

JMX-based Grid Management Services

Kazimierz Balos, Krzysztof Zielinski

Department of Computer Science, AGH University of Science & Technology, Al. Mickiewicza 30,
30-059 Krakow, Poland, e-mail: {kbalos, kz}@agh.edu.pl

Abstract— The paper presents Global Discovery Service (GDS), configuration and deployment tools for Grid infrastructure and advanced monitoring services implemented with JMX technology. The GDS service is designed to work over firewalls and could be used not only for resource discovery but offers also mechanisms to take control over resources hidden by firewalls. It creates alternative to SOAP Gateways based on Web Service. The described services have been designed to be scalable and lightweight. They have been put into service within JIMS – Grid infrastructure monitoring system implemented under CrossGrid IST Project.

Index Terms— Grid, JMX, management, discovery

I. INTRODUCTION

JMX (Java Management Extensions) [1] is going to play a very important role as a general framework for distributed resources management. It offers general architecture suitable for construction of monitoring systems and management applications written in Java. It is very important in context of the current trend manifested by wide acceptance of open standards, such as Java and Web Services, for Grid middleware construction. The spectrum of JMX applicability covers among others grids systems [7], [8], active networks [6], and mobile systems. Management of Grid resources over Internet requires their dynamic discovery, configuration management and monitoring at the first place. The built into JMX implementation standard discovery services do not work over firewalls. This is a basic constraint that reduces a scope of JMX applications over public networks. Full exploitation of JMX functionality in Grid systems is influenced also by implementation of scalable and easy to use configuration and deployment tools. Finally advanced grid services such as load balancing, or resource brokering require monitoring of more complex metrics such as data transfer time between selected network nodes, delay or transaction throughput.

Design of JMX-based monitoring and management system results from requirement of more portable and flexible system construction than MDS version 2 (Monitoring and Discovery System) [17], which is the part of Globus Toolkit 2, used in both the DataGrid and Crossgrid project. There are several

reasons for developing a new framework for grid resources management and monitoring. The most important are that existing system (MDS) is more CE-oriented (CE – Computing Element) and doesn't provide detailed information about all worker nodes (WNs) and detailed state of computational infrastructure. There are also other monitoring services derived from DataGrid project, which give similar functionality, but all of them are limited by following issues:

1. they are tied to certain version of operating system (RH Linux 7.2),
2. they give overall look at the Grid as a set of CEs and SEs,
3. they store mainly static information and many preconfigured parameters due to the usage of LDAP as storage for monitoring information,
4. information collected in long periods of at least 1 second (typically 300[s] or 600[s]),
5. reconfiguration typically requires restarting the whole monitoring system.

JIMS has following advantages comparing to the monitoring system from Globus:

1. it is installed on every WN and provides detailed information about CPUs' utilization (current and average from last seconds and minutes), as well as other information about memory and network state,
2. it has the ability for autoconfiguration, and adaptation to the operating system and kernel version. Autoconfiguration concerns WN in clusters and clusters in Grid while adaptation concerns the version of IP protocol (IPv4 vs. IPv6), version of kernel (2.4 vs. 2.6) and many others,
3. JIMS is the real on-line management tool with ability to add or upgrade the functionality without restarting the whole monitoring system,
4. it supports hierarchical information structure, designed to be fast (response time in the order of 100[ms]), scalable and lightweight,
5. JIMS is implemented using open source software as AXIS [16], JMX RI [1] and SNMP [5] as base software components, and is exposing monitoring information as Web Services.

Besides the presented characteristics, monitoring system as JIMS, built on the top of manageable Java objects (MBeans), introduces new approach for monitoring and management systems construction which benefits from powerful connector

architecture integrated with JMX, providing transparency of managed resources from the client point of view.

The main goal of this paper is to present Global Discovery Service (GDS), configuration and deployment tools for Grid infrastructure and advanced monitoring services implemented with JMX technology. The proposed GDS service is called global because it has been designed to work over firewalls. The proposed service could be used not only for resource discovery but also offers mechanisms to take control over resources hidden by firewalls. In the design and implementation phases every measure has been taken to make these services scalable and lightweight. They have been put into service within JIMS – Grid JMX Infrastructure Monitoring System implemented by CrossGrid IST Project [10].

The paper is structured as follows. In Section 2 architecture of JIMS has been shortly described. Next in Section 3 GDS concept and architecture have been specified. In Section 4 deployment and configuration services of JIMS exploiting JMX mechanisms are presented. Some aspects of monitoring or value estimation of Grid system load metrics, already supported by JIMS, have been described at the end of this section. Section 5 contains sequence diagrams of GDS activity and some implementation details. Finally in Section 6 case study of the GDS and JIMS applications have been presented. The paper is ended with conclusions.

II. JIMS ARCHITECTURE

The grid management services presented in this paper have been constructed in accordance to contemporary trends in distributed systems architecture expressed by Service Oriented Architecture (SOA) [6]. Systems of that class are decomposed into smaller components responsible for providing functionality of particular services. Because the adaptability and fault tolerance of large-scale networked distributed systems is of high importance, the components of a system can migrate or be duplicated. Moreover, the environment, in which a distributed system is operating, can have a fluctuating nature. Since that, large distributed systems have to be able to adapt to changing network conditions.

The aforementioned issues impose new requirements on the underlying architecture. The most important of them are effective coupling of internal system services, describing information flow, implementing communication channels, maintaining security, etc. The services a SOA-compliant distributed system is built from are expected to support introspection, be discoverable, loosely coupled and platform, location and transport independent. There are a number of middleware platforms that help to cope with the requirements. A good example of such platform is Web Service (WS) providing access to the system functionality built with Java Management Extensions (JMX) support [11].

A system for grid resources management should have scalable and flexible architecture, easy for development and maintenance. JIMS, the JMX-based Infrastructure Monitoring System – developed by the DSRG at AGH-UST, meets the requirements for monitoring systems operating in distributed environment, and supports resources abstraction, dynamic system configuration and interoperability. Being based on SOA principles, it makes use of modern technologies and solutions, such as JMX, Web Services, dynamic discovery and automatic configuration.

The JMX architecture consists of three levels: instrumentation, agent, and distributed services. The current version of the JMX specification [1], [2] addresses the first two levels and provides only a brief overview of the latter. JMX provides interfaces and services adequate to monitoring and management systems requirements [3]. This functionality involves abstracting resources by using components called MBeans (Managed Beans) and remote instrumented resources accessibility through JMX connectors. The functionality developed by JIMS project exploits dynamic discovery of monitored resources. In order to achieve more flexibility and interoperability, Web Service provides monitored objects' proxies that are used to implement lightweight clients. The architecture of JIMS is shown in Fig.1. The monitoring system is decomposed into the following layers:

1. resources instrumentation layer (JMX MBean Servers, SNMP, RMI),
2. interoperability layer (SOAP Gateways, Web Services),
3. integration layer,
4. interface and presentation (GUI: web applications and standalone visualization tools, CLI applications, Java API).

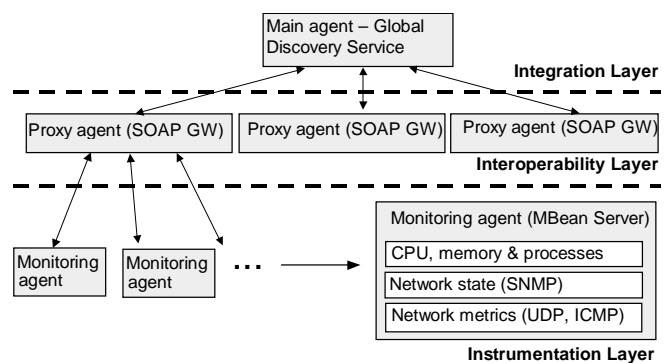


Fig. 1. Layered architecture of monitoring and management system and the modules of JIMS.

The resources instrumentation layer is responsible for delivery and management of information from monitored resources. Its functionality includes reading and writing attributes of monitored components (MBeans), performing measurements (for example, network latency and throughput) and sending notification triggered by pre-programmed conditions.

The instrumentation layer obtains information from

monitored resources using the following mechanisms:

1. virtual file system (/proc) and Java Native Interface (JNI) to access operating system parameters,
2. SNMP agent running locally on monitored system (usually on each Worker Node in a site). These agents communicate with monitoring system through SNMP protocol using SNMP API [5] to be accessible from Java,
3. Network Metrics module, measuring network condition using standard protocols like ICMP and UDP.

The information from the instrumentation layer is made available for further use by JMX RMI connectors, which connect Monitoring Agents with SOAP Gateways, which build up the interoperability layer of the proposed architecture.

SOAP Gateways act as translators between Java RMI and SOAP and are implemented as AXIS-based Web Services, exposing all monitored parameters in a uniform way. It is important that all monitored agents can be accessed directly using RMI and SNMP protocols or alternatively through SOAP, which is the preferred protocol for outer applications and visualization tools. SOAP Gateways play also an important role in dynamic configuration of the local system and discovery of its entities, because they are responsible for finding all currently running monitoring agents. The integration layer performs global discovery and keeps track of all currently running SOAP Gateways. The proposed hierarchical architecture is suitable for large distributed systems and offers required scalability. JIMS layered architecture will be further elaborated in the following sections.

III. GLOBAL DISCOVERY SERVICE

The typical configuration of Internet firewalls allows communication via HTTP or HTTPS protocol under the condition that this communication is initiated from a secure region to a public network. This asymmetry creates the additional difficulties not only in JMX discovery services activity but also in standard management operations invocation.

During the design phase of GDS the following major requirements have been taken into account:

1. the service should allow management and monitoring resources hidden by firewalls using only HTTP/HTTPS connections. It means that operation invocations from management application should be possible on MBeans located in the secure region,
2. it should allow discovery of resources and accommodation to their temporal accessibility,
3. the service should also provide mechanism supporting localization of resources in a network topology. This mechanism will be used for the neighborhood discovery and available resources visualization,
4. the implementation should be lightweight and scalable and should be fully compliant with services already defined within JMX framework.

Design phase of GDS disclosed a few problems that have to be resolved. The most important problem is to establish connection to MBean Server located in a secure region. The reason is that communication must be initiated from the secure region. To solve this problem a new element called Join Agent has been introduced. Function of this agent is to initiate communication connection with management application from secure region on one side and with MBean Server on the other. To discover MBean Server in the secure region Join Agent may use the standard JMX Discovery Services. The proposed solution has been depicted in Fig.2.

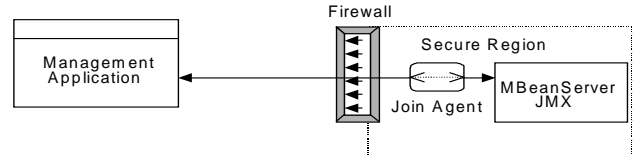


Fig. 2. Connection establishment from secure region [4].

This solution has three disadvantages:

1. It is impossible to discover of MBean Servers by Management Application what is the major functionality of GDS.
2. It is very difficult to choose a good strategy when Join Agent should establish connection to Management Application – there is no information when this application has been started.
3. It is necessary to inform Join Agent in some way where Management Application has been started. This is in particularly complicated in case when more than one Management Application is started in different localizations.

To eliminate these drawbacks the second element called Mediator Agent has been introduced. It is placed outside the secure region (or in the same region as Management Application) and by assumption is active all the time. The activity sequence in this case is as follows: first Mediator Agent is started, next in any time MBean Server via Join Agent could establish communication with Mediator Agent. The Management Application (one or more) may also open communication with Mediator Agent obtaining this way access to all already connected MBean Servers. The proposed solution has been depicted in Fig. 3. It is free of drawbacks of the previous solution and provides:

1. discovery of MBean Servers connected to Mediator Agents – as will be shown later in this section,
2. establishing communication in any time because Mediator Agents is always active so the Join Agents can access it,
3. easy finding of Mediator Agents by Join Agents because localization of it could be fixed under well know address.

Mediator Agent could be seen as single point of fault. This problem could be solved by replication but authors don't want to discuss it further in this paper. Components of GDS system are shown in Fig. 3.

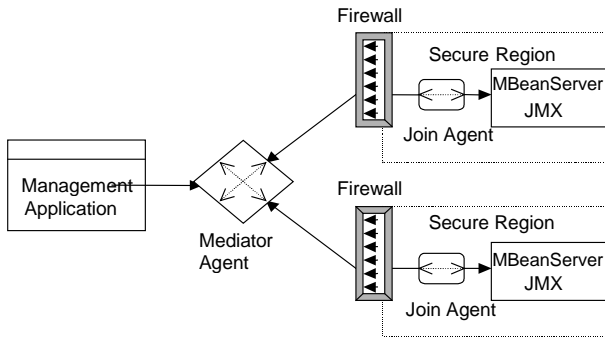


Fig. 3. GDS system elements [4].

IV. MANAGEMENT SERVICES DEPLOYMENT AND CONFIGURATION

JIMS places big emphasis on installation and configuration process, which allows to automatically deploy and upgrade management modules. Deployment of grid management services consists of two phases:

1. the installation, configuration and startup process of JIMS monitoring system,
2. global configuration of running JIMS instances including initialization of Join Agents and registering JIMS instances in GDS.

Deployment and configuration of JIMS and GDS system is based on standard techniques taken from reference implementation of JMX [1], [2] built by SUN Microsystems (dynamic on-demand instrumentation layer and M-Let service), and some mechanisms, like discovery, and dynamic auto-configuration.

A. Dynamic Deployment

JIMS system startup process relies on mechanisms for on demand software loading to dedicated system agents. JMX technology supports such functionality with its M-Let (dynamMic appLet) service. The service provides online loading and installation of Java classes. Usage of the service in JIMS is depicted in Fig. 4.

Using the dynamic loading service, monitoring agents are automatically installed and started in a cluster. Management station starts all pre-installed monitoring agents using ssh remote shell. Next, started agents download and install monitoring modules, which are components of the instrumentation layer. It is important that these modules can be deployed (downloaded, installed and started) automatically at start-up time or at any time later. It is possible to upgrade or install newly developed modules as well as removing existing one without restarting the whole monitoring system.

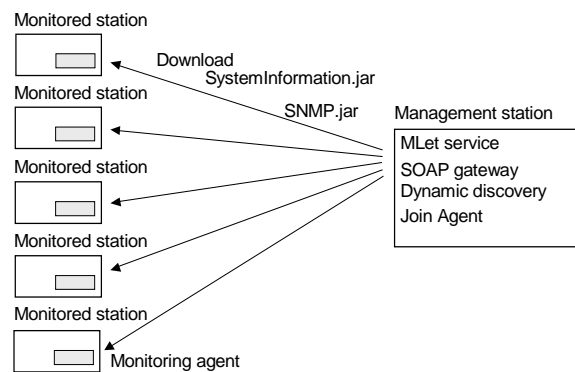


Fig. 4. JIMS components download and installation.

B. Interoperability

Interoperability could be defined as possibility of remote operations invocation in a system implemented in different technology than a client is constructed with. In more wider sense it could be seen as enabling technology which overpass exiting shortcomings with getting access to a service functionality. JIMS systems offers two solutions addressing this problem: GDS already described in previous Section and SOAP Gateway.

The SOAP Gateway (SG) concept is based on general approach described in OGSi (Open Grid Services Infrastructure) specification [7], where grid service is exposed as WS defined using WSDL (Web Service Definition Language), conforming to a set of conventions (interfaces and behaviors) that specify how clients and services interact [12]. The SG concept is also based on architectural approach taken from OGSA (Open Grid Services Architecture) [8]. Just as in case of other grid services, the layer of interoperability for infrastructure monitoring system should support transient service instances, created and destroyed dynamically, and give unified way to access all monitored resources. SG allows hiding the complexity of managing monitored stations and exposes interfaces consistent with other grid services. Because clusters in grids consist of many monitored computing elements, the interoperability layer should also perform the role of router, forwarding requests from one outer point of communication to the specified node. To achieve this goal it should store addresses of all available monitored stations. In big installations with hundreds of nodes, administrative assigning of RMI address of each monitored station in SG would be clearly ineffective. To solve the problem of registering new stations appearing in a cluster with SG, as well as deleting inactive ones from the registry, a mechanism of active stations discovery is used. The proposed interoperability layer assumes one SOAP Gateway per cluster. In some cases

SG-s can be doubled for fail-over purpose.

SG resides in a multiprotocol environment, i.e. with SOAP at the external system side, and Java RMI at the side of monitored stations. Because of that, it should perform a role of translator, connecting itself to monitored nodes through RMI connectors and to client applications through WS.

Summarizing, the interoperability layer supports:

1. automatic installation to facilitate management of numerous nodes in clusters,
2. automatic configuration with dynamic discovery mechanisms for finding monitored stations that are currently available and heart beat mechanism for removing stations that do not operate properly and are not responding for a certain period of time,
3. self-adaptation mechanism (dynamic discovery and heart beat) from the user,
4. exposing one point of communication through one - due to firewalling - well defined address and port, with WSDL describing its functionality,
5. forwarding requests from WS clients to the specified monitored stations [9].

With comparison to standard SG approach the GDS service offers the interoperability solution dedicated only to JMX enabling this technology working over firewalls. It will be further described in Section 5 covering implementation aspects.

C. Automatic and Dynamic Configuration

SG autoconfiguration is based on two mechanisms applicable over cluster: dynamic discovery and heartbeat. First mechanism uses Discovery Monitors at the side of SG and Discovery Responders in monitored stations in order to provide Active Discovery in much the same way as in JDMK [3]. SG periodically sends multicast requests to all monitored stations, and they respond with their RMI addresses of JMX connectors. The second mechanism, heartbeat, is a complementary process to discovery mechanism and is used for finding monitored stations that do not respond for some reasons. If a station is not responding repeatedly for a certain number of times, it is removed from the SG registry and is no longer available. For this purpose each station registered in SG has its own counter of retries which is started after first access failure.

As it can be expected, SG requires very little logic on the client side of application, because whole logic can be hidden behind the interoperability layer. It encapsulates the complexity of discovery and heartbeat mechanisms. The advantage of using SG as the point of access to monitoring data is location transparency of monitored resources. Each change of the monitoring station (vanishing or changing JMX RMI address – due to MBean server restart or physical crash) is handled by a SOAP Gateway, so the client application each time obtains proper list of valid and active monitored resources [10].

D. Monitoring and QoS Estimation

Presented grid management services encompass grid infrastructure monitoring and QoS estimation. These two functions present only one of possible applications of dynamic JMX-oriented services and provide information for other grid components, like resource broker, network prediction component, postprocessing and aggregation of collected information, benchmarking and grid performance evaluation.

To assure the required level of services offered by grid networks it's necessary to measure and evaluate the free resources available for users. Presented system is equipped with such functionality, enabling grid components to measure free resources in following domains:

1. CPU statistics for every WN (Worker Node),
2. memory statistics,
3. filesystem statistics for huge storage media,
4. network statistics.

Information provided by instrumentation layer is delivered in the on-request manner (like CPU load or free memory). This information could be exploited by a grid system resource allocation component such as Resource Broker.

The JIMS unique feature is provisioning some estimation of QoS networking parameters such as throughput or delay. These parameters could be used for on-line decision about where replicated data should be accessed from or how the computations should be distributed. It is accomplished on-demand or periodically by dedicated service performing packets delay measurements using different protocols (ICMP and UDP) and throughput estimation between different WN in the same cluster and between WN in different clusters as well, what is required by larger grid installations.

V. IMPLEMENTATION ASPECTS

Described services were implemented in one, coherent technology, using Java platform, JMX and its Remote API. This approach allows managing remote resources in a unified way, without changes to JMX API. JIMS adds the instrumentation layer for grid networks management and QoS estimation, while GDS provides configuration of different sites using global registry. Implementation details of JIMS' Instrumentation Layer and SOAP Gateway have been described in [9], so in this section only Global Gateway implementation will be elaborated.

A. Global Gateway

Global Gateway is the registry for SOAP Gateways. This registry could be updated referring to UDDI where SGs are typically registered or exploiting GDS. UDDI is a registry for WebServices and typically operates on chosen CE (Computing Element) in cluster. In the second case Global Gateway acts also as a proxy between clients and Mediator Agent enabling the client applications to communicate from the outside world to the secure region. To finally clarify the concept of GDS

implementation it is necessary to explain how the operation invocation could be forwarded from Mediator Agent to Join Agent by the connection opened in the opposite direction. JMX notification mechanism [1], [14], is employed for this purpose. Join Agents is implemented as MBean, which is registered with Mediator Agents. Mediator forwards operations' invocations encapsulated in a notification events. The internal structure of the Mediator Agents has been depicted in Fig. 5. It is determined by Mediator functionality, which may be summarized as follows:

1. allows connection of Join Agents and Management Applications – connector servers have to be installed for that purpose,
2. forwards management operation invocation to suitable Join Agents,
3. makes possible searching for connected agents with Discovery Service.

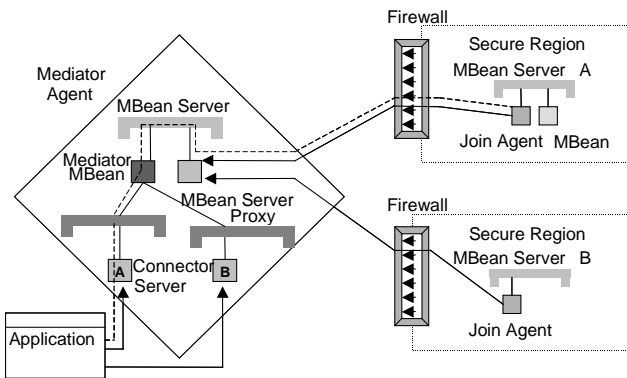


Fig. 5. Mediator Agent internal structure [4].

Remote communication with HTTP/HTTPS protocol in JMX is supported by connector server/client objects. JMX connector construction assumes that during operation invocation is specified only a MBean name and is not specified MBean Server identifier. It creates a conflict when the same name is used by different MBeans. The solution is to create a separate connector server for each Join Agent. The connectors should be registered with Mediator MBean which forwards invocations to Join Agents. Unfortunately it could not be done directly because connector server has to be registered in JMX server to perform its activity.

Creation of MBean Server for each registered Join Agent is not a scalable solution, because MBean Server is rather complex and heavy object. In this situation authors decided to construct MBean Server Proxy object. It has the same interface as MBean Server but its functionality is limited only to forwarding operation invocations to Mediator MBean. The connector server is registered with MBean Server Proxy first. The presented construction resolves the problem of addressing.

Mediator Agents has only one MBean Server where Mediator MBean and the connector server providing communications from Join Agents are registered. The sequence diagram of the Mediator Agent activity has been

presented in Fig. 6. During the initiation phase Join Agent is created, it connects to Mediator MBean in Mediator Agent what results in the connector server creation. Next the connector server is started and confirmation is sent back to Join Agent.

During the operation phase any operation is performed on the connector server of suitable MBean Server. The connector performs *sendNotification()* operation on Mediator MBean which notify suitable Join Agent. Join Agents extracts the operation invocation from the notification events and invokes an operation on MBean Server.

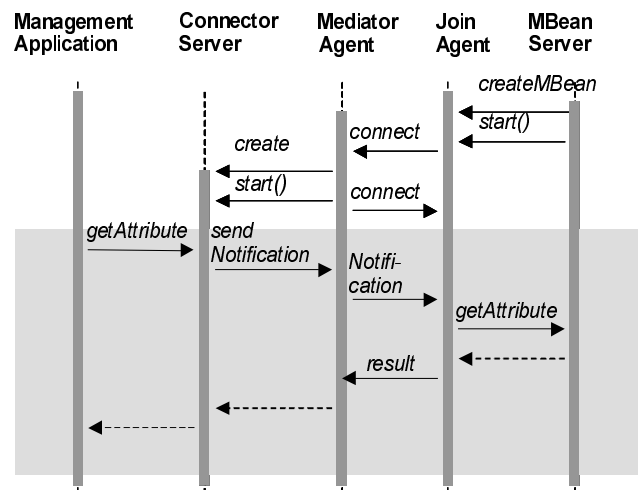


Fig. 6. GDS activity sequence diagram [4].

Described architecture solves the problem of resources management in secure regions but doesn't explain how the newly created MBean Server proxy objects are discovered and registered by management application. Assuming public character of network where agents are running, there should be used better mechanism for agents discovery than available in Java Dynamic Management Kit - JMDK from Sun Microsystems [3]. The main issue of standard dynamic discovery is its local character due to multicast transmission used for sending requests to monitored agents. To overcome this problem in GDS there is implemented an improved version of discovery service. Multicast transmission can be stopped by routers between the management application and agents, so there is possibility that there can be discovered another new agents which were not available for discovery from management application using standard methods. Global Discovery Service is based on cascade discovery using already discovered agents. Firstly there is performed ordinary discovery using standard mechanism available in JMX. Having connection to limited number of agents, where some of them are Mediator Agents, there is performed further discovery using Discovery Clients installed on discovered agents. If Discovery Client is not installed, there is registered a new instance of it in discovered agent, which from this moment can search for its neighbors and report their JMX addresses to the management application. The process is stopped when there

aren't any other agents and the application has established unicast connection to all known agents, even to these hidden from the multicast transmission. This kind of discovery service is called global because it operates in networks where standard multicast discovery service isn't sufficient (i.e. grids) and where managed resources are hidden behind the firewalls or NAT (Network Address Translation) routers.

The last feature of presented system is its ability to display neighborhood relations between discovered agents. To achieve this goal there is performed special discovery procedure with TTL initially set to 1 and increased subsequently. If given agent doesn't respond with TTL equal to $n-1$ and responds with TTL equal to n there is probability that agent is in distance of n , where n is the number of hops (routers) between the client application and the agent. It can be used as a simple way for establishment of network topology of management agents, what can be used in environment where agents are hosted by mobile devices.

B. Join Agent

Join Agent is implemented as JoinAgent class with JoinAgentMBean interface, what means that it is a standard MBean and can be deployed in any MBean Server used to host other MBeans of managed resources. Before registering in MBean Server JoinAgent should be configured, i.e. at least its MediatorAgentAddress and MediatorAgentPort attributes should be set to point out to an existing and operating Mediator Agent.

C. Mediator Agent

The main classes of Mediator Agent of this package are:

1. MediatorAgent class, which includes MBean Server hosting HTML adaptor and HTTP connector MBeans; in this server there are registered also DiscoveryResponder and Mediator MBeans,
2. Mediator class, providing communication between management application and managed agents,
3. ProxyMBeanServer class, implementing MBeanServer interface; includes one connector-server, which sends all requests to Mediator, which in turn sends them to specified managed MBean.

MediatorAgent acts as a container for managed MBeans: HTMLAdaptor, HTTPConnector, DiscoveryResponder and Mediator, and doesn't perform any specific role for mediator agent module, hence the whole logic is encapsulated in Mediator class, which will be discussed now. Mediator handles the whole logic responsible for communication with management applications and Join Agents by creation MediatorConnector for each connecting JoinAgent, making them in this way available for outer management applications.

At first, JoinAgent is connecting in the standard way to HTTPConnectorServer running permanently in MediatorAgent. Next, invokes the *connect(Long seq, String mBeanServerId)* method, what entails creation one, additional

MediatorConnector. JoinAgent registers itself as a listener for notifications sent by Mediator. All management requests incoming to MediatorConnector are sent by the notifications to JoinAgent, which conveys them to its own MBean Server. JoinAgent receives notifications through its notification interface and processes them in *handleNotification()* method. The main purpose of this method is to translate incoming notification from Mediator Agent and performing adequate actions on managed MBeans. Because handling notification is the only way for communication with JoinAgent, *handleNotification()* is used also for some administrative purposes, like confirmation of connecting or disconnecting from given Mediator Agent.

After receiving and processing the notification message, JoinAgent responds to Mediator Agent invoking method *send(String action, Object[] params, String[] signature)* with action set to "result", and params and signature set to resulting values and their class names respectively. This involves invoking and passing results by the remote *result()* method of Mediator, what completes this process. Action field can be also filled with "connect" value, which is used during connection establishment.

Very simple and interesting logic due to its unconventional approach was implemented in MediatorConnector class, which is in fact custom implementation of JMX MBean Server. Its purpose was to avoid using normal and heavy standard MBean Server objects and to make kind of wrapper of remote managed objects and make them available for outer management applications. Idea of Mediator Connector is very simple. JoinAgent connects to Mediator, invokes *connect(Long seq, String mBeanServerId)* method on Mediator MBean and creates new MediatorConnector MBean Server. MediatorConnector is implemented as a wrapper for requests incoming from management application. Methods that return information about registered MBeans return in fact information about MBeans registered in remote MBean Server placed in secure region. For administration purposes they return additionally information about the two MBeans really registered in MediatorConnector: HTTPConnectorServer and DiscoveryResponder. Parameters of all management methods invoked on virtual MBeans registered in MediatorConnector are forwarded to proper methods of remote MBeans and results are sent back to client. This mechanism is fully transparent for connected applications, so it doesn't require any modification by application developer to adapt it to GDS.

Since there is no multicast connection from management application to Mediator Agent, so management application has to maintain the values of ProxyMBeanServers' IP addresses and ports as well. Each ProxyMBeanServer is operating using different port, so for two MBean Servers there are two different addresses of remote agents, for example: 149.156.97.150:1300 and 149.156.97.150:1301, because each proxy MBean Server is identified by different port of HTTP connector. Once the server is configured, there is possibility to

browse information about all MBeans registered on remote hosts as if they were registered in the proxy MBean Server placed on host where Mediator Agent is operating.

VI. APPLICATION CASE STUDY

Sophisticated architecture of GDS system involves many use cases that should be studied. They include:

1. registering new manageable resources in the mediator agent; conversely: removing manageable resources from mediator agent due to administrative action or network failure,
2. connecting management application to mediator agent; conversely: disconnecting management application from mediator agent; it doesn't matter what kind of disconnection occurred, because GDS doesn't rely on management applications,
3. performing global discovery by the management application, including neighbour agents discovery and *DiscoveryClient* installation in case of lack of such functionality,
4. obtaining data from manageable resources by management application.

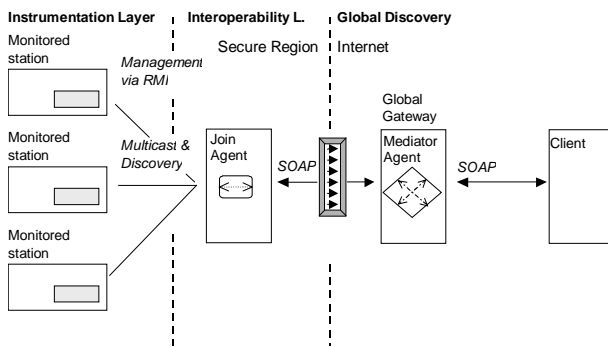


Fig. 7. Management services for grid: SOAP Gateway and Discovery.

Typical scenario for resources management using the GDS was shown in

Fig. 7:

1. client chooses the preferred application (GUI, CLI or web-based application),
2. connects to Mediator Agent and finds all available clusters using the GDS,
3. reads the list of all available Monitored Stations and chooses some of them for management,
4. performs some management operations like reading or writing attributes of managed resources or invokes methods for performing some measurements for QoS estimation.

GDS is being implemented in EU CrossGrid project, where it performs the role of global autoconfiguration mechanism for infrastructure monitoring service. Currently there is implemented the instrumentation and the interoperability layer (JIMS) and GDS is considered to be the mechanism for finding the sites (whole clusters) where Soap Gateways are running.

Presented concept of GDS is the base for dynamic resources discovery in wide area network, providing also required transparency for management and monitoring applications using JMX as a base technology. First experiments performed using GDS and its localization service proved that the concept can be successfully implemented using Java-based approach. There was possible to run MBean Servers in hidden areas and locate them using the GDS. Communication from the outside application was carried out by the JDMK HTTP connector. Since JDMK is the commercial implementation and cannot be freely used in Open Source project [10], in the future there is necessary to implement it using other packages like MX4J [15]. Future work can exploit the Soap Connector from MX4J substituting the one from JDMK. Due to component architecture of JMX this operation should not affect the whole system and not be even noticeable from the user's point of view. Using Soap Connector narrows usage of the Soap Gateway down to JMX clients only but provides more consistent view of monitored resources. Alternative approach can make advantage of Web Services and UDDI, what may be more flexible and universal. However, GDS seems to be the preferred solution, because UDDI is a very sophisticated and heavy mechanism, requiring database as a storage for the registry and interface-provider approach, what can be omitted here. The problem with keeping the registered sites up-to-date can be achieved in GDS using periodic updates sent by Join Agents to the Mediator Agents.

VII. CONCLUSION

JMX based Grid management services offer very attractive solution for the grid resource monitoring and configuration. The experiences gained by JIMS implementation show that GDS designed to solve the problems of communication through public networks, as well as give the users the potential of global discovery of hidden JMX agents is a very good alternative to SGs based on Web Services GDS provides an excellent level of transparency of managed resources what doesn't imply necessity of changing existing management applications. It is also JMX compliant, what means that it should work with other existing JMX implementations of MBeans and MBean Servers. GDS can be fully operational using the other, modern communication protocol, like JMX MP (JMX Messaging Protocol over TCP/IP), which can be easily configured to operate also on standard port for HTTP communication.

Presented architecture and its first implementation prove that dynamic resources discovery is not reserved only for local area networks, but can be fully operational in wide area networks, as it is in the Internet. GDS provides not only the global discovery utility functions, but makes discovered resources fully transparent for management applications.

Plans for the future GDS evolution should involve moving from JDMK to JMX reference implementation in order to

create GDS package not limited by proprietary license. In some cases traffic encryption should be also considered, what can be probably achieved using standard secure communicators found in Sun's implementation of JMX Remote API [2].

ACKNOWLEDGMENT

The authors would like to thank M.Sc. graduate students at DCS at AGH-UST L. Bizon, J. Midura, T. Sekman, M. Smet, and M. Rozenau for their contribution to the presented work.

REFERENCES

- [1] Sun Microsystems, Java(TM) Management Extensions (JMX TM) Reference Implementation v1.2, <http://java.sun.com/products/JavaManagement/download.html>
- [2] Sun Microsystems, Java(TM) Management Extensions (JMX TM) Remote API 1.0 Early Access 2, <http://developer.java.sun.com/developer/earlyAccess/jmx/>
- [3] Sun Microsystems, Java(TM) Dynamic Management Kit (JDMK TM), <http://java.sun.com/products/jdmk>
- [4] Jacek Midura, Kazimierz Balos, Krzysztof Zielinski "Global Discovery Service for JMX Architecture", ICCS 2004, LNCS 3038, pp. 114–118, 2004
- [5] Westhawk's Java SNMP v4.13, <http://snmp.westhawk.co.uk/>
- [6] James McGovern et al., A Practical Guide to Enterprise Architecture, Prentice Hall PTR, 2003
- [7] The Open Grid Services Infrastructure Working Group (OGSI-WG): OGSI Specification, <http://www.ggf.org/ogsi-wg>
- [8] The Open Grid Services Architecture Working Group (OGSA-WG): Globus Tutorial, www.globus.org/ogsa/
- [9] K. Balos, L. Bizon, M. Rozenau, K. Zielinski, Interoperability Architecture for Grid Networks Monitoring Systems, CGW '03, Workshop Proceedings, pp. 245-253, 2004
- [10] EU CrossGrid project, <http://www.eu-crossgrid.org>
- [11] K. Balos, D. Radziszowski, P. Rzepa, K. Zielinski, S. Zielinski, "Monitoring GRID Resources – JMX in Action", Department of Computer Science, AGH-UST, in press
- [12] L. Bizon, M. Rozenau: Web services application in computer system integration. M.A. thesis (in polish), Kraków, pp. 33-37, 2003
- [13] Object Management Group (OMG): Event Service Specification. Doc. 97-12-11, 1997
- [14] Object Management Group (OMG): Notification Service Specification. New Edition, v.1.0, 2000
- [15] MX4J - Open Source Java Management Extensions, <http://mx4j.sourceforge.net/docs/api/index.html>
- [16] AXIS, Apache Web Services Project, <http://ws.apache.org/axis/>
- [17] MDS, Monitoring and Discovery System, <http://www-unix.globus.org/toolkit/mds/>