TASK QUARTERLY 8 No 4, 487–501

MONITORING GRID RESOURCES: JMX IN ACTION

KAZIMIERZ BALOS, DOMINIK RADZISZOWSKI,
PAWEŁ RZEPA, KRZYSZTOF ZIELIŃSKI
AND SŁAWOMIR ZIELIŃSKI

Department of Computer Science, AGH-University of Science and Technology,
Al. Mickiewicza 30, 30-059 Cracow, Poland
{kbalos, radzisz, rzepa, kz, slawek}@ics.agh.edu.pl

(Received 6 June 2004)

Abstract: This paper summarizes research on monitoring GRID resources, which resulted in the implementation of the JIMS system. It contains an overview of the most important architectural and software concepts that make the constructed system flexible and user-friendly. The paper evaluates JMX and Web Service technologies as foundations for implementing monitoring systems. Particular attention has been paid to system adaptability, autoconfiguration and interoperability.

Keywords: JMX, Grid, Monitoring, Distributed System, SOA, Web Services

1. Introduction

Monitoring distributed computer system resources is an integral part of any management activity. The grid computing concept [1] addresses issues related to accessibility and transparent sharing of distributed computational, storage and communication resources among groups of users, putting the management aspects at the forefront of grid research. Recently, the grid research community has been inspired by a new approach based on SOA (Service Oriented Architecture) [2].The most popular way of implementing this type of systems is to use Web Services. Exposing the system's functionality, computation or visualization modules as services seems to be a powerful and elegant approach. As services are going to be the components shared among users, they should be monitored and managed similarly to the hardware infrastructure. This approach requires a uniform approach to the monitoring and management of hardware and software resources. The dynamic character of 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 [3–5] and Jiro [6] technologies to monitor grid infrastructures. The research was carried out as a task of

the CROSSGRID Project [7]. It resulted in the construction of a few prototype systems, which have been reported in an MSc. thesis [8] and articles [9, 10]. 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 the context of monitoring grid systems. It contains an overview of the most important architectural and software concepts that make the constructed systems flexible and user-friendly.

The structure of the paper is as follows. Section 2 consists of an overview of the constructed systems' general architecture. It clarifies the multi-layer system design and specifies the technology used to implement each layer. Section 3 presents the deployment and configuration management aspects of the monitoring system. In particular, dynamic discovery of the monitored resources and the monitoring system's auto-configuration features are presented. The concept of on-demand instrumentation is discussed. Section 4 covers collection and distribution of the monitored data. Various techniques for accessing the monitored resources are briefly described and compared. In Section 5 the problem of storing the monitoring data is discussed in more detail. The section also discusses the problem of designing a universal database schema in a way flexible enough for adaptation to different requirements of monitoring applications. Section 6 contains a description of requirements for and prototypes of user interfaces for monitoring systems. The paper ends with conclusions.

2. The monitoring system's architecture

The monitoring system presented in this paper has been constructed in accordance with contemporary trends in distributed systems' architecture expressed by Service Oriented Architecture (SOA) [2]. Systems of this 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 great importance, the components of a system can migrate or be duplicated. Moreover, the environment in which a distributed system is run may have a fluctuating nature. Therefore, 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 these are effective coupling of internal system services, describing the information flow, implementing communication channels, maintaining security, *etc.* The services an 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 meet these requirements. A good example of such platform for implementing a monitoring system are Web Services (WS) providing access to the system's functionality built with a Java Management Extension (JMX) support.

A system for monitoring grid resources should have a scalable and flexible architecture, easy to develop and maintain. JIMS, the JMX-based Infrastructure Monitoring System, developed by DSRG at AGH-UST, meets the requirements for monitoring systems operating in distributed environments and supports resource abstraction, dynamic system configuration and interoperability. Based on the 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 [3, 4] addresses the first two levels and provides only a brief overview of the latter. JMX provides interfaces and services adequate to the requirements of monitoring and management systems [9]. This functionality involves abstracting resources by using components called MBeans (Managed Beans) and remote instrumented resource accessibility through JMX connectors. The functionality developed in the JIMS project employs dynamic discovery of the monitored resources. In order to achieve more flexibility and interoperability, a Web Service provides monitored objects' proxies that are used to implement lightweight clients. The architecture of JIMS is shown in Figure 1. The monitoring system is decomposed into the following layers:

1. the resource instrumentation layer (JMX MBean Servers, SNMP, RMI);
2. the interoperability layer (SOAP Gateways, Web Services);
3. an integration layer implemented with UDDI;
4. interface and presentation (GUI: web applications and stand-alone visualization tools, CLI applications, Java API).

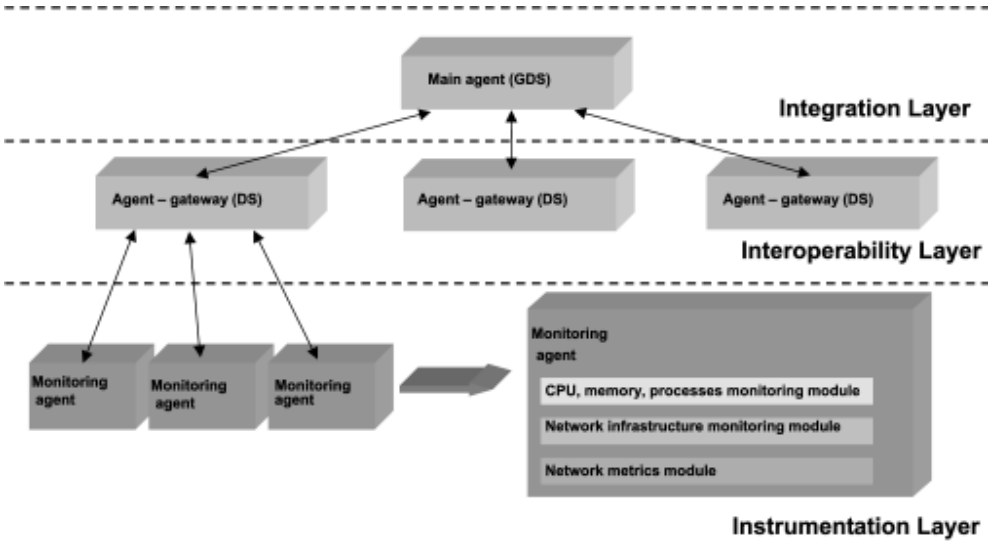

Figure 1. The modules of JIMS

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

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

- a Java Native Interface (JNI) for reading virtual file system (/proc);

- an SNMP agent running locally on the monitored system, usually at each Worker Node of a site (these agents communicate with the monitoring system through an SNMP protocol using SNMP API [11] to be accessible from Java);
- a Network Metrics module, measuring network conditions 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, building 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 the monitored parameters in a uniform way. It is important that all the 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 also play 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 the currently running SOAP Gateways. The proposed hierarchical architecture is suitable for large distributed systems and offers the 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 the 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 level of intelligence proper for performing self-determined actions, thus relieving the management center of executing its usual tasks such as querying the 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 an arbitrary technology to build monitoring applications. These applications can easily access the monitored resources communicating through a JMX agent. Furthermore, the environment provides a distributed management model independent of any communication protocol. Given that, an application can rely on a particular API instead of the properties of a specific protocol [12].

3.1. The event distribution model

JMX delivers the three basic monitoring modes: sampling (the pull mode), tracing (the push mode) and periodic notification on events. The user is able to choose the most suitable mode of delivery of monitoring data that depends on the 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 the MBean generating events. This situation enables an observer to register for notification either directly in the event-source MBean or through an MBean server. The way in which registration is performed does not influence the path that an event traverses from the event broadcaster to observers: they always go directly to the listener. In a 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 a connector mechanism and thus provides observers with transparent access to event sources. The use of connectors reduces collaboration to transferring registration requests from remote observers to the MBean representing the 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 construction requires a local proxy mechanism: a connector server has to create local observers that store the received notifications in a buffer. The connector server is expected to deliver the messages to remote observers at a later time. At the same time, the proxy mechanism has some advantages: it permits the introduction of a variety of event delivery policies and a choice of the appropriate policy according to the adopted quality of service.

In 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 notification of events is not a part of the 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 may register as a *notification listener* with a *notification broadcaster* MBean and receive notifications of 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 [13]).

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 a *connector client* and a *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 a JMX agent through a specific *protocol adapter*. The HTML adapter, for example, renders MBean interfaces as web pages. The 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 possible collaboration with any existing monitoring or managing system.

3.3. Additional JMX services

The JMX architecture introduces the powerful concept of a *service agent* that facilitates process of management and creation of monitoring application. The basic functionalities of service agents can be described as follows:

- Querying and filtering – provides clients with the possibility of searching for MBeans. The search criteria include full or partial MBean name as well as expressions based on the current MBean attribute values. A 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 agent functionality and introducing new resources to a continuously operating environment.
- The monitoring service – supplies a mechanism for polling MBean attribute values. It is possible to observe numerical attributes, floating point attributes that fit 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 a particular action at that moment. The service makes it possible to define single or periodic notifications and manages a list of dated notifications that determine the launch of an actual action sequence.

The aforementioned aspects make the JMX architecture a leading technology integrating management and monitoring in the data collection layer.

4. The system's deployment and configuration

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

4.1. The system's installation and start-up

The JIMS system's start-up process relies on mechanisms for loading software on demand to dedicated system modules. The JMX technology supports such functionality with its M-Let (dynaMic appLet) Service. The service provides online loading and installation of Java classes. The usage of the service in JIMS is depicted in Figure 2.

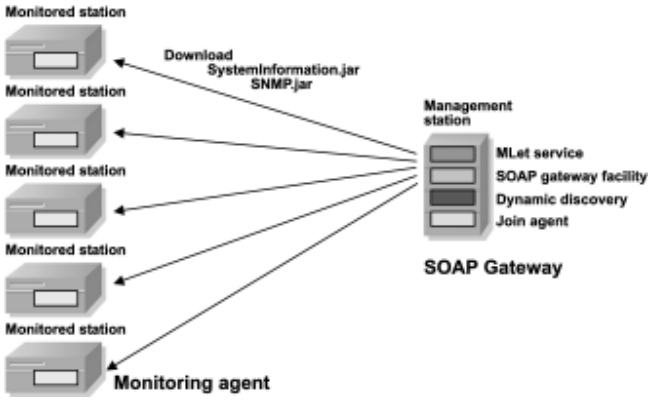

Figure 2. JIMS components' download and installation [14]

Using the service, monitoring agents are automatically installed and run in a cluster. A management station starts all pre-installed monitoring agents, which then download and install monitoring modules, being 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 later time. It is possible to upgrade or install newly developed modules, as well as remove existing ones without restarting the whole monitoring system.

4.2. The interoperability issue

The SOAP Gateway (SG) concept is based on the general approach described in the OGSI (Open Grid Services Infrastructure) specification [15], where grid service is exposed as a WS defined using a 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 the architectural approach taken from OGSA (Open Grid Services Architecture) [16].

Just like with other grid services, the layer of interoperability for an infrastructure monitoring system should support transient service instances, created and destroyed dynamically, and offer a unified way to access all the monitored resources. SG allows hiding the complexity of managing the monitored stations and exposes interfaces consistent with other grid services. As clusters in grids consist of many monitored computing elements, the interoperability layer should also perform the role of a router, forwarding requests from an outer point of communication to a specified node. To achieve this, it should store addresses of all the available monitored stations. In large installations with hundreds of nodes, administratively assigning the 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 the purpose of fail-over.

SG resides in a multiprotocol environment, *i.e.* with SOAP on the external system side and Java RMI on the side of monitored stations. Therefore, it should

perform the role of translator, connecting itself to the monitored nodes through RMI connectors and to client applications through WS.

In summary, the proposed interoperability layer supports:

- automatic installation to facilitate management of numerous nodes in clusters;
- automatic configuration with dynamic discovery mechanisms for finding monitored stations that are currently available and a heartbeat mechanism for removing stations that do not operate properly and are not responding for a certain period of time;
- a self-adaptation mechanism (dynamic discovery and heartbeat) from the user;
- exposing one point of communication through one – due to firewalling – well defined application address and port, with WSDL describing its functionality;
- forwarding requests from WS clients to specified monitored stations [10].

4.3. Automatic and dynamic configuration

SG autoconfiguration is based on two mechanisms: dynamic discovery and heartbeat. The former uses Discovery Monitors on the side of SG and Discovery Responders in the monitored stations in order to provide Active Discovery in much the same way as in JDMK [17]. SG periodically sends multicast requests to all the monitored stations, and they respond with their RMI addresses of JMX connectors. The latter mechanism, heartbeat, is a process complementary to the 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 the first access failure.

As can be expected, SG requires very little logic on the client side of the application, as the 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 the monitored resources. Each change of the monitoring station (vanishing or changing the JMX RMI address due to an MBean server's restart or physical crash) is handled by a SOAP Gateway, so that the client application each time obtains a 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 the monitored resources. Such a database should support heterogeneity of resources as well as dynamic setup of attributes, and provide a uniform access interface for all kinds of monitoring data. There are many important factors that have to be taken into account while creating such model; the most crucial are as follows:

- dynamic attribute definition: the list of attributes describing a monitored resource has to be easily extendable;

- fine-grained data support: the ability to store each monitored attribute data item separately; data that are more interesting or change more frequently may be logged more often independently of other data from the resource;
- support for heterogeneous resources: the possibility to store data from completely different areas of interest, *e.g.* host, network and storage;
- uniform access: a common interface has to be created to access heterogeneous dynamic data stored in a system.

As the whole model built to meet these general requirements is quite complex, we shall restrict ourselves to describing the most important aspects of the proposed solution.

5.1. The data model

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

Metadata specify:

- site – which has the same meaning as in the grid terminology and is used for narrowing resources in different geographical locations;
- kind – is a group of homogenous resources of common attributes (*e.g.* computer, network device);
- resource – is a group of simple (String) data describing a source of monitoring data (having a name, description and a unique identification string);
- attribute – which 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.* a location attribute may have street and city sub-attributes. Both of them may be vector or scalar, depending on 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 SNMP [18]. More complex data, like structures and vectors, are specified by an appropriate definition of the metadata layer.

5.2. Data warehouse access

The system will expose three independent interfaces: a data store interface, enabling the data collection layer to store data; a data access interface, that makes querying and data access possible, and an administration interface, designed for administrative purposes. Each of the system's interfaces will be accessible remotely and, if desired, it will be possible to make use of the business delegates pattern [19, 20].

5.2.1. The data store interface

The key role of this interface is to accept data coming from 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 a semantic context, appropriate configuration of the metadata layer is a key point for further data processing. The configuration process starts with registration of a new resource, following which all the monitored attributes of this resource are registered. Please note that an attribute set may be extended at any time if the resources agent decides to monitor additional attributes. After the initial configuration phase, the values of resource 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 is required for this interface.

5.2.2. The 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 detail which data it is interested in. First of all, it will be required to select sites, kinds of resources and resources by choosing from the list of all monitored resources or by specifying names of resources. As a response, it will receive a list of each resource's monitored attributes. It will have access to all the additional data that are stored together with attributes, incl. name, description and unit. Next, it will have to point out which attributes it is interested in (simple attributes or 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. The entire dialog will be kept opaque by appropriate classes according to the common OOP directives.

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

5.2.3. The administrative interface

As the volume of collected data will grow very quickly, a mechanism is required for maintaining and especially removing useless data. 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 system's implementation is J2EE. This environment offers EJB as the standard mechanism for implementation of database access. This solution could be enhanced further 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 an adequate design of the system and its configuration is even more important. The current project stage assumes the usage of the J2EE persistence concept (CMP, DAO or JDO) for the meta-data layer, mainly because meta-data management is quite complicated. Its object-oriented structure suits this concept perfectly and efficiency is not crucial. The tests performed so far have shown that this approach is not useful for collecting the monitored attributes' values. These data are supposed to be processed much more efficiently by direct JDBC calls that ensure a minimal time overhead.

6. User interface construction

The grid environment is set up by a substantial number of objects that can be and actually are monitored. The monitored objects usually have numerous attributes that define their state. This implies the existence of an 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 the diversity of data sources make the problem of providing user-friendly access to grid state a challenging one. In order to be useful, the system modules dedicated to collaboration with the 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, *e.g.* text-based, graphic charts, XML;
- provide means for inspecting the history of the monitored object's attributes;
- have flexible graphic interfaces adjustable to various end-devices;
- provide authorized access to monitored values on the per user or role basis.

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

- a text-based, stand-alone Java application that accesses the system through web services;
- HTTP access based on a built-in html adapter;
- a stand-alone Java application acting as an SNMP client;
- a web application accessing the monitored data warehouse.

6.1. Web services-based clients

The system implements two types of clients based on web service access that exhibit grid state to the user: a text-based application and a stand-alone Java application. The former is merely an example of using a web service interface in text-based clients and it exposes latency and throughput among the monitored objects in a cluster (see Figure 3). The latter is a powerful application that enables users to select an appropriate cluster, monitored object, object's attribute and to view its value (see Figure 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 *e.g.* to constantly monitor a selected attribute. This permits to exhibit a sequence of attribute values in the near history as a graphic chart.

```
[kbalos@galaxy kbalos]$ cg-jims-cli
JIMS Interactive CLI v. 1.4.0, type 'help' for help
JIMS>configure
JIMS-SOAP Gateway HOST>ce010.fzk.de
JIMS-SOAP Gateway PORT>7702
JIMS>netstats 141.52.160.36
[00] 141.52.160.36:     ICMP:  0.063 [ms],      UDP:   0.15 [ms],       Throughput: 5.62E7 [bit/s]
[01] 141.52.160.33:     ICMP:  0.359 [ms],      UDP:    0.3 [ms],       Throughput: 2.81E7 [bit/s]
[02] 141.52.160.34:     ICMP:  0.493 [ms],      UDP:   0.35 [ms],       Throughput: 2.81E7 [bit/s]
[03] 141.52.160.35:     ICMP:  0.425 [ms],      UDP:   0.35 [ms],       Throughput: 2.81E7 [bit/s]
JIMS>
```

Figure 3. The command line of a web service-based client

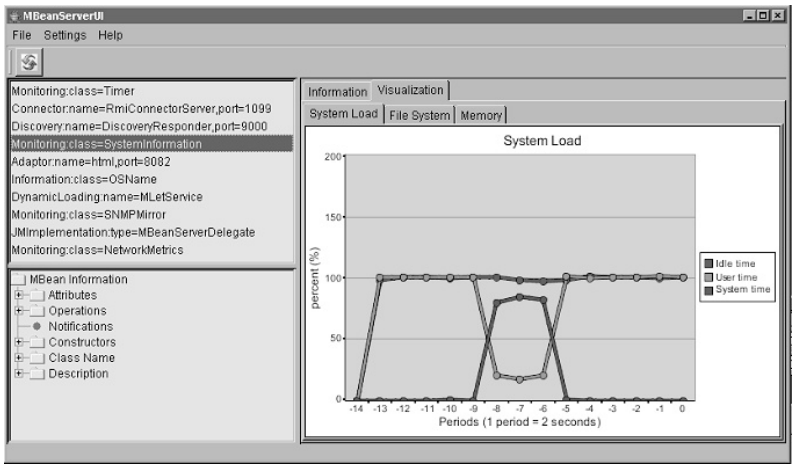

Figure 4. A stand-alone Java client using web services

6.2. HTTP clients

The JIMS system also offers HTTP-based access to all the monitored resources that are equipped with a JMX HTTP adapter (see Figure 5). It is a simple but powerful way of accessing monitored objects that gives full access to their functionality. The user must know the URI of a monitored object as a prerequisite. After providing the URI he is able to view a list of the monitored object's attributes, read the attribute value and, if the monitored object allows that, to set the value. When an object exposes operations that can be called upon it, the HTTP interface enables users to invoke them. A user is able to supply the operation with a 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 service-based stand-alone Java application or a data warehouse client described in this section.

6.3. SNMP clients

The system introduces a stand-alone Java client application that utilizes SNMP to connect to the monitored resources. An SNMP client (depicted in Figure 6) provides users with access to any monitored resource equipped with a JMX SNMP adapter. The user must specify an IP address of a resource to access 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 making it possible to invoke an operation on this object, it is impossible to do so through an SNMP client. This is not a limitation of the SNMP client, but of the SNMP protocol itself, which restricts the interaction with a managed object to getting or setting values of its attributes.

6.4. Data warehouse clients

The last client category 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 an up-to-date view of a grid state as well as a history of the grid's activity and thus exposes grid resources with a broader

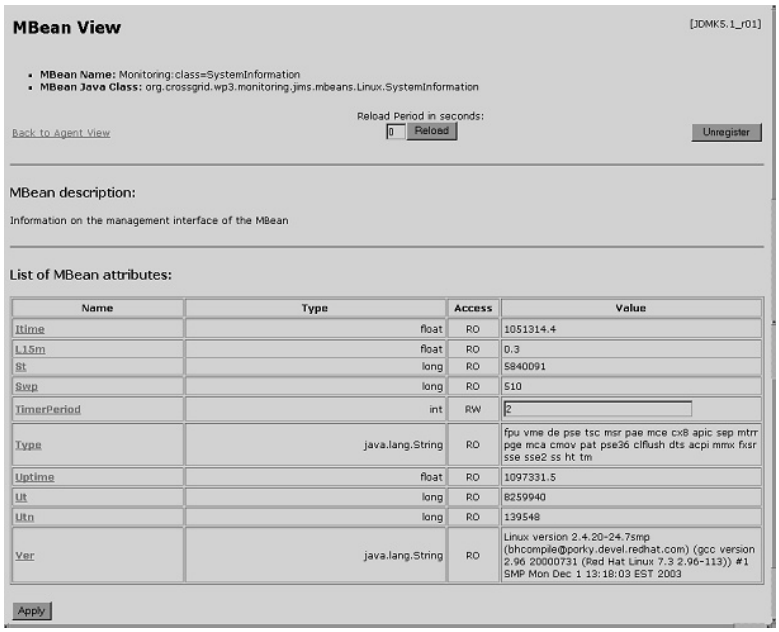

Figure 5. HTTP-based access to MBean functionality

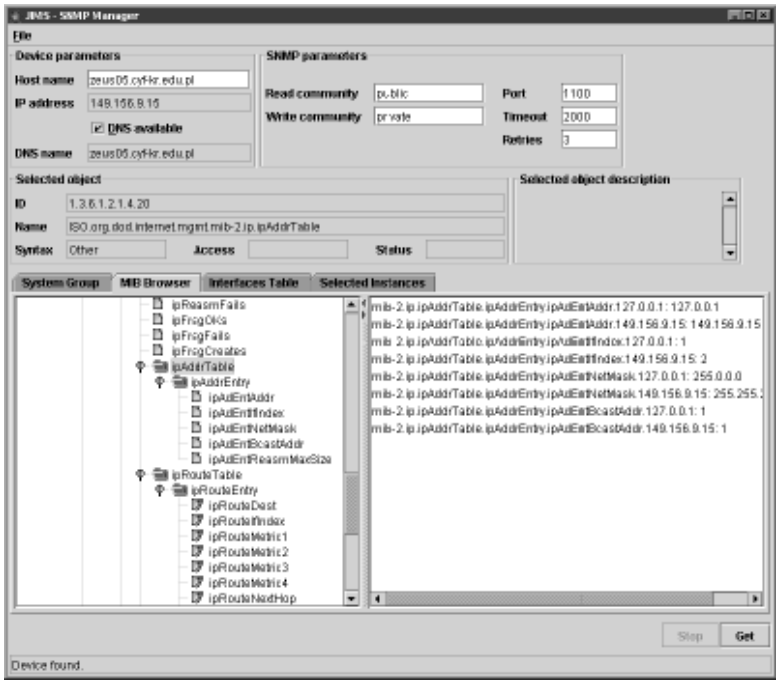

Figure 6. SNMP-based client application

characteristic than just the most recent state, as is the case with 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 other method is more scalable: a user is capable of selecting proper components relying on a component state, fixed in time constraints. The client application uses a query language specified by a monitored data warehouse interface. As an example of such query, let us consider: Select all computers that CPU load has been greater than 80% during last hour. After selecting proper components the user is able to specify time constraints in which the client wants to review the components' state. The subsequent action is to determine the way in which 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 the selected attributes.

Another interesting feature of this client is its built-in internationalization support. It is so far possible to customize the user interface to work in English or Polish. An example of such client is presented in Figure 7.

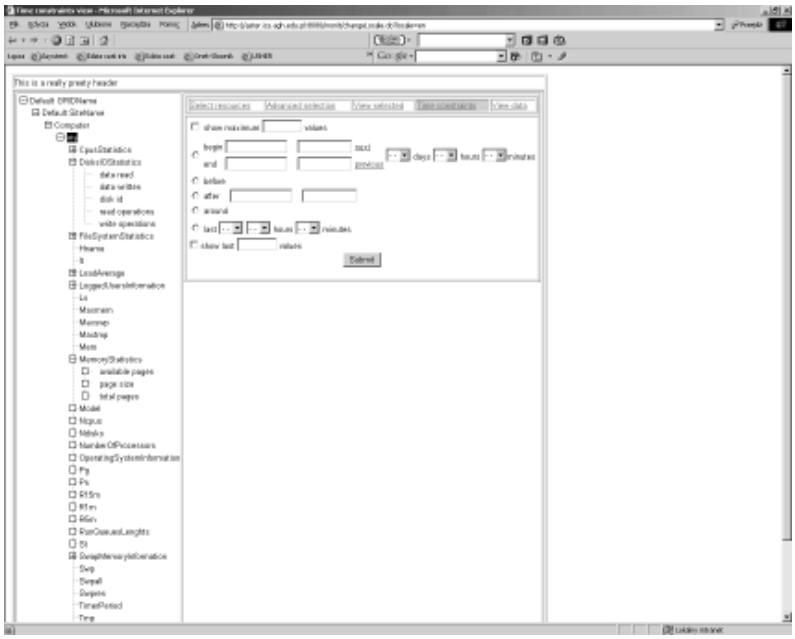

Figure 7. Data warehouse client

7. Conclusions

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

Considering the storage and accessibility of monitoring data, the EJB technology seems to be a very natural choice. Efficient use of this technology requires great care, especially when persistency mechanisms are implemented. Direct JDBC-based access to the database seems to be the most appropriate for monitoring data storage. The different technologies employed result in different client applications offering access to monitoring data collected by JIMS. An advantage of the proposed solution is that it is based on open standards, which makes JIMS easier to extend and deploy in any environment.

Acknowledgements

We would like to thank our students, L. Bizon, B. Lawniczek, G. Majka, J. Midura, M. Rozenau, T. Sekman and M. Smet, for their valuable contribution to the development of JIMS.

References

[1] Global Grid Forum, http://www.gridforum.org/
[2] McGovern J, Ambler S W, Stevens M E, Linn J, Jo E K and Sharan V 2003 *A Practical Guide to Enterprise Architecture*, Prentice Hall PTR
[3] Sun Microsystems *Java™ Management Extensions™*, http://java.sun.com/products/JavaManagement/
[4] Sun Microsystems *Java™ Management Extensions™ Remote API 1.0*, http://developer.java.sun.com/developer/earlyAccess/jmx/
[5] Jasnowski M 2002 *JMX Programming*, John Wiley&Sons Inc.
[6] Sun Microsystems *Jiro™ Technology*, http://www.jiro.org/
[7] *EU CrossGrid Project*, http://www.crossgrid.org
[8] Sekman T and Smet M 2004 *Dynamically Configurable System for Grid Resources Monitoring*, MSc. Thesis, AGH-UST Cracow (in Polish)
[9] Midura J, Balos K and Zieliński K 2004 *Global Discovery Service for JMX Architecture*, ICCS Cracow
[10] Balos K, Bizon L, Rozenau M and Zieliński K 2003 *Proc. CGW'03 Workshop*, Cracow, Poland, pp. 245–253
[11] *Westhawks Java SNMP v4.13*, http://snmp.westhawk.co.uk/
[12] Laurentowski A and Zieliński K 2002 *Integracja systemow B2B: JMX*, TeleNetForum
[13] Gamma E, Helm R, Johnson R and Vlissides J 1994 *Design Patterns*, Addison-Wesley
[14] Balos K and Zieliński S *JIMS the JMX Infrastructure Monitoring System*, http://www.eu-crossgrid.org/Seminars-INP/JIMS_monitoring_system.pdf
[15] The Open Grid Services Infrastructure Working Group (OGSIWG), Specification, http://www.ggf.org/ogsi-wg/
[16] The Open Grid Services Architecture Working Group (OGSAWG), Globus Tutorial, http://www.globus.org/ogsa/
[17] Sun Microsystems *Java Dynamic Management Kit*, http://java.sun.com/products/jdmk/
[18] Stallings W 1993 *SNMP, SNMP v2 and CMIP The Practical Guide to Network Management Standards*, Addison-Wesley
[19] Marinescu F 2002 *EJB Design Patterns*, John Wiley&Sons Inc.
[20] Roman E and Ambler S W 2002 *Mastering Enterprise JavaBeans*, John Wiley&Sons Inc.