



**PROTOTYPE DOCUMENTATION**  
**TASK 3.3.3 JIRO BASED GRID INFRASTRUCTURE**  
**MONITORING**

**WP3**

---

Document Filename: **CG-3.3.3-CYF-D3.3-v1.1-JIRO.doc**  
Work package: **WP3**  
Partner(s): **CYFRONET**  
Lead Partner: **CYFRONET**  
Config ID: **CG3.3.3-CYF-D3.3-v1.1-Jiro**  
Document classification: **PUBLIC**

---

Abstract: This document contains the description of the process of installation of the first prototype of the Jiro Based Grid Infrastructure Monitoring System and a user guide for the prototype. Note that both installation procedure and user interface are subjects to change in the future.



### Delivery Slip

	Name	Partner	Date	Signature
<b>From</b>				
<b>Verified by</b>				
<b>Approved by</b>				

### Document Log

Version	Date	Summary of changes	Author
Draft	16/01/2003		Sławomir Zieliński
1.0	23/01/2003		Sławomir Zieliński
1.1	21/02/2003	Added reference to test description (T3.5 D3.3), minor editorial changes.	Sławomir Zieliński

---

## CONTENTS

<b>EXECUTIVE SUMMARY .....</b>	<b>4</b>
<b>1. INTRODUCTION .....</b>	<b>5</b>
1.1. PURPOSE .....	5
1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS .....	5
<b>2. REFERENCES .....</b>	<b>6</b>
2.1. SOURCE CODE .....	6
2.2. CONTACT INFORMATION .....	6
2.3. REFERENCED DOCUMENTS .....	6
<b>3. IMPLEMENTATION STRUCTURE .....</b>	<b>7</b>
<b>4. PROTOTYPE FUNCTIONALITY .....</b>	<b>9</b>
<b>5. USER MANUAL .....</b>	<b>11</b>
5.1. DEPENDENCIES .....	11
5.2. INSTALLATION .....	11
5.3. RUNNING.....	15
5.3.1. <i>ExtFinder</i> .....	15
5.3.2. <i>JiroMBeanServiceUI</i> .....	17
5.4. INTERFACE DESCRIPTION.....	18
<b>6. INTERNAL TESTS.....</b>	<b>19</b>
<b>7. ISSUES .....</b>	<b>20</b>

## **EXECUTIVE SUMMARY**

This document describes the first prototype of the Jiro-based grid infrastructure monitoring system. Sections 1 and 2 contain an introduction and references, respectively. Sections 3 and 4 describe the entities implemented in the first prototype as well as their functionality. The user manual is included in section 5.

## 1. INTRODUCTION

### 1.1. PURPOSE

The purpose of the system described in this document is to gather and expose information concerning the state of devices used to build a grid environment. The main application of the system will be grid state prediction, which is covered by Task 3.3.4. The idea is to reuse a standardised approach to instrument the monitored devices in order to expose their parameters to a database system in a unified way. The first prototype provides the means of instrumentation for computational and storage nodes and SNMP-capable devices.

### 1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

CrossGrid	The EU CrossGrid Project IST-2001-32243
DataGrid	The EU DataGrid Project IST-2000-25182
FMA	Federated Management Architecture, defined by Sun Microsystems
GUI	Graphical User Interface
Jiro	Sun Jiro, an implementation of the FMA
OGSA	Open Grid Services Architecture
R-GMA	DataGrid Relational Grid Monitoring Architecture

## 2. REFERENCES

### 2.1. SOURCE CODE

Because the first prototype was developed using commercial software, its code is not available publicly.

### 2.2. CONTACT INFORMATION

Contact information for partners responsible for the prototype and for technical staff:

Sławomir Zieliński, slawek@agh.edu.pl

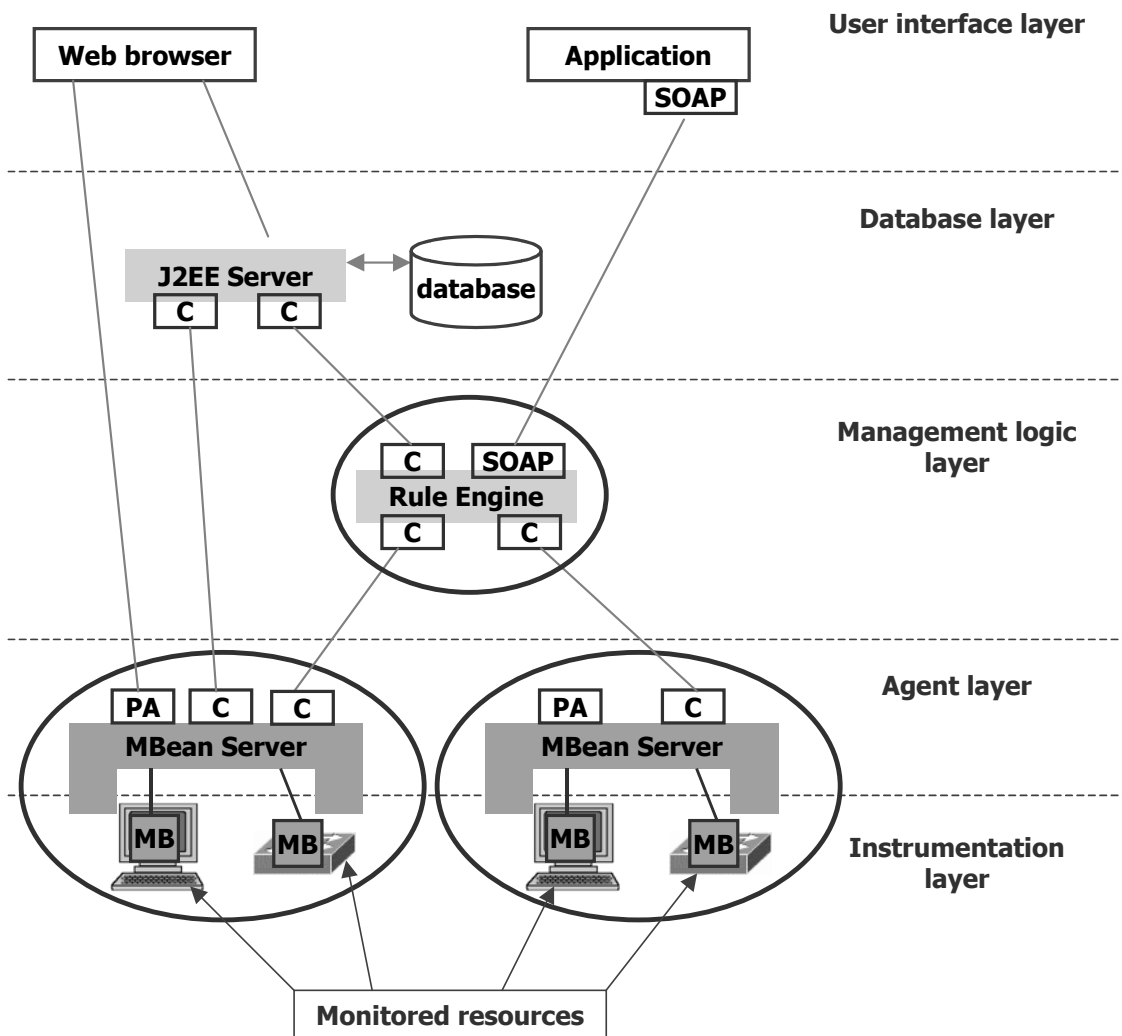
### 2.3. REFERENCED DOCUMENTS

- [1] Task 3.3 Grid Monitoring Software Requirements Specification CG-3.3-SRS-0013
- [2] Jiro Monitoring Design, Task 3.3.3 Deliverable 3.2,  
CG3.3.3-D3.2-v1.0-TCD100-JiroMonitoring
- [3] Java Management Extension Specification, © Sun Microsystems,  
[java.sun.com/products/JavaManagement/](http://java.sun.com/products/JavaManagement/)
- [4] Jiro Technology Installation and Configuration Guide, © Sun Microsystems
- [5] Test and Integration Process Description, Task 3.5, Deliverable 3.3,  
CG3.5-D3.3-v1.0-CSIC021-TestIntegrationPrototype

### 3. IMPLEMENTATION STRUCTURE

The system will consist of five layers (see fig. 1): instrumentation, agent, management logic, database and user interface. The functionality of two the bottommost, the first prototype is concerned about, is as follows:

1. The **instrumentation layer** is expected to expose devices' parameters' values to the outside world in a standardised way. The key technology chosen for that purpose is JMX, which defines an interface exposed by monitoring and management components. The instrumentation of a given resource is provided by one or more Management Beans, or MBeans, which are called either standard or dynamic. Standard MBeans are Java objects that conform to certain design patterns derived from the JavaBeans™ component model, while the dynamic MBeans conform to a specific interface, which offers more flexibility at run-time. The first prototype uses dynamic MBeans to instrument the monitored devices.



**Fig. 1:** Jiro based grid infrastructure monitoring system architecture.  
MB – Management Bean, C – connector, PA – Protocol Adaptor.

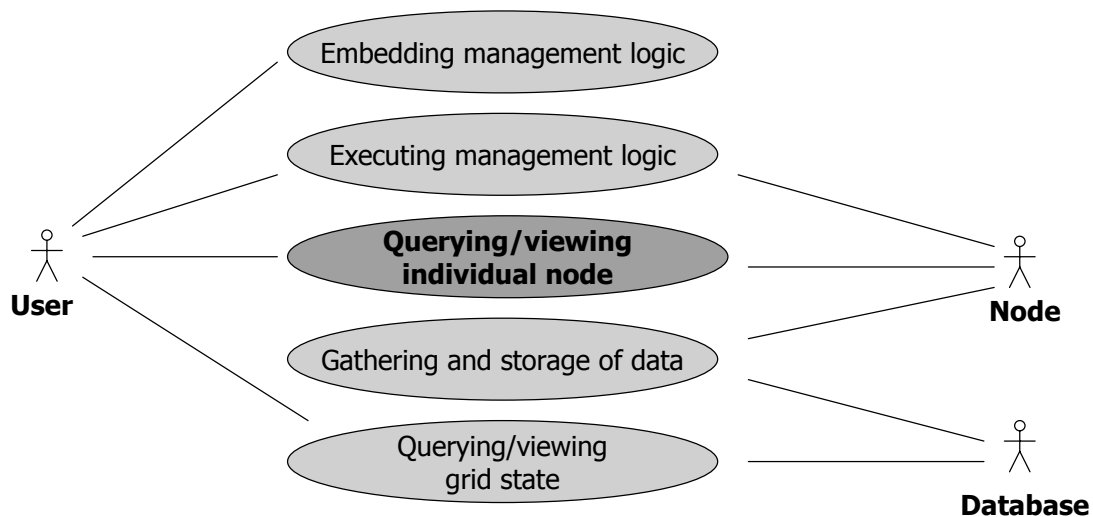
- 
2. The **agent layer** provides means for upper layers to communicate with the instrumentation layer. This functionality is also covered by JMX implementations, for example the Java Dynamic Management Kit™ (JDMK). The key components of this layer are management agents, which control the resources directly and make them available to remote management applications. Although it is not necessary, the agents are usually located on the same machine as the resources they control. Running an agent on the monitored system is impossible especially in the case of network components, such as routers or switches. Therefore, the networking devices will be monitored by MBeans located on a machine that can access the monitored ones via SNMP. As in the case of the instrumentation layer, the interfaces of monitoring agents are defined by the JMX specification. That makes it easier to integrate the bottommost layers with external database systems. Currently, the integration with OGSA-compliant implementation of R-GMA is under consideration.

The first prototype consists of the first implementation of the two described layers as well as of some preliminary user interfaces.

The functionality of the other layers can be found in the Jiro Based Grid Infrastructure Monitoring System Design Document [2].

#### 4. PROTOTYPE FUNCTIONALITY

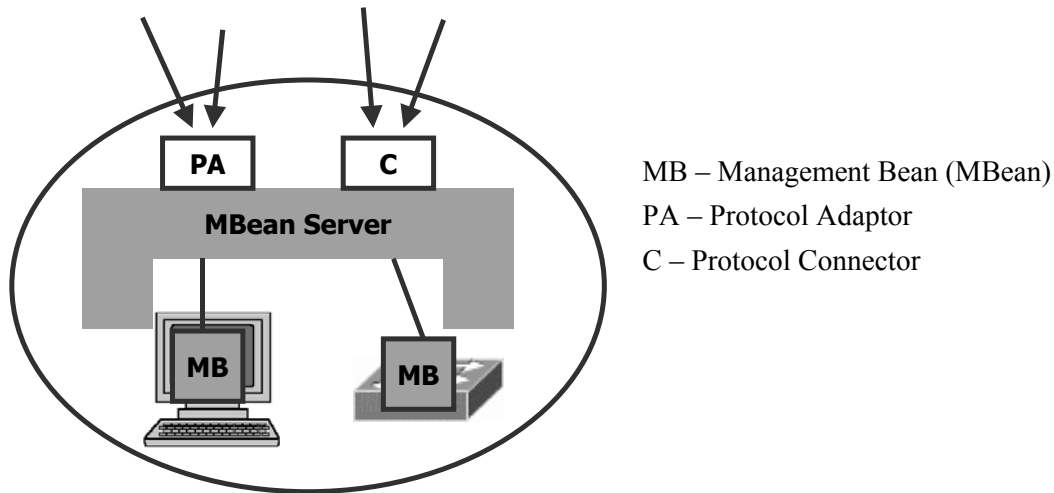
The system's first prototype functionality is mainly concerned about the instrumentation of resources and dynamic deployment issues, which are crucial for the usability of the software. The functionality of the monitoring agents, the instrumentation and agent layers consist of, was provided by using JMX and Jiro technologies. The first prototype aim was to enable the user to view single node's parameters, which is one of the five use cases (see fig. 2) specified in the Design Document [2].



**Fig. 2:** The use cases of the monitoring system and the one chosen to be implemented by the first prototype.

The first prototype of the system consists of implementation of instrumentation and agent layers as well as some preliminary user interfaces, which will be described in section 5.3. The monitoring agents, which are basic building blocks of the system, are implemented according to the JMX rules. The structure of a monitoring agent is depicted in Fig. 3.

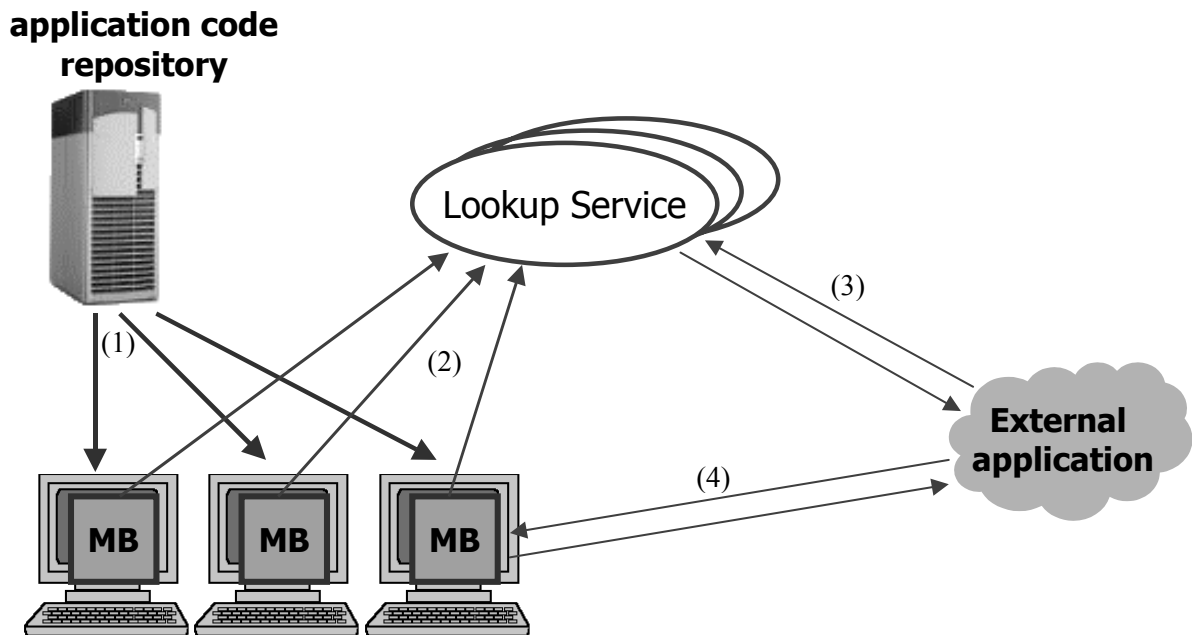
Each of the monitoring hardware pieces is presented to the agent as an MBean. If the monitored entity cannot run Java Virtual Machine and Jiro Deployment Station (such entities are, for example, network appliances, such as routers and switches), their MBeans can reside on other machines and query the state of target devices by SNMP. The MBean Server, which is a core part of the agent manages communication of the registered MBeans with entities external to the agent, either local (residing on the same machine) or remote. The remote entities can interoperate with MBeans via protocol adapters (e.g. HTTP protocol adapter) or protocol connectors (e.g. SNMP connector).



**Fig. 3:** The structure of a monitoring agent.

One of the most valuable system features is the ability to dynamically deploy and revoke monitoring agents. Such an approach makes the management of agents much easier. The actions taken by the system when deploying an agent are depicted in Fig. 4.

First, the monitoring agents are deployed by an application repository client on target devices. During their initialisation, the agents register in an instance of Lookup Service. Since there can be multiple instances of Lookup Service and Jiro technology takes care of synchronising their contents, the service does not form a “single point of failure”. Then, an external application that wants to use the agents, obtains their references from the service and calls their (standardised) interfaces directly.



**Fig. 4:** The actions taken by the system when deploying new monitoring agents and by an application intending to query an agent.

## 5. USER MANUAL

### 5.1. DEPENDENCIES

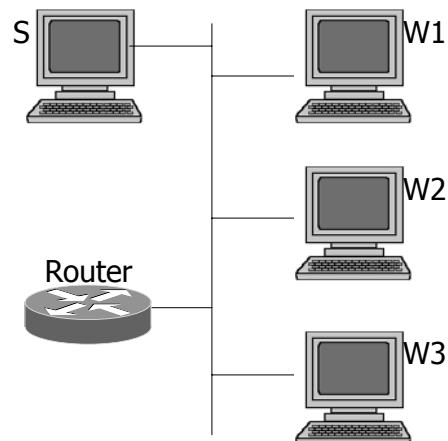
The prototype has been tested to work properly in the following environment:

- hardware: PC machines with Red Hat Linux 6.2,
- software:
- Java Runtime Environment (JRE) version 1.3.1\_04 or 1.3.1\_06,
- Jiro version 1.5.1,
- Java Dynamic Management Kit version 4.2,
- OKI Service Finder version 1.1,
- BeanShell version 1.2b6.

### 5.2. INSTALLATION

Assuming the network topology depicted in Fig. 5, in order to install the monitoring system prototype the following steps need to be performed:

- installing and starting the Jiro domain on S (a machine selected to host the Jiro domain),
- installing and starting the Jiro Deployment Station on W1, W2, W3,
- deploying and instantiating the JiroMBeanServer on W1, W2, W3,
- configuring the JiroMBeanServer on one of the workstations to monitor router through SNMP.



**Fig. 5:** An example of network topology the system could be run on.

#### Step 1: Installing and starting the Jiro domain

The installation of Jiro domain can be performed by using an installation program provided with Jiro 1.5.1 implementation.

Starting the Jiro domain can be conducted by using the “igniter” tool. Full installation and configuration instructions are provided in [4]. After starting up a domain, an instance of the Jini Lookup Service is running on S.

## Step 2: Installing and starting the Jiro Deployment Station

The Jiro Deployment Station has to be installed on each host that is to be monitored. The deployment station is started using tool called "DeploymentStationStarter" provided with Jiro distribution.

### Step 2a: Starting the rmid daemon

Since the Jiro Deployment Station is an activatable service, the RMI daemon, rmid, must be started on each workstation (W1, W2, W3) in order to run the Jiro Deployment Station. To start rmid on Linux, issue the following commands:

```
export LD_LIBRARY_PATH=/home/infmon/lib
rmid -J-Djava.security.policy=rmid.policy -log logRmid&
```

The LD\_LIBRARY\_PATH system variable should point to the directory containing native monitoring libraries provided with distribution. The -J-Djava.security.policy parameter specifies policy file that is used to verify whether or not program is allowed to launch a JVM for an activation group.

Permissions have to be granted **explicitly**. Following is a sample policy file (full documentation of rmid parameters can be found in rmid documentation):

```
grant {
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Djava.security.manager=*";
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Djava.security.policy=*";
};
```

### Step 2b: Starting the Jiro Deployment Station

The DeploymentStationStarter is a setup program which registers information about the Jiro Deployment Station with the RMI daemon. To run this program following command should be executed:

```
DeploymentStationStarter [impl_jar_file] [dl_jar_file]
[station_config_file] [security_policy_file]
```

where:

*impl\_jar\_file* – location of the depStation.jar (provided with distribution);  
this file will be downloaded by a spawned JVM  
to construct Jiro Deployment Station,

*dl\_jar\_file* – URL to station-dl.jar (provided by Jiro), this file will server as a codebase argument of  
the spawned JVM,

*station\_config\_file* – Jiro Deployment Station configuration file,

*security\_policy\_file* – policy file which will be used by spawned JVM.

An example command could be as follows:

```
java -jar depStation.jar
"/home/infmon/ClusterMonit/DeploymentStation/depStation.jar"
"http://localhost/~infmon/jiro1.5/station-dl.jar"
station.config
"/home/infmon/.java.policy"
```

A sample station.config file follows:

```
domain=jiro:InfMonit
name=station1
role=shared
manufacturer=agh
vendor=agh
version=1.0
model=0
serialNumber=1.0.0.0
iserver_classpath=
iserver_port=8067
store=station_log/station.ser
service_inventory_store=station_log/si.ser
data_manager_setup_string=station_log/state_server/state_server.ddm
jar_location=station_log/jars
config_class_name=com.sun.fma.deployment.DeploymentStationConfigurat
ion
implementation_jar=station-impl.jar
dl_jar=station-dl.jar
```

The distribution comes with a generic configuration file, so there is no need to create one from scratch. The most important settings are the paths for the station to store its files.

The Deployment Station Starter produces the following output:

```
Got the stub for DeploymentStation
Activating service...
DeploymentStation address:
javax.fma.common.StationAddress(name=station1,manufacturer=agh,
vendor=agh,version=1.0,model=0,serialNumber=1.0.0.0,domain=jiro:InfM
onit,role=shared)
Queris lookup for deployment station
not found... retrying...
FOUND, ServiceID: f20b0075-cb02-42e7-ad2e-bc4952e6f0d3
Service was successfully created and registered in the Lookup Service.
```

Fig. 6 depicts the sample network scheme with the basic Jiro services installed. Note that if the administrator wants S to be monitored also, he should install the Jiro Deployment Station there as well.

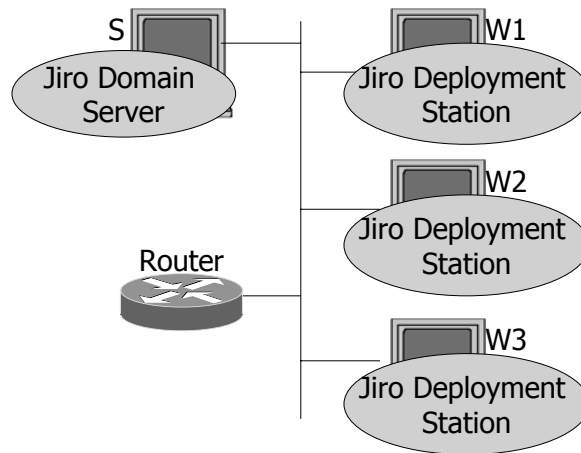


Fig. 6: The sample network after completing step 2.

### Step 3: Deploying and instantiating the JiroMBeanServer

The user had to perform the previous steps on each host separately. From this point, the user is able to automate operations using ExtFinder and BshConsole. By using the BshConsole in the ExtFinder user is able to push JAR files into the Jiro Deployment Station, and to create instances of services.

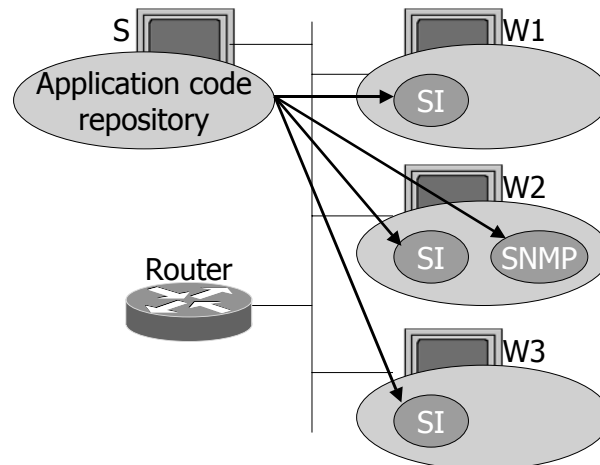
User has an option to use helper BeanShell functions for deploying and instantiating of Jiro services. These functions have following semantic:

```
jiroDeploy(station, implJar, dlJar)  
jiroNew(station, className)
```

where:

*station* – reference to Jiro deployment station,  
*implJar* – path to the implementation jar,  
*dlJar* – path to the downloadable jar,  
*className* – name of the class.

After completing the deployment of MBean Servers, which contain MBeans, the user is able to view states of the monitored nodes either by using a client started from the ExtFinder or by using Web browser. This state is presented in Fig. 7. Note the assumption of S being also the code repository for the system. It is not necessary for the system to operate properly, but convenient for system administrator.



**Fig. 7:** The sample network with MBeanServers containing deployed MBeans. SI – SystemInformationMBean, SNMP - SnmpMBean

#### **Step 4: Configuring the JiroMBeanServer to monitor a networking device through the SNMP protocol**

In order to monitor a networking device (e.g. router) through SNMP protocol, the user must create an instance of SnmpMBean. Any MBeanServer that has network access to the router can host this MBean, e.g. W2. The instantiation can be performed using HTML protocol adaptor. After creating SnmpMBean user must configure it by providing address of the SNMP agent. After this step user is able to query the MBean for the state of the selected device.

### **5.3. RUNNING**

This section contains description of user interfaces available in the system prototype. In order to clarify descriptions screenshots are also provided.

#### **5.3.1. ExtFinder**

ExtFinder is a universal tool for browsing Jini Lookup Registry (ServiceRegistrar). It is based on Finder from OKI Lab. Fig. 8 shows the ExtFinder's main frame. The top of the main frame is a condition panel. As tabs show, this condition panel consists of three subpanels: „Registrars”, „Service Types” and „Attributes”. Each of these subpanels provides an UI to specify conditions to find expected Jini services. The „Registrars” Panel searches for services based on the Lookup Service instances that are searched for at the application startup, „Service Types” performs the search based on interfaces implemented by services and the „Attributes” tab allows to search for all services registered with any of the locators that have the specified value of a particular attribute.

The lower section of the finder frame shows a set of services that resulted from lookup operation. Since the lookup is performed every time the user changes the conditions, and as a response to remote events signalled by Lookup Services, this panel contents should be always up to date.

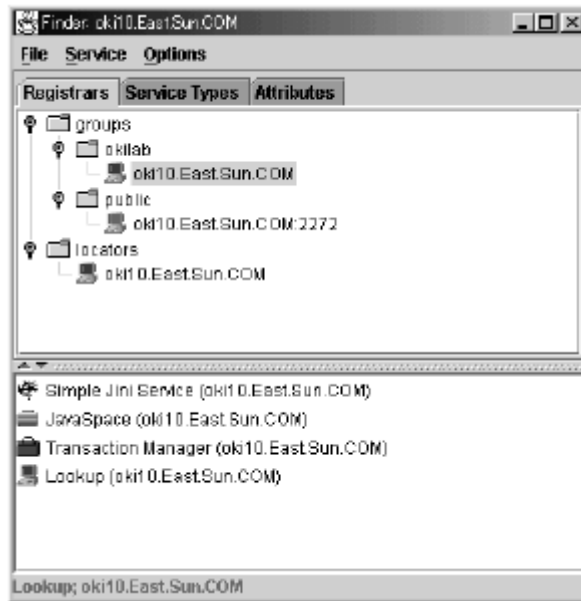


Fig. 8: The ExtFinder's main window.

### Extensions to the original Finder

The user is able to enable finder extension by pressing „Options”→„BeanShell” menu. That results in displaying Bsh console windows. The user is now able to use bsh set command (pop-up menu on service pane) in order to have a BeanShell local variable bound to a remote service (e.g. Deployment Station) and will be asked for variable name that will be created in the BeanShell environment. After doing that, the user is able to invoke service's methods. The user can also select multiple services. They will be inserted into the BeanShell environment as an array of services (this allows for deployment on large number of nodes). The process of binding a service to a local variable is presented in Fig. 9.



Fig. 9: Binding a remote service to a local BeanShell console variable.

After binding a Deployment Station(s) to local variable(s) the user can invoke the jiroDeploy() and jiroNew() methods described earlier in this document.

### 5.3.2. JiroMBeanServiceUI

After successful instantiation, each of the JiroMBeanServer instances registers itself in the Lookup Service. Then it becomes visible to the user as an entry in the lower panel of ExtFinder. Then the user is able to launch a client that connects to the JiroMBeanServer using RMI and is able to show the returned values either in text form (all parameters) or as a chart (selected parameters). The client

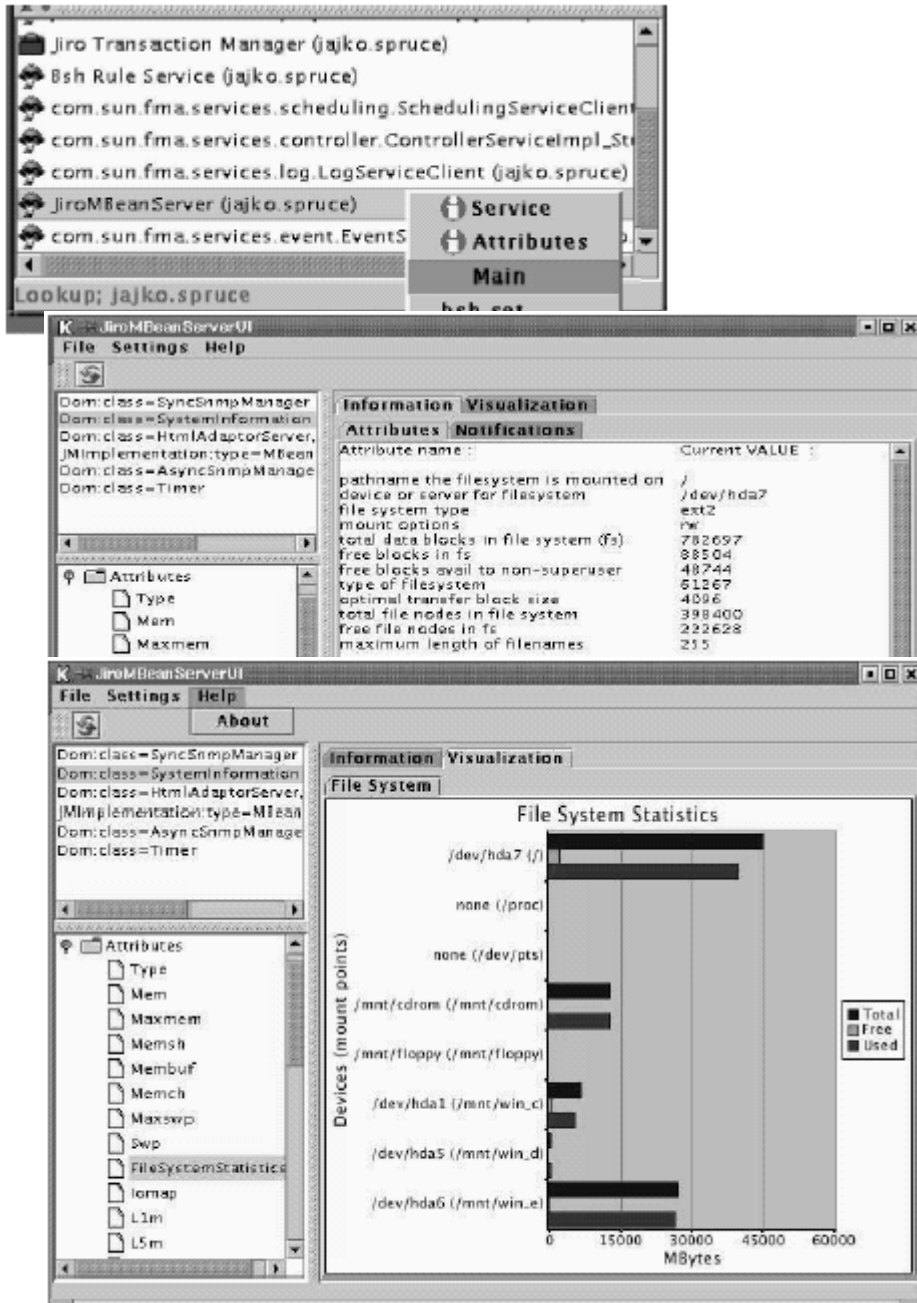


Fig. 10: Starting the JiroMBeanServiceUI client.

window is presented in Fig. 10.

#### **5.4. INTERFACE DESCRIPTION**

The interface of the MBean Server and dynamic MBeans, which are the basic building blocks of the first prototype, is specified in [4]. Each MBean Server is equipped with an RMI connector that is responsible for communicating with external entities.

## **6. INTERNAL TESTS**

Basic system prototype startup and simple communication tests have been conducted. The test procedure is described in [5].

## 7. ISSUES

- 1) There are some problems with running the Jiro “igniter” tool on Linux implementation of JDK 1.4 – the application starts, but it takes even 10 minutes(!) to run the platform. Although it does not happen often, we recommend using JDK 1.3. With that platform the process takes less than 30 seconds.