# Monitoring GRID Resources – JMX in Action

K. Bałos, D. Radziszowski, P. Rzepa, K. Zieliński , S. Zieliński

Department of Computer Science

AGH-UST

**Abstract**

This paper summarizes research on monitoring GRID resources, which results in JIMS system implementation. It contains an overview of most important architectural and software concepts that make the constructed system very flexible and easy to use. The paper evaluates JMX and Web Service technologies as foundation for implementing monitoring systems. Particular attention was put on system adaptability, autoconfiguration, and interoperability.

## 1 Introduction

Monitoring of distributed computer system resources is an integral part of any management activity. The grid computing concept [] addresses issues related to accessibility and transparent sharing of distributed computational, storage and communicational resources among group of users, putting the management aspects at the first place in grid research. Recently, the grid research community has been inspired by a new approach based on SOA (Service Oriented Architecture) [16]. The most widely used way of implementing this type of systems is to use Web Services. Exposing system functionality, computation or visualization modules as services seems to be a very powerful and elegant approach. Because services are going to be the components that are shared between users, they should be monitored and managed similarly to hardware infrastructure. Such approach raises the requirement of uniform approach to monitoring and management of hardware and software resources. The dynamic character of the grid systems configuration requires searching for a system enabling instrumentation and activation of monitoring on demand at runtime.

Taking these general trends and requirements into account, three years ago the Distributed Systems Research Group – a group of scientists from the Department of Computer Science at AGH-UST started research on applying JMX [19, 20] and Jiro [21] technologies to monitoring grid infrastructure. That research activity has been carried out as a task of the CROSSGRID Project [22]. The research resulted in construction of a few prototype systems, which have been reported in M.Sc. thesis[18] and articles [8, 11]. The most advanced monitoring system constructed so far is JIMS, which is deployed in the CROSSGRID testbed system.

This paper summarizes the application of JMX in context of monitoring grid systems. It contains an overview of most important architectural and software concepts that makes the constructed systems very flexible and easy to use.

The structure of the paper is as follows. Section 2 consists of an overview of constructed systems' general architecture. It clarifies the multi-layer system design and specifies the technology used for implementation of each layer. Section 3 presents the deployment and configuration management aspects of monitoring system. Particular attention was put to

present dynamic discovery of monitored resources and the monitoring system auto-configuration features. Also a concept of on-demand instrumentation has been discussed. Next, section 4 covers the topics of collection and distribution of monitored data. Various techniques for accessing monitored resources have been briefly described and compared. In Section 5 the problem of storing monitoring data has been discussed in more detail. The section discusses also the problem of designing a universal database schema in a way flexible enough for adaptation to different requirements of monitoring applications. Section 6 contains description of requirements for and prototypes of user interfaces for monitoring systems. The paper is ended with conclusions.

## 2 Monitoring system architecture

The monitoring system presented in this paper has been constructed in accordance to contemporary trends in distributed systems architecture expressed by Service Oriented Architecture (SOA) [16]. 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 run, 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 for implementing a monitoring system are Web Services (WS) providing access to the system functionality built with Java Management Extension (JMX) support.

A system for monitoring grid resources 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 [19, 20] 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 [8]. 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 below 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 implemented with UDDI,
4. interface and presentation (GUI: web applications and standalone visualization tools, CLI applications, Java API).

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.

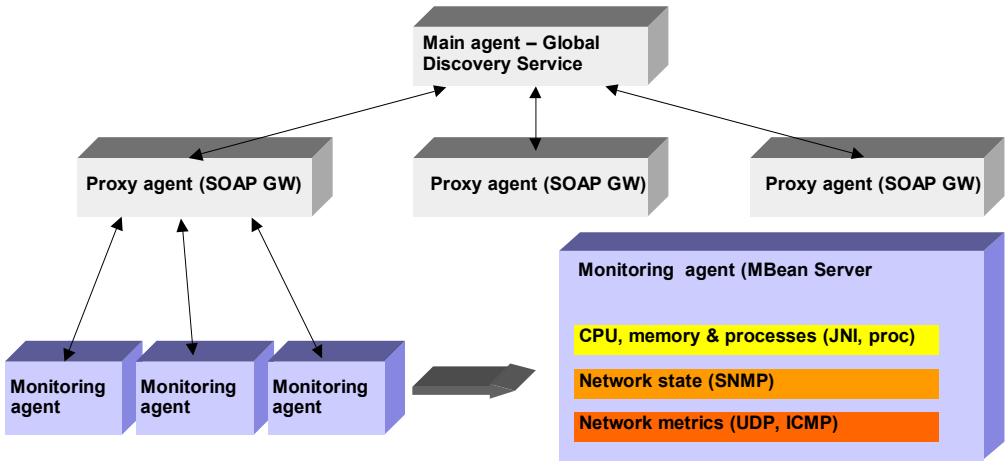


Fig. 1. The modules of JIMS

The instrumentation layer obtains information from monitored resources using the following mechanisms:

- Java Native Interface (JNI) for reading virtual file system (/proc),
- 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 [15] to be accessible from Java.
- 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 system and discovery of its entities. 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.

# 3 Monitored data acquisition

Classical network monitoring systems are usually centralized applications that monitor resources through direct management of associated agents acting as their representatives. An agent executes commands, gathers data and reads resources' state. It is typically a plain entity without any intelligence that could allow making decisions, placed close to the managed resource. An agent's role is limited to tasks delegated by a central management system.

JMX monitoring services are executed directly on the resource level. JMX agents are autonomous entities equipped with a proper level of intelligence for performing self-determined actions, thus relieving a management center of executing usual tasks such as querying resources' state. At the same time, it decreases network load and increases scalability of the monitoring system. JMX standardizes interfaces for monitoring resources that enable anyone to use arbitrary technology for building monitoring applications. Those applications can easily access monitored resources communicating through a JMX agent. Furthermore, the environment provides a distributed management model independent from any communication protocol. Given that, an application can rely on a particular API instead of properties of a specific protocol.

## 3.1 Event distribution model

JMX delivers the three basic monitoring modes: sampling (pull-mode), tracing (push-mode) and periodic notification on events. The user is able to choose the most suitable monitoring data delivery mode that depends on application. The architecture of JMX permits an MBean to broadcast event notifications both to other MBeans and to management applications acting as observers. In the simplest case, observers reside in the same Java Virtual Machine (JVM) as an MBean generating events. This situation enables an observer to register for notification either directly in an event-source MBean or through an MBean server. The way the registration is performed doesn't influence a path that an event traverses from event broadcaster to observers - they always go directly to the listener. In more complicated scenario, observers do not share a JVM with the event source and possibly operate on a different host.

Fortunately, JMX architecture hides the fact of physical separation through connector mechanism and in this way provides observers with transparent access to event sources. Use of connectors reduces collaboration to transferring registration requests from remote observers to an MBean representing particular resource and transferring notifications in the reverse direction. This notification model is based on a Java event model constrained to a single virtual machine. Such a construction introduces a need for local proxy mechanism - a connector server has to create local observers that store received notifications in a buffer. The connector server is expected to deliver the messages to remote observers at a later time. On the other hand, the proxy mechanism has some advantages: it permits to introduce a variety of event delivery policies and enables to choose appropriate policy according to the adopted quality of service.

In the JMX's notification model, event reports can be emitted by MBean instances and by the MBean Server, generally on specific changes of the MBean's attributes which are the fields or properties of the MBean's management interface. The mechanism for detecting changes in attributes and triggering the notification of events is not a part of JMX specification, at least in its current release. The attribute change notification behavior is therefore generally dependent upon the implementation of each MBean. The monitor MBeans are exceptional – they are in fact predefined sensors performing periodic sampling of an attribute of the MBean they

observe. The three types of monitor MBeans that are provided in every JMX implementation are *CounterMonitor, GaugeMonitor* and *StringMonitor*. If switched on, each of them automatically sends a relevant notification when a specific set of conditions concerning the value of the observed attribute is satisfied.

A JMX-enabled client application in may register as a *notification listener* with a *notification broadcaster* MBean and receive notifications on all events that may occur in the broadcaster, i.e. the listener's *handleNotification()* method will be invoked when any notification is issued by the MBean (an explicit implementation of the Observer design pattern [7]).

### 3.2 Multiprotocol access interface

The JMX architecture solves the problem of communication with monitoring systems by leveraging a variety of communication protocols. Monitoring applications developed in Java perform operations on remote objects and receive notifications from these objects through local representatives called proxies. The communication process between a proxy and its relevant remote object is hidden from clients. It is carried out by dedicated system components called *connector client* and *connector server*. These components enable mutual communication over diverse protocols such as RMI, HTTP/TCP or HTTP/SSL.

Other monitoring systems that use different protocols, such as HTML or SNMP, can connect to JMX agent through a specific *protocol adapter*. The HTML adapter, for example, renders MBean interfaces as web pages. SNMP adapter exposes SNMP MIBs representing MBeans that respond to SNMP commands.

The JMX architecture allows enclosing vendor-specific connectors and adapters that use arbitrary communication protocols. This proves that JMX is an open solution that makes collaboration with any existing monitoring or managing system possible.

### 3.3 Additional useful JMX services

JMX architecture introduces a powerful concept of *service agent* that facilitates monitoring and managing creation of application. Basic functionality of service agents can be described as follows:

- Querying and filtering --- provides clients with possibility of searching for MBeans. The search criteria include MBean full or partial name as well as expressions based on current MBean attribute values. The query results in a list of managed resources that can be subsequently used to invoke the elements' operations.
- Dynamic loading --- the service supports uploading a Java class from any network location and using it to construct an MBean object that can be later registered in an MBean server. This feature establishes a mechanism for enhancing existing agent functionality and introducing new resources to continuously operating environment.
- Monitoring service --- supplies a mechanism for polling MBean attribute values. It is possible to observe numerical attributes, floating point attributes that fit to a specified range and character string attributes in the terms of pattern matching.
- Timer service --- brings a mechanism for triggering notifications to registered listeners at a specified time enabling them to run particular action at that moment. The service makes possible to define single or periodic notifications and manages a list of dated notifications that determine the launch of actual action sequence.

The aforementioned issues cause the JMX architecture to be considered a leading technology that integrates management and monitoring in the data collection layer.

# 4 System deployment and configuration

Deployment and configuration of JIMS system is based on standard techniques taken from reference implementation of JMX built by SUN Microsystems (dynamic on-demand instrumentation layer and M-Let service), and some mechanisms, like discovery, and dynamic auto-configuration, are being developed by DSRG group itself.

## 4.1 System installation and startup

JIMS system startup process relies on mechanisms for loading software on demand to dedicated system modules. JMX technology supports such functionality with its M-Let (dynaMic appLet) Service. The service provides online loading and installation of java classes. Usage of the service in JIMS is depicted in Fig. 2.

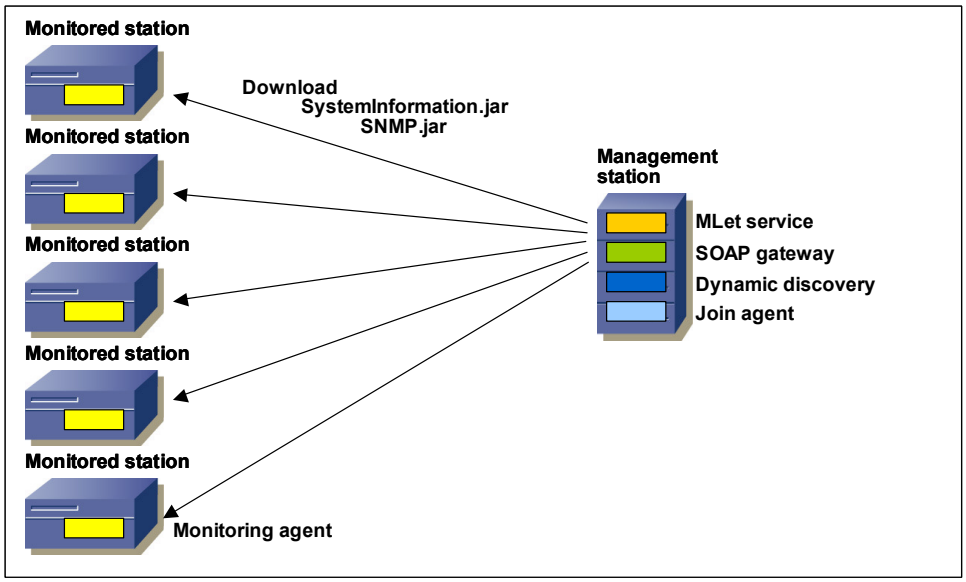


Fig. 2. JIMS components download and installation

Using the service, monitoring agents are automatically installed and run in a cluster. Management station starts all pre-installed monitoring agents, which then 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 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.

## 4.2 Interoperability issue

The SOAP Gateway (SG) concept is based on general approach described in OGSI (Open Grid Services Infrastructure) specification [12], where grid service is exposed as WS defined using WSDL (Web Service Definition Language), conforming to a set of conventions (interfaces and behaviours) that specify how clients and services interact. The SG concept is also based on architectural approach taken from OGSA (Open Grid Services Architecture) [13].

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 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 clearly be 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 proposed interoperability layer supports:

a. automatic installation to facilitate management of numerous nodes in clusters,
b. 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,
c. self-adaptation mechanism (dynamic discovery and heart beat) from the user,
d. exposing one point of communication through one - due to firewalling - well defined application address and port, with WSDL describing its functionality,
e. forwarding requests from WS clients to specified monitored stations [11].

### 4.3 Automatic and Dynamic Configuration

SG autoconfiguration is based on two mechanisms: 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 [14]. 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.

# 5 Monitored data warehouse

The key point of the data warehouse module in an open monitoring system is to create a universal database for storing data obtained from monitored resources. Such a database should support heterogeneity of resources as well as dynamic attributes setup, and provide uniform access interface for all kinds of monitoring data. There is lot of important factors that have to be taken into account while creating such model, the most crucial ones follow:

- Dynamic attributes definition – the list of attributes describing a monitored resource has to be easily extendable.
- Fine grained data support – ability to store each monitored attribute data item separately, data that are more interesting or that change more frequently may be logged more often independently of other data from the resource.
- Support for heterogeneous resources – possibility for storing data from complete different areas of interest e.g. host, network and storage.
- Uniform access - one common interface for access to heterogeneous dynamic data stored in a system has to be created.

The whole model built to meet these general requirements is quite complex, so we only describe the major issues of the proposed solution.

## 5.1 Data model

The types of data stored in the system may be divided into two groups; first - metadata - describes an environment (resources and attributes), while the second is formed by monitoring data values themselves.

Metadata specify:

- site — has the same meaning as in grid terminology and is used for narrowing resources in different geographical locations
- kind — is the group of homogenous resources having some common attributes (e.g. computer, network device)
- resource — is the group of simple (String) data describing a source of monitoring data; it has a name, description and unique identification string
- attribute — represents an attribute exposed by the resource for monitoring; there are two types of attributes: simple for simple attributes and compound for attributes that have sub-attributes (e.g. ‘location’ attribute may have ‘street’ and ‘city’ sub-attributes). Both of them may be vector or scalar – depending of the values they represent.

The values of monitored attributes may have different types. The system supports the following primitive types:

- String – for textual data
- Long – for all Integer values
- Float – for floating point values

These three types are sufficient to represent a variety of commonly used simple data and are successfully used in e.g. SNMP[1]; more complex data like structures and vectors are specified by appropriate definition of metadata layer.

### 5.2 Data warehouse access

The system will expose three independent interfaces: data store interface, enabling data collection layer to store the data; data access interface that makes querying and data access possible, and the administration interface designed for administrative purposes. Each of the system interfaces will be accessible remotely and, if desired, it will be possible to make use of the business delegates pattern [3].

#### Data store interface

The key role of this interface is to accept data coming form the data collection layer. This process will consist of two basic parts: metadata layer configuration and storage of attributes' values. Because the values of the monitoring attributes need semantic context, appropriate configuration of metadata layer is a key point for further data processing. The configuration process starts with registration of a new resource, and then all monitored attributes of this resource are registered. Note that an attribute set may be extended in any time if the resource's agent decides to monitor additional attributes. After the initial configuration phase the values of resource's attributes may be stored. The interface will support both individual values and packets containing sets of values. Because the lower layer (built upon JMX) supports all basic monitoring models, only the 'push' model for this interface is required.

#### Data access interface

Access to the monitoring data is based on a dialog between a client and the system. A client will specify in more and more details which data it is interested in. First of all it will be required to select sites, kinds of resources and resources by choosing form the list of all monitored resources or by specifying names of the resources. As a response, it will receive the list of each resource's monitored attributes. It will have access to all the additional data that are stored together with attributes e.g. name, description, and unit. Next, it will have to point out, which attributes it is interested in (it may select simple attributes, compound attributes with scalar or vector values) and set time clauses (e.g. from-to, all up to now, all older than). Finally, it will receive the desired data according to the selection made. This entire dialog will be kept opaque by appropriate classes according to the common OOP directives.

Similarly to the data store interface, data access interface will expose a programmer interface according to the business delegate pattern and additionally will be accessible from a web browser.

#### Administrative interface

As the volume of collected data will grow very quickly, a mechanism for maintenance and especially for removing unneeded data is required. The administrative interface will also allow for setting or changing descriptions and units in metadata. It will be based on a web interface and will be accessible form a web browser.

### 5.3 Implementation and efficiency issues

Because the system is designed as a multi-tier, distributed application, the proposed platform for the implementation of the system is J2EE. This environment offers EJB as standard

mechanism for implementation of database access. This solution could be further enhanced with load balancing, fail-over and security mechanisms. Another key implementation issue is the efficiency of the proposed solution. The usage of scalable application servers is very helpful but adequate design of the system and its configuration is even more important. The current project stage assumes usage of J2EE persistence concept (CMP, DAO or JDO) for the meta-data layer mostly because the meta-data management is quite complicated. Its object oriented structure suits perfectly to this concept and efficiency is not crucial. The already performed tests showed that this approach is not useful for collecting monitored attributes' values. These data are supposed to be processed much more efficiently by direct JDBC calls that ensure minimal time overhead.

# 6 User interface construction

Grid environment is set up by substantial number of objects that can be and actually are monitored. The monitored objects usually have numerous attributes that define their state. Both issues imply existence of almost unlimited quantity of information that can be presented to an end user in order to let him monitor grid activity. The amount of data provided by JIMS and diversity of data sources make the problem of providing user-friendly access to grid state challenging. In order to be useful, the system modules dedicated for collaboration with end user, should meet the following requirements:

- allow reviewing all the monitored objects and their attributes,
- support efficient, scalable selection of specific grid components,
- allow examination of values of selected attributes,
- allow concurrent examination of values of several attributes,
- allow viewing values in various formats — for example text-based, graphic charts, XML,
- provide means for inspecting the history of monitored object attributes,
- have flexible graphic interfaces adjusting to various end-devices,
- provide authorized access to monitored values per user or role basis.

The implemented system introduces several client applications for different levels of grid state examination:

- a text-based and standalone java application that access system through web services,
- an HTTP access based on built-in html adapter,
- a standalone java application acting as an SNMP client,
- a web application accessing monitored data warehouse.

## 6.1 Web services based clients

The system implements two types of clients based on web services access that exhibit grid state to the user: a text-based application and a standalone java application. The former is only an example of using web services interface in text-based clients and expose latency and throughput among monitored objects in a cluster (see Fig.3). The latter is a powerful application that enables users to select appropriate cluster, monitored object, the object's

attribute and to view its value (see Fig.4). A user can launch multiple applications in parallel in order to monitor many attribute values at the same time. The application is equipped with advanced features allowing for example to monitor a selected attribute constantly. That characteristic permits to exhibit the sequence of attribute's values in the near history as a graphic chart.

```
[dsrg@maksi bin]$ ./cg-jims-client 149.156.9.15
JIMS client v 1.4
Timestamp for [Ut]: ts=1081997570476
149.156.9.44: Ut=11163238  ICMP lat: 0.212 [ms],      Throughput: 2.8106666666666668E7 [bit/s]
149.156.9.23: Ut=11884675  ICMP lat: 0.125 [ms],      Throughput: 2.8106666666666668E7 [bit/s]
149.156.9.26: Ut=7141152   ICMP lat: 0.132 [ms],      Throughput: 2.409142857142857E7 [bit/s]
149.156.9.39: Ut=7621342   ICMP lat: 0.161 [ms],      Throughput: 2.8106666666666668E7 [bit/s]
149.156.9.20: Ut=171871991 ICMP lat: 0.129 [ms],      Throughput: 340686.8686868687 [bit/s]
149.156.9.16: Ut=10230438  ICMP lat: 0.126 [ms],      Throughput: 2.8106666666666668E7 [bit/s]
149.156.9.18: Ut=120823614 ICMP lat: 0.141 [ms],      Throughput: 2.8106666666666668E7 [bit/s]
149.156.9.24: Ut=8538910   ICMP lat: 0.13 [ms],       Throughput: 2.8106666666666668E7 [bit/s]
```

Fig. 3. Command line web services based client

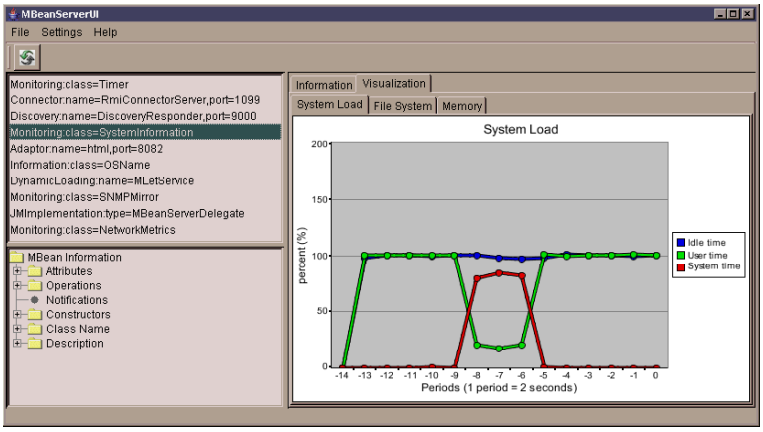


Fig. 4. Standalone Java client leveraging web services

## 6.2 HTTP client

The JIMS system offers also HTTP-based access to all monitored resources that are equipped with a JMX HTTP adapter (see Fig.5). It is a simple but powerful way of accessing monitored objects that gives a full access to the monitored object functionality. The user must know a URI of a monitored object as a prerequisite. After providing the URI he is able to view the list of monitored object's attributes, read the attribute value and, if it the monitored object allows that, to set the value. When an object expose some operations that can be called upon it, the HTTP interface enables users to invoke them. A user is able to supply the operation with suitable argument list, invoke it on an object and view the results. This client is able to present the last known value of an attribute. In order to see how the attribute value fluctuates one should make use of either the web services based standalone java application or data warehouse client described in Section 6.

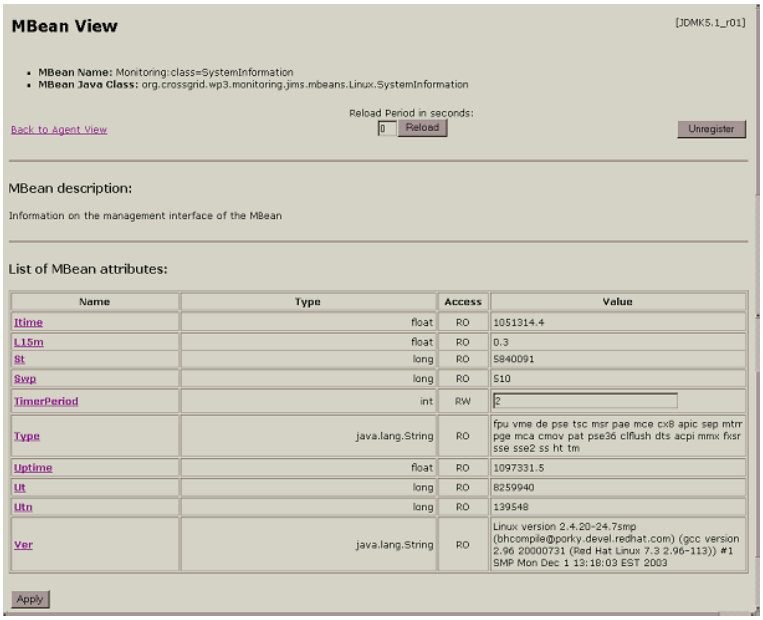


Fig. 5. HTTP-based access to MBean functionality

## 6.3 SNMP client

The system introduces a standalone java client application that utilizes SNMP to connect to monitored resources. The SNMP client enables user to access any monitored resource equipped with JMX SNMP adapter. The user must specify an IP address of a resource and then he gets access to the standardized management information base. It is possible to review all attributes of a monitored object and get a textual attribute value. When a managed object exposes an interface that enables to invoke some operation on this object, it is impossible to do so through SNMP client. This is not a limitation of the SNMP client, but SNMP protocol itself, which restricts the interaction with a managed object to getting or setting a value of its attributes.

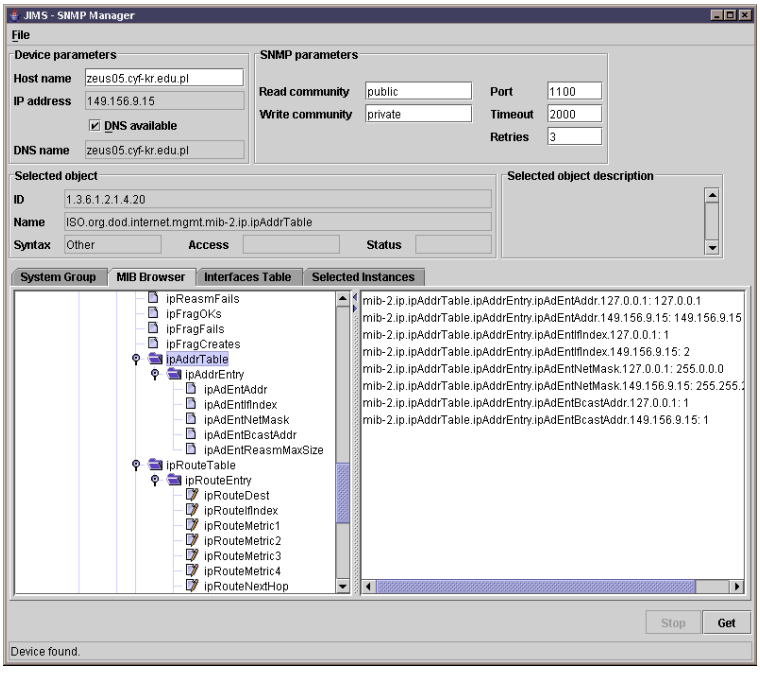


Fig. 6. SNMP based client application

### 6.4 Data warehouse client

The last client interacts not with the active grid itself, but with the grid image stored in the monitored data warehouse. This is not a weakness of a client or a solution, however. The warehouse keeps the up to date view of a grid state as well as a history of the grid activity and in this way exposes grid resources with the broader characteristic than just the most recent state, as it happens to the other clients. With the data warehouse the client is a user able to select a proper attribute or resource to view its state it two different ways. It can walk through the hierarchy of grid components selecting a site, then a kind, next a resource, after that an attribute and recursively an attribute's sub-attribute. The second method is more scalable: a user is capable of selecting proper components relying upon a component state, fixed in a time constraints. The client application uses a query language specified by a monitored data warehouse interface. As an example of such query consider "Select all computers that CPU load has been greater that 80% during last hour". After selecting proper components the user is able to specify time constraints in which the client wants to review components state. The subsequent action is to determine the way the results should be presented. The user can choose text-based documents, html pages, XML documents and finally graphic charts to review the values of selected attributes.

Another interesting feature of that client is built-in internationalization support. At this moment it is possible to customize user interface to work in English or Polish. An example of such client is presented in Fig.7.

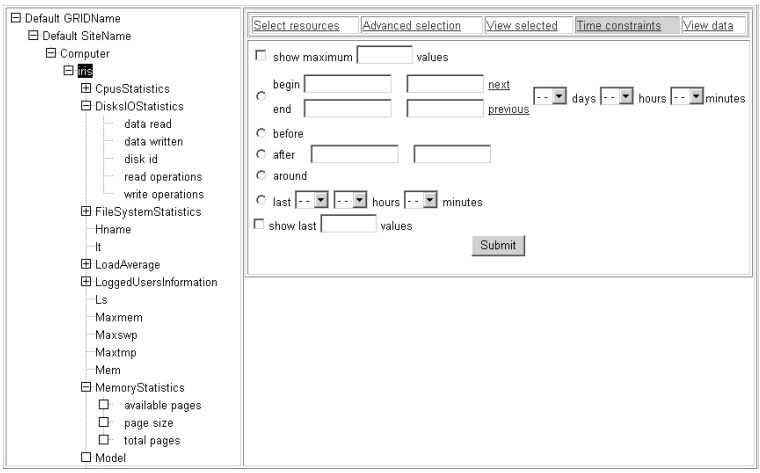


Fig. 7. Data warehouse client

## 7 Conclusions

The presented paper summarizes research on monitoring systems for grid infrastructure. Such class of systems is characterized by multi-layer structure and wide spectrum of technologies that should be very carefully selected for their implementation. The adaptability and configurability requirements make the JMX technology the most suitable for the instrumentation and agent layers implementations. The interoperability issues put Web Services on the first place among technologies offering support for SOA implementation. This technology is going also to be most suitable when integration problems have to be resolved.

Considering monitoring data storage and their accessibility, EJB technology seems to be a very natural choice. Efficient usage of this technology requires a lot attention that must be put especially when persistency mechanisms are implemented. For monitoring data storage, direct

JDBC-based access to database seems to be most appropriate. The different technologies employed result in different client applications offering access to collected by JIMS monitoring data. An advantage of the proposed solution is that it is built over open standards that makes JIMS easier to be extended and deployed in any environment.

**Acknowledgements**

We would like to thank our students, L. Bizoń, B. Ławniczek, G. Majka, J. Midura, M. Rozenau, T. Sekman and M. Smęt, for their great contribution to the development of JIMS.

# 8 Bibliography

1. William Stallings "SNMP, SNMP v2 and CMIP – The Practical Guide to Network-Management Standards" Addison-Wesley 1993
2. Ed Roman, Scott W. Ambler "Mastering Enterprise JavaBeans" Jhon Wiley&Sons Inc. 2002
3. Floyd Marinescu "EJB Design Patterns" Wiley&Sons Inc. 2002
4. Mike Jasnowski "JMX Programming" Wiley&Sons Inc. 2002
5. Bruce Eckel, Thinking in Java, edycja polska, Wydawnictwo Helion, 2001
6. Global Grid Forum, http://www.gridforum.org
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns, Addison-Wesley (1994) 273-283
8. Jacek Midura, Kazimierz Bałos, Krzysztof Zieliński "Global Discovery Service for JMX Architecture" ICCS 2004 Krakow
9. Aleksander Laurentowski, Krzysztof Zieliński „Integracja systemów B2B: JMX", TeleNetForum 2002
10. K. Bałos, S. Zieliński, JIMS – the JMX Infrastructure Monitoring System, http://www.eu-crossgrid.org/Seminars-INP/JIMS_monitoring_system.pdf
11. K. Bałos, L. Bizoń, M. Rozenau, K. Zieliński, Interoperability Architecture for Grid Networks Monitoring Systems, CGW '03
12. The Open Grid Services Infrastructure Working Group (OGSI-WG): OGSI Specification, http://www.ggf.org/ogsi-wg
13. The Open Grid Services Architecture Working Group (OGSA-WG): Globus Tutorial, www.globus.org/ogsa/
14. Sun Microsystems, JDMK, http://java.sun.com/products/jdmk
15. Westhawk's Java SNMP v4.13, http://snmp.westhawk.co.uk/
16. James McGovern et al., A Practical Guide to Enterprise Architecture, Prentice Hall PTR, 2003
17. http://www.serviceoriented.org/service_composition.html
18. T. Sekman, M. Smęt, Dynamicznie konfigurowalny system monitorowania zasobów gridowych, M.Sc. Thesis, Kraków 2004

19. Sun Microsystems, Java(TM) Management Extensions (JMX TM) Reference Implementation v1.2, http://java.sun.com/products/JavaManagement/download.html
20. Sun Microsystems, Java(TM) Management Extensions (JMX TM) Remote API 1.0 Early Access 2, http://developer.java.sun.com/developer/earlyAccess/jmx/
21. Sun Microsystems, JIRO™ Technology, www.jiro.com
22. EU CrossGrid Project, www.crossgrid.org