removeServer

public void **removeServer**()

It removes a server from the server list. It creates a dialog where the user inputs the name of the server. It checks what the state of the server is before removal and informs the user.

sendMsgResource

public void **sendMsgResource**(ResourceMessage msg)

It receives a resource message and puts it in the message window by using the resource message's methods.

Parameters:
msg - a resource message

sendMsgComponent

public void **sendMsgComponent**(ComponentMessage msg)

It handles a component message that is received.

Parameters:
msg - a component message.

Class: myCircle

This class handles the creation of the colors which signify the condition of the server.

myCircle

public **myCircle**()

The constructor simply initializes the object and sets its dimension.

paint

public void **paint**(java.awt.Graphics g)

The paint method is used to paint the actual circle and its color.

Parameters:
g - the graphics.

drawMe

public void **drawMe**(java.awt.Color _color)

This method is called to change the color of the circle.

Parameters:
_color - the color that should be painted.

Class: serverChoiceDialog

This class handles the choice of the servers by opening a dialog which reads from the servers.txt file and allows the user to choose the servers that he wants to initialize. It is possible to choose one, some, all or no servers.

serverChoiceDialog

```
public serverChoiceDialog(java.util.Vector _servers,
                 GSControl _control,
                 org.smartfrog.sfcore.compound.Compound _pound)
```

The constructor of the class accepts the servers which have been read from the text file and initializes the dialog which displays the servers as a list.

Parameters:

_servers - the servers.
_control - an object of the GSControl class.
_pound - the compound.

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

This method is called when an action event is performed. If the button has been pressed the selected values are sent to the main window which is initialized.

Specified by:
actionPerformed in interface java.awt.event.ActionListener

Parameters:
e - an action event.