UNIT4 - Platform SNMP Provider

Alejandro Aguilera Ruiz, *Universitat Autònoma de Barcelona.*

Abstract—UNIT4 is a multinational dedicated to the software industry for business management; the Research and Development department develops and uses the *karat* platform to deploy their products and services. The objective of this Project was to publish management information from the *karat* platform in order to administrate *karat* servers using third party market tools, which support network or application management standards like the Simple Network Management Protocol or Java Management eXtensions. A Java Application was developed to achieve the objective and meet the requirements of the Project, using an iterative agile methodology and providing not one, but two standard management services for the whole platform. *Karat* servers can now be administered by most of the commercial network management systems of our days.

Index Terms— Java Management eXtensions, Management Information Base, network management, network monitoring, Simple Network Management Protocol

Abstract (cat)—UNIT4 és una multinacional dedicada a la indústria del software per a la gestió empresarial; el departament d'Investigació i Desenvolupament desenvolupa i utilitza la plataforma *karat* per desplegar els seus productes i serveis. L'objectiu d'aquest Projecte va ser la publicació d'informació administrativa de la plataforma *karat* per tal de gestionar els servidors *karat* utilitzant eines de mercat de tercers, que suporten estàndards de gestió de xarxes o d'aplicacions com el Simple Network Management Protocol o les Java Management eXtensions. Una aplicació Java va ser desenvolupada per aconseguir l'objectiu i complir amb els requeriments del Projecte, utilitzant una metodologia iterativa àgil i proveint no un, sinó dos serveis estàndard de gestió per a la plataforma. Servidors *karat* ja poden ser administrats per la majoria dels sistemes comercials de gestió de xarxes dels nostres dies.

Paraules Clau— gestió de xarxes, Java Management eXtensions, Management Information Base, monitorització de xarxes, Simple Network Management Protocol

1 INTRODUCTION

BEFORE the adoption of the Internet Protocol Suite (TCP/IP) in the early eighties [1], companies that wanted to manage their primitive network had only one option: to rely that their network vendor would provide a highly expensive Network Management System that operated at physical layer. Then TCP/IP occurred and Network Management evolution began, interoperability became a huge concern and standardization efforts started.

It was in 1988 when "A Simple Network Management Protocol" (SNMP) was introduced, as an answer to the need of a network management standard for TCP/IP network vendors and operation communities [2], and since then a several number of technologies have appeared with one goal: to guarantee the higher availability possible to the managed devices.

UNIT4 is a multinational company that develops and deploys services and products for business management, and it is their obligation to ensure the availability, performance and security of their services to the customer.

The Research and Development Department (R&D) of UNIT4 (Spanish branch) currently develops and uses the *karat* platform, fully written in Java. On this platform all products are installed and the configuration is done by using the karat Control Panel. The panel is the central point for all the configuration and management of karat servers, but it presents 3 problems:

1. It does not store historic data.
2. It does not use notification or alarm mechanisms.
3. It does not comply with any network or application management standard.

Platform SNMP Provider is a project developed in the context of an internship program, where network management is approached from the vendor point of view; the objective is to publish management information from the *karat* platform, in order to provide the user with the ability to administrate these servers by means of market tools which support network or application management standards, like SNMP, Java Management eXtensions (JMX) or Web-Based Enterprise Management (WBEM) technology.

This article begins with the state of the art regarding network and application management, followed by an explanation of the methodology used in order to accomplish the objectives. At this point, the achieved results will be presented and this will finally allow us to evaluate the Project in the conclusion section and show the possible future work.

- *Contact e-mail: alejoag9900@gmail.com*
- *Specialization in: Software Engineering*
- *Work supervised by: Oriol Ramos Terrades (Computer Science Department) and Ezequiel Parra Mestre (UNIT4-Iber)*
- *2013-2014 course*

2 State of the Art

Nowadays, network management is a critical factor to consider in any company. A common way of characterizing network management functions is the 5 category acronym FCAPS, which stands for Fault, Configuration, Accounting, Performance and Security. It is the model and framework of the ISO Telecommunications Management Network Group [3].

Many network management systems provide the full FCAPS capability, though in this project we are especially interested in fault and performance. Therefore, technologies that allow this kind of network management are: Simple Network Management Protocol (SNMP) [4] and Web Based Enterprise Management (WBEM) [5].

On the other hand, since the *karat* platform is fully written in Java, the technology that provides management functionality for this language is called Java Management eXtensions (JMX) [6]. There is also a WBEM API specification under the Java Community Process JSR-48 [7].

All these technologies and standards were considered in order to choose the best option for the Project; the description of each one will be explained further in the following subsections.

Before the description of each technology, some basic terms shared in network management should be explained. In a typical network environment, there are one or more administrative computers, called managers; their function is to monitor or manage a host or a group of hosts.

In order to do that, they need some sort of software component in each host, which will gather the information and perform the management tasks, this is called an agent.

The manager on the other hand runs several components or drivers which are compliant with many technologies for network management (SNMP, JMX and/or WBEM amongst them), so is often called a Network Management System (NMS) instead of just manager because it integrates all these monitoring and management protocols, standards and/or tools.

2.1 Simple Network Management Protocol (SNMP)

SNMP is a component of the TCP/IP stack, as defined by the Internet Engineering Task Force (IETF). It consists on a set of standards for network management contained in a series of Request for Comments (RFC) documents [8], including:

1. An application layer protocol: Simple Network Management Protocol or SNMP (RFCs 3411-3418).
2. A database schema: Structure of Management Information Version 2 or SMIv2 (RFC 2578).
3. A set of data objects: Management Information Base Version 2, usually called MIB-II or MIB-2 (RFC 1213).

An SNMP manager can query data from the agent or agents and also set this data. SNMP agents on the other hand gather data from the host and reply to the managers requests; in addition, they can send alarms or notifications asynchronously to the manager.

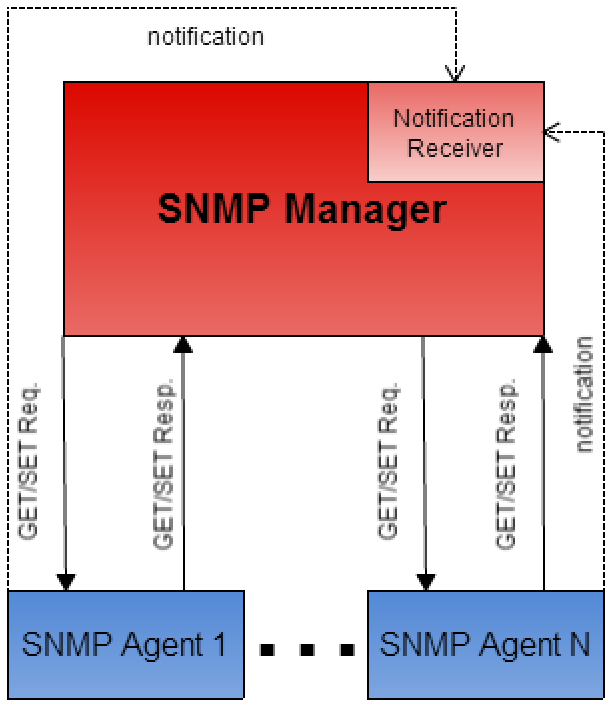


Fig. 1. Architecture of SNMP. SNMP Managers send SET and GET requests; SNMP Agents send the respective responses. In addition, SNMP Agents can send asynchronous notifications.

2.2 Java Management eXtensions (JMX)

Java Management eXtensions is the Java technology that supplies tools for managing and monitoring applications, devices or service oriented networks. It is not a protocol but a less restrictive technology that does not force any transport or communication methods, instead provides different options for those purposes. It uses a 3 level architecture [9]:

1. Probe level: includes the probes (classes) that instrument the information. They are called MBeans.
2. Agent level: it is a server where all MBeans are registered. The implementation is called MBean Server.
3. Remote management level: enables other applications to access the MBean server, via connectors using various communications methods (RMI, IIOP, etc.) or via adaptors to other protocols (SNMP, HTTP, etc.)

A JMX manager application accesses the MBean Server and therefore the MBeans; the manager can get and/or set attributes and call methods remotely, which are defined in each MBean (see Figure 2).

Similar to SNMP, MBeans can also send asynchronous notifications, but in this case the manager will have to subscribe explicitly to each MBean in order to receive them.

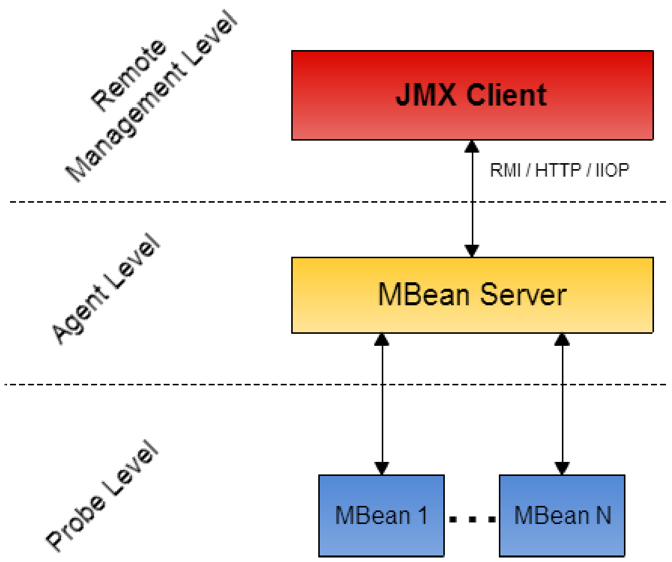


Fig. 2. Architecture of JMX. JMX Clients reach the MBean Server over various connectors (RMI, HTTP, IIOP and others). MBean Servers contain the MBeans which are the probes that manage the resources and send asynchronous notifications.

2.3 Web-Based Enterprise Management (WBEM)

Web-Based Enterprise Management is a set of management and Internet standard technologies developed to unify the management of distributed computing environments, facilitating the exchange of data across different technologies and platforms. It includes:

1. A data definition: Common Interface Model (CIM).
2. A transport/encoding method: CIM-XML.
3. An access mechanism: HTTP.

A WBEM manager application will find the WBEM server and then the devices; the manager will send encoded messages, commonly containing requests, to the WBEM server via HTTP.

The server will decode the requests and consult the model of the device to determine how to handle the request. The interaction is made by the providers, which are the agents in this infrastructure; the providers interact with the real hardware or software and make the respective system calls (see Figure 3).

CIM defines a synchronous alarm mechanism containing indications and triggers. Triggers are the criteria's for an object or set of objects; triggers contain a temporary instance of the object that emitted the alarm, this is called an indication. Triggers stay on the manager until acknowledged.

2.4 Comparison

In brief, the architecture of all these technologies is very similar; they mainly differ in the data definition. SNMP data definition is much more restrictive (defined by MIB-2) while JMX and WBEM are object oriented and more flexible.

Obviously alarm and notification mechanisms are affected by the data definitions, so in JMX and WBEM alarms are more flexible than in SNMP. In JMX and WBEM criteria for each object or even instance can be defined while in SNMP alarms are bound to a specific agent and MIB-2 data set implementation.

It is evident that any of these 3 options would help us to accomplish our objective and at first sight it is difficult to determine which one is more appropriate, but because of the wide use of SNMP in the current Network Management Systems, it seemed this was the path to take.

However, the team that develops the *karat* platform expressed their interest in the implementation of JMX technology, since it is the standard in Java; subsequently both options were chosen for development.

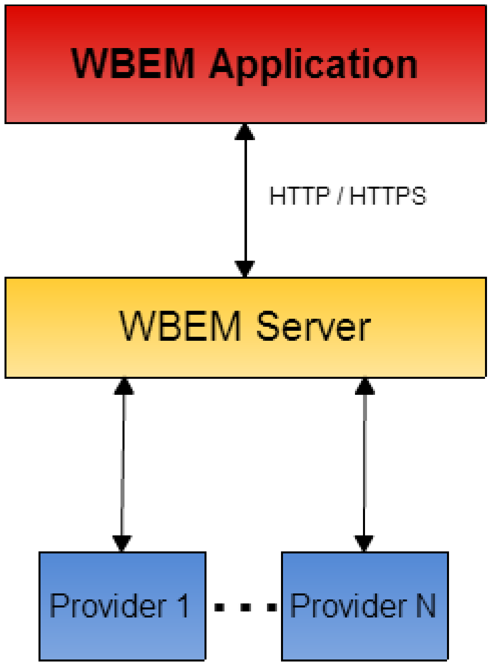


Fig. 3. Architecture of WBEM. The WBEM application sends encoded CIM-XML requests to the WBEM server. The WBEM server processes them and reaches the correct provider. The providers manage the resources and make the system calls if needed.

3 Methodology

For the correct development of the Project and in order to accomplish the objective, the Scrum methodology was employed [10]. At UNIT4, all the teams use this methodology and it was important to keep this Project in line with the rest.

Coupled with the Scrum methodology, some development constraints were considered and acknowledged.

First of all, the whole Project had to be developed in Java, because karat and its products are written in this programming language (the control panel amongst them).

In consequence and to fulfill the need of software configuration management, a Subversion (SVN) versioning and revision control system [11] was used, for the company had all the infrastructure and tools ready for the process.

The configuration management rules were simple: 1) only commit correct code to the repository (verified and properly tested), and 2) a single branch will be used for the whole Java Project. The first rule implied that the development process would occur in local, and the second rule implied that there would be no parallel development, since one person was to be developing the Java Project at any time.

3.1 Scrum

Scrum is an iterative and incremental agile methodology that challenges assumptions of the traditional sequential approach. It acknowledges that in the course of a project, customers can change their mind about their needs, leading to unpredicted changes; therefore scrum teams focus on delivering software constantly and responding to emerging requirements, instead of fully understanding or defining the problem, which is most of the time impossible.

In Scrum, cycles or sprints are defined in a closed time box, usually 3 to 4 weeks long. These cycles are composed by stories which are atomized requirements. A story is functionality or a set of functionalities that provide value to the customer and help the team accomplish the objectives.

The product backlog is an ordered list of everything needed in the project. It is the ensemble of all the stories, by priority order. This artifact is never complete, since during the project it evolves with new stories, fixes, changes, needs, etc. It assumes that software can always be improved and extended.

The initial definition of the stories and their measurement is realized at the beginning of the project, this process is called sizing. Stories are measured in points, the value of a point is measured in hours and it is commonly about 15 hours (2 days of work), depending on the working environment and schedule.

To complete a story we need to complete one or several tasks. A task is the minimum unit of work. Tasks are measured in hours and it is not recommended that they exceed 2 days of work. In this project the following tasks were considered as a template:

1. Analysis.
2. Construction or development.
3. Test writing.
4. Test verification.
5. Documentation.

The analysis task allows examining the needs and risks of implementing the story; normally this task is used for requirement elicitation and objective clarification for the project, the story or even the sprint itself.

In the construction task, the design and implementation of a usable piece of software is made, this deliverable needs to provide value to the customer and therefore, it covers a wide spectrum of elements: documents, diagrams, applications, prototypes, mock-ups and several others.

The test writing task defines with more precision the scope of the deliverable component, including its quality. In addition with this task, the test verification task allows the programmer to validate the content of the story and perform any regression tests from previous stages of the process.

Finally in the documentation task, the preparation for the delivery is made. This means that all documents, diagrams, installers, user manuals and test results are adapted, completed and fixed for the client who will receive the component. The client will have to be able to use the component and return some feedback.

However, it should be kept in mind that some stories might not need all of this standard tasks. A market study for example, where tasks like test writing or verification make no sense at all. Similarly, the order of these tasks is not fixed or forced in any way; this allows flexibility to the development process, making it highly customizable for each programmer, team or organization.

On the other hand, the planning of the project is done in 2 phases. In the first phase the quantity and length of the sprints is fixed, defining the final effort (in points) that will be needed in each sprint. As a result, the team can determine the scope of the project by placing stories inside the sprints. In the second phase an iterative process begins.

Given a first baseline of the scope of the Project, the first sprint is filled up with stories until it reaches its maximum work effort, then the sprint is carried out until the times runs out. When this occurs, the sprint is reviewed and the next sprint is planned. This process is called sprint planning, and it allows perceiving each sprint as a closed project, with its own goals, stories and tasks.

Granted that a sprint is planned, each team member will have to pick a story and develop it, during this time 2 situations can happen: 1) the time runs out and the unfinished stories need to return to the product backlog, to be planned for the next sprint, or 2) all stories are finished and the sprint can be extended with more stories. Obviously, the second situation is the least common.

During the period of time in which each team member is working in a sprint, every day at the beginning of the working schedule, the team will gather and they will exchange information, regarding these 3 questions:

1. What did I do yesterday that helped the development team meet the sprint goal?
2. What will I do today to help the development team meet the sprint goal?
3. Do I see any impediment that prevents me or the development team from meeting the sprint goal?

This way, the team is aware of the sprint development and problems are detected early. This meeting, called daily scrum meeting, improves communications, eliminates other meetings, highlights and promotes quick decision making and improves the team level of knowledge. This is a key meeting in Scrum methodology.

At the end of each sprint 2 more meetings are consummated: 1) the review meeting and 2) the retrospective meeting.

In the review meeting the team talks about what was accomplished in the sprint, they make changes to the product backlog and discuss possible stories for the next sprint.

In the retrospective meeting a more deep analysis is performed, it's the opportunity for the team to discuss improvement to their work approach and the decision making process; the purpose is to detect what went well and what went wrong, regarding people, social relations, processes and tools.

Generally speaking, the methodology was followed almost by the book; the only activity that wasn't carried out was the daily scrum meeting, because the whole Project was done by one team member (this author) and it seemed redundant to perform such a communication task to itself; Even so, review and retrospective meetings were carried out with the designated Project tutor at UNIT4.

3.2 Project Planning

At this point, given the base concepts of the methodology and the development constraints, the Project planning can be introduced.

After the sizing of the product backlog it was evident that, because of the length of the Project (560 hours), all the stories could not be completed; In consequence, the stories were prioritized according to the supposed value they would provide to the end client. The value of each story was determined by the Project tutor and this author.

At a first instance, the length of the sprints was determined, resulting in the following date lines:

1. Sprint 0: from 11/02/2014 to 10/03/2014.
2. Sprint 1: from 11/03/2014 to 31/03/2014.
3. Sprint 2: from 01/04/2014 to 12/05/2014.
4. Sprint 3: from 13/05/2014 to 30/05/2014.
5. Sprint 4: from 31/05/2014 to 27/06/2014.

In Figure 4 there is a Gantt diagram showing the length in hours for each sprint.

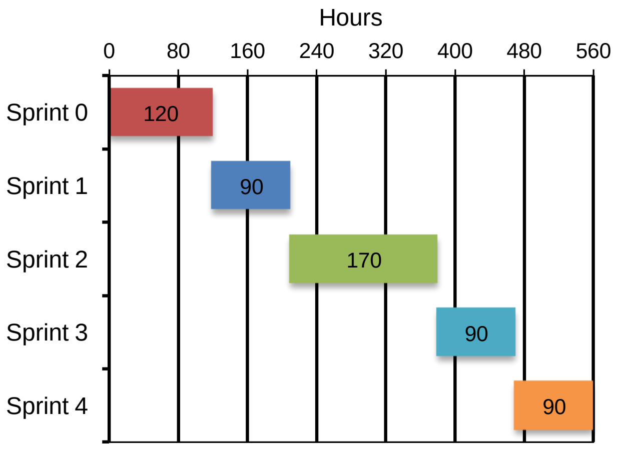


Fig. 4. Gantt diagram of the Project's planning. Vertical axis shows the sprints numbered from zero to four, horizontal axis shows the hours of work needed for each sprint. A total of 560 hours were scheduled for the Project.

In the first sprint, the objective was to study the protocols chosen, to construct some examples of these protocols and to study the karat Control Panel.

The second sprint objective was to construct the base of the server, implementing session monitoring. An intermediate presentation was scheduled by UNIT4 in order to check the Project status and its development.

In the third sprint, 3 main tasks were required: first a deployable of the Java Project was needed, it should provide configuration parameters and guarantee security; secondly, a study for market tools (JMX and SNMP) and thirdly, further development of the server was to be achieved: server monitoring.

For the fourth sprint, the alarm and notification mechanisms were first priority. Also a discovery tool for SNMP and JMX servers was to be developed.

In the last sprint, more development of the server was scheduled: cache and property files refreshment. A study of the WBEM protocol was also planned in order to study the feasibility of the Project in this matter, although the Final Presentation was the main concern of the sprint.

Nevertheless, the effort in sprint 4 was not measured correctly; alarm and notification mechanisms (especially for SNMP) needed a lot of effort. In consequence the discovery tool was postponed for the last sprint.

In addition, the Project closure and the final adjustments for the integration of the Java Project in the *karat* platform were not considered in the initial planning. This together with the discovery tool made it impossible for the study of the WBEM protocol and the cache/property files refreshment to take place.

4 Results

As a result of the iterative methodology followed and the Project planning, a Java Application was developed as a dual agent wrapper (SNMP and JMX). The application was built in 5 sprints, each of them with a different objective. The results of each sprint are detailed in the following subsections.

4.1 Sprint 0: Research and Study

In the first sprint of this Project, research and study tasks were carried out with the objective of acquiring knowledge regarding SNMP and JMX. This was achieved by redacting definition documents and building prototypes for both technologies.

The first activity exposed the purpose and the architecture of each option; the second one allowed acknowledging the limitations, advantages and disadvantages in a more practical context.

JMX technology was easy to use and to develop, a class instrumenting a resource has to implement an interface with the termination "MBean" after the name, for example *Server*MBean; in this interface attributes are defined by the getters (read) and setters (write), other methods can be defined as well.

To register the MBeans, an MBean server is needed; the JVM MBean server can be obtained through a static method but a custom MBean server can be created and used as well; the registration method requires the instance of the object (the MBean itself, implementing the interface) and a unique name. The name is composed by the domain and a list of properties, for example: "com.mbeanexamples.server:id=server1". The properties usually help identify and classify MBeans under the same domain.

SNMP on the other hand had a little more work to do and it needed from an open source library (SNMP4J [12 /*SNMP4J*/]) to handle the communication and message processing.

The resources can be instrumented in classes as well, but they need to be mapped into MIB objects. MIB objects can be seen as a composed data type, containing syntax, an access and an Object Identifier or OID; the syntax is in essence the data type, Integer32 for example, the access defines if the object can be read, written or both, the OID identifies the object in the MIB, which follows a tree-like structure.

For our study case, the data definition had to occur in 2 places: 1) the source code and 2) the vendor MIB definition. For the first one, the library was used to map the classes into MIB objects and for the second one, a vendor MIB had to be written from scratch in SMIv2 syntax. As you can see the work is double, first the source code definition and implementation and then the vendor MIB writing.

When the previous analysis was completed, research about the *karat* Control Panel started; this tool was the perfect example of getting managing information from the karat Platform, functionality we would later use for the Monitor Server.

The *karat* Control Panel makes a TCP connection to a Main Server to obtain its administrative information. Each Main Server can host several servers, these "common" servers are identified by their MAC address. Applications, services and products are hosted under each server (see Figure 5).

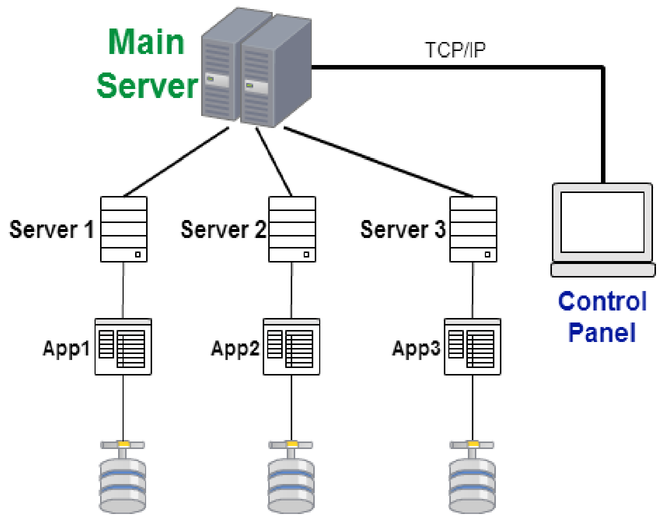


Fig. 5. Architecture of the *karat* Platform. Main Servers host different servers and these servers are the ones offering the services or hosting the applications. The Control Panel connects to a Main Server and asks for its administrative information (servers, sessions, licenses and more).

The administrative information published by the Control Panel is organized in eight tabs:

1. Servers and related information.
2. Configurations for the *karat* platform.
3. Active client sessions.
4. Installed licenses.
5. Licenses being used at the moment.
6. SMS service configuration
7. Certificate store, to administrate and distribute the security certificates for all the clients.
8. Clusters configuration, to add/delete servers and configure them mainly for load balancing.

In the meantime, the formal requirement document was made. This document included the description of the control panel, the justification of the Project, the alternatives considered and the proposal for the Project. The proposal contained deadlines, costs and a first benefit/risk analysis.

4.2 Sprint 1: Server Base

In the second sprint a first version of the application was developed. MonitorServer was divided in 3 modules:

1. DATA.
2. SNMP.
3. JMX.

The DATA module connected to a *karat* Main Server and obtained the administrative information just as the *karat* Control Panel, the SNMP and JMX modules first parsed this information into their respective data representations and then published this information using their respective protocols (see Figure 6).

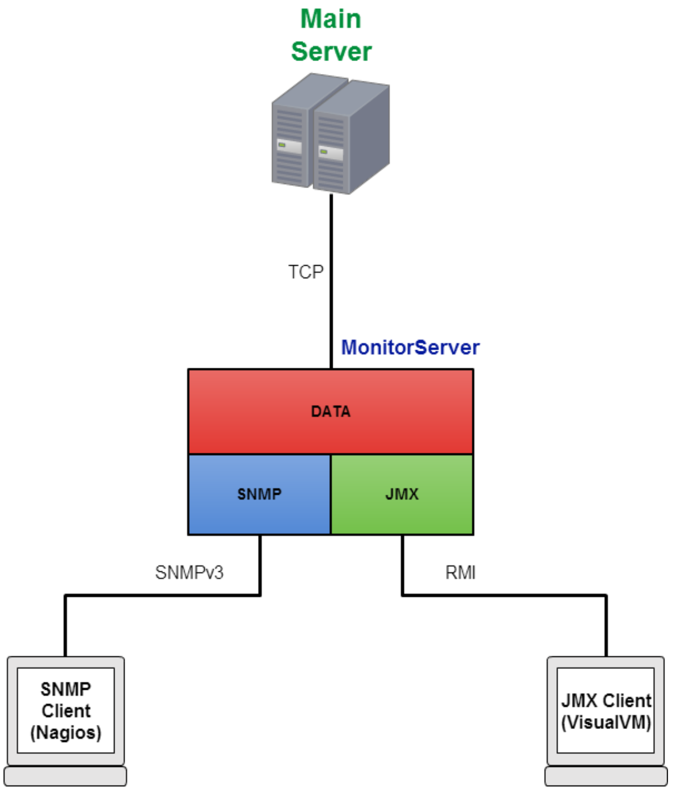


Fig. 6. Modular diagram of the MonitorServer Project. The DATA module gets the information from the Main Server, the SNMP module publishes it via SNMP version 3 and the JMX module publishes it via a JMX service URL, using an RMI registry.

SNMP (version 3) uses TCP and UDP transport protocols, one port for communication (161) and one for notifications (162); all this is handled by the SNMP4J library.

JMX on the other hand can use different varieties of connectors for transport, but the most used and the one used in this Project was the Remote Method Invocation (RMI) connector [13], which uses various TCP ports randomly chosen, although in this Project the RMI connector used a single TCP port specified in the JMX service URL.

The JMX service URL is the identification of the JMX connector and service (see Figure 7), it allows any JMX client to locate:

1. The target machine: where the service is hosted (IP address or hostname).
2. The RMI server port: the port where the RMI server is hosted.
3. The RMI registry port: the port where the RMI registry is located.

```
service:jmx:rmi://<TARGET_MACHINE>:
<JMX_RMI_SERVER_PORT>/jndi/rmi://
<TARET_MACHINE>:<RMI_REGISTRY_PORT>/
jmxrmi
```

Fig. 7. JMX service URL. The TARGET_MACHINE is the machine where the service is hosted (IP address or hostname), JMX_RMI_SERVER_PORT is the port where the RMI server is hosted and the RMI_REGISTRY_PORT is the port where the registry is located. The rest are constants of the URL (service, jmxrmi, jndi …).

This first version of MonitorServer published only the list of sessions in JMX and SNMP. Additionally, since JMX technology allows remote invocation of methods, the JMX module of the server was capable of killing a session remotely.

In this sprint, a first presentation was requested by UNIT4. The company wanted to trace Project objectives, schedule and the work done until then (02/04/2014); the activity included a demo of the server base.

The demo showed that tabular information in JMX was not user friendly because only one row could be seen at a time, the suggestion was noted and in the sprint review a new story was added to the product backlog.

4.3 Sprint 2: Configuration, Security and Evaluation of Market Tools.

In this sprint, security and configuration were the main focus.

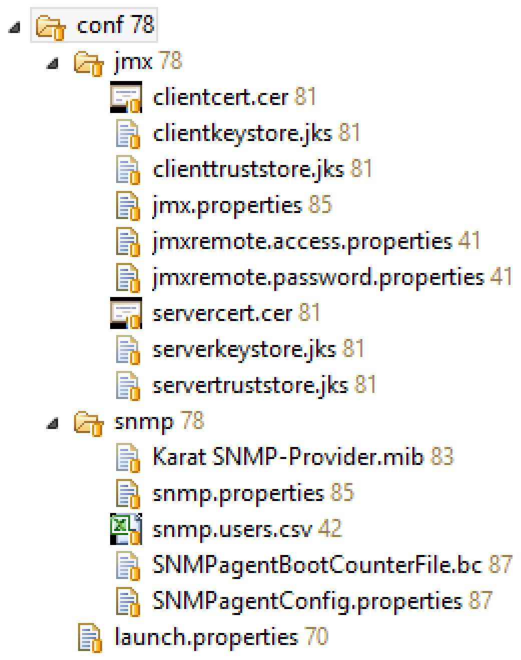
First of all, configuration parameters for both servers were moved to property files and a launcher was created. The configuration folder "conf" (see Figure 8) contained the launcher configuration file and two more folders, one for SNMP configuration and another for JMX configuration.

The launcher set up the properties for each module and allowed starting one server or both of them; launcher properties included the Main Server address and port, and the admin password for authentication. This was common for the other two modules (JMX and SNMP).

For JMX, the main configuration file "jmx.properties" included the hostname and port to publish the JMX connection; regarding security, SSL was used together with access and password files [14]:

1. *client/server* SSL certificates.
2. *client/server* keystore: a container of private keys.
3. *client/server* truststore: a container of public keys.
4. jmxremote.access file: the users and their role.
5. jmxremote.password file: the users and their password.

Keystores and truststores are created using the *keytool* command tool [15], it allows the creation of public and private keys within a store, the keys can be exported as certificates later. For this set up, the first step is to create both private keys for the server and the client within keystores, then the public keys need to be exported as certificates, finally the certificates have to be imported into the truststores, the client must trust the server and the server must trust the client.



```
conf 78
  jmx 78
    clientcert.cer 81
    clientkeystore.jks 81
    clienttruststore.jks 81
    jmx.properties 85
    jmxremote.access.properties 41
    jmxremote.password.properties 41
    servercert.cer 81
    serverkeystore.jks 81
    servertruststore.jks 81
  snmp 78
    Karat SNMP-Provider.mib 83
    snmp.properties 85
    snmp.users.csv 42
    SNMPagentBootCounterFile.bc 87
    SNMPagentConfig.properties 87
  launch.properties 70
```

Fig. 8. Configuration file directory. The file launch.properties contains information needed to reach a *karat* Main Server, the JMX folder contains security and configuration files for the JMX agent and the SNMP contains the user's file, the configuration file and the vendor MIB for the SNMP agent.

For SNMP, the main configuration file "snmp.properties" included the address and port to be used by the agent.

Regarding security, SNMP version 3 supports MD5 and SHA protocols for authentication, and DES and AES (128 bits version) for privacy [16]. A unique user's file was created to provide the configuration: "snmp.users.csv"; the file had 6 fields:

1. Security name or user name.
2. Role: defines if authentication or privacy protocols are to be used.
3. Authentication protocol: MD5, SHA or none.
4. Authentication password
5. Privacy protocol: DES, AES or none.
6. Privacy password

Although it is advised to use authentication and privacy protocols, users that don't use any of them can be created (no authentication and no privacy); for this reason View-Based Access Control Model (VACM) [17] was implemented, to restrict the access to these users. Basically, not authenticated nor privacy protected users have no write privileges and restricted read access while authenticated and privacy protected users have full read/write privileges and access.

The vendor MIB, "Karat SNMP-Provider.mib" in Figure 8, was placed inside the SNMP folder. SNMP clients need to import it in order to understand the MIB objects supported by the agent and perform the requests.

The remaining two files in this folder were automatically created by the SNMP4J library; the files contain serialized non-volatile information:

1. SNMPagentBootCounterFile.bc: information about values of MIB objects.
2. SNMPagentConfig.properties: configuration of the agent.

With the security assured, the functionality to publish servers and their hierarchical information was developed. Services, processes and configuration files were published and available for management via JMX and SNMP; in the case of the services, JMX allowed starting or stopping them remotely.

For the SNMP part of the Project this included an update of the vendor MIB. It is not possible to nest tables in SNMP, so server information had to be mapped into 4 tables, one for the servers themselves and the remaining 3 for the services, processes and configuration files. The last 3 tables were indexed by server MAC in order to distinguish from which server each service, process or configuration file came from.

The improvement of tabular information in JMX was solved in this sprint; the solution was to register each row from the tables as a different MBean object and group MBeans using properties.

The improvement represented a major visual update for the JMX side of the Project; re-design was needed in order to implement this solution.

When the previous stories were finished, market tools were installed to be linked with MonitorServer. This showed that karat servers can be monitored and administrated by any JMX monitoring tool, or SNMP compliant NMS. AggreGate Network Manager v5.01 [18] and PRTG Network Monitor v14.1 [19] were used for SNMP testing; AppDynamics Application Performance Management v3.8 [20] was used for JMX.

However, it was noted that JMX tools for production environments were scarce or insecure while SNMP tools were abundant. In contrast, some SNMP tools weren't able to import vendor MIBs and were restricted only to MIB-2 default objects.

4.4 Sprint 3: Notifications and Alarms

The wish of providing notification or alarm mechanisms for *karat* servers was especially considered, and in this sprint, SNMP and JMX notifications were implemented.

SNMP notifications are defined in the vendor MIB and assembled in the agent, they contain: the source, the SNMP version and the variable binding (see Appendix A1). The variable binding is a set of variables, system uptime and trap OID are mandatory.

JMX notifications are defined in the MBean, they contain: the timestamp, the type of the notification, the sequence number, a message and the MBean instance name (see Appendix A2).

Four different alarms, relative to servers, are triggered automatically:

1. Server down: the server is not responding to requests.
2. Memory threshold exceeded: memory exceeds a certain threshold.
3. CPU threshold exceeded: the CPU hosting the karat server exceeds a certain threshold.
4. Maximum session number exceeded: the number of sessions in this server exceeds a certain limit.

The last 3 alarm thresholds were placed as properties in the launch configuration file; the notification rate for each agent was placed in each of their main configuration files (snmp.properties and jmx.properties).

For SNMP, a new user role was defined: the admin role; SNMP notifications are targeted to a user role so only admin users receive them; admin users must be authenticated and privacy protected.

Despite of the fact that market tools were able to detect and show notifications for both agents, the tools advised to use their alerting mechanisms.

Testing some of the market tools used in the sprint 2, it was noted that their alarm mechanisms were much more configurable. They allowed configuring multiple alerts for each instance or object and there were many different types of them: upper threshold exceeded, lower threshold exceeded, state changed, host not reachable, host packet loss threshold exceeded, and several others.

4.5 Sprint 4: Server Discovery Tool and Project Closure.

In the final sprint, a server discovery tool was developed using a multicast self-made protocol.

Each server needed to run a multicast listener thread all the time, when a new client joined the network it would query from active servers in a multicast channel; the servers would answer with a description, containing location and services offered (JMX and SNMP monitoring among them).

Additionally, SNMP auto-discovery is a feature of most SNMP clients (market tools) and no further functionality is needed for that; it consists in sending an SNMP request to all network hosts to the default port (161) and listening for the replies.

At last, the final adjustments to the server were made; before integration, some security, unit and integration tests were carried out. Some mistakes included user access, default ports and default addresses.

MonitorServer was integrated in the karat platform v9.1.1.0 as a service. The Project was presented with a final demo, using Java VisualVM v1.7.0_51[21] for JMX and AggreGate Network Manager v5.01 [18] for SNMP.

5 Conclusions and Further Work

The main objective of the Project was accomplished satisfactorily since management information of *karat* servers can be exposed through the SNMP protocol version 3 and JMX technology in a secure way. Managing *karat* servers through market tools that support these protocols is now possible.

First of all, it should be acknowledged that high availability of any system or service is a key factor to be aware of, for this reason it is very important to be able to administrate networks and applications in a standard way.

Even though JMX is a very powerful technology that provides means of remote method invocation, the lack of market tools for production environments makes JMX monitoring by market tools an odd choice. Custom solutions are preferred, like the *karat* Control Panel which is more user-friendly.

SNMP is security reliable and much more flexible in data access levels, although is more restrictive regarding alarm mechanisms and data definition; However, notification mechanisms are better supported by market tools which are protocol independent, therefore the importance of this functionality is minimum.

By consequence, this author concludes that the choice of SNMP as a protocol was more accurate for the purpose of the Project, mainly because SNMP is widely used and supported by many NMS.

Future improvement has yet a lot of room. For a starter, lot of information from the control panel is still missing in the agents, valuable information like SSL certificates, licenses and server clusters could be published.

Additional development could include a WBEM provider, for the DATA module is independent from the rest of the modules.

Acknowledgment

This author would like to thank Jordi Pons, the Escola d'Enginyeria and by extension the Universitat Autònoma de Barcelona for the opportunity to carry out this Project in an Internship Program.

This work was supported by a grant from UNIT4 which is also worth of acknowledgment.

Last but not least, this author wishes to thank Ezequiel Parra and Arnau Escursell for their dedication, effort and time spent supervising the Project.

References

[1] R. Hauben, "From the ARPANET to the Internet," 23 June 1998. [Online]. Available: http://www.columbia.edu/~rh120/other/tcpdigest_paper.txt. [Accessed 26 June 2014].

[2] IETF, "RFC 1067 - A Simple Network Management Protocol," August 1988. [Online]. Available: http://www.ietf.org/rfc/rfc1067.txt. [Accessed 26 June 2014].

[3] ISO/IEC, "ISO/IEC 7498-4: 1989 Information processing systems -- Open Systems Interconnection – Basic Reference Model – Part 4: Management framework," 21 July 2006. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=14258. [Accessed 26 June 2014].

[4] IETF, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," December 2002. [Online]. Available: http://tools.ietf.org/html/rfc3411. [Accessed 26 June 2014].

[5] DMTF, "Web-Based Enterprise Management," 2014. [Online]. Available: http://www.dmtf.org/standards/wbem. [Accessed 26 June 2014].

[6] J. S. Perry, Java Management Extensions, O'Reilly Media, 2002.

[7] Oracle Corporation, "JSR 48: WBEM Services Specification," 2014. [Online]. Available: https://jcp.org/en/jsr/detail?id=48. [Accessed 26 June 2014].

[8] IETF, "Request for Comments (RFC)," 2014. [Online]. Available: http://www.ietf.org/rfc.html. [Accessed 26 June 2014].

[9] Oracle Corporation, "Trail: Java Management Extensions (JMX)," 2014. [Online]. Available: http://docs.oracle.com/javase/tutorial/jmx/index.html. [Accessed 26 June 2014].

[10] L. A. Salazar, "Scrum.org | Scrum Guide," July 2013. [Online]. Available: https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf. [Accessed 26 June 2014].

[11] C. M. Pilato, B. Collins-Sussman and B. W. Fitzpatrick, Version Control with Subversion, Sebastopol, United States: O'Reilly Media, 2008.

[12] F. Fock and J. Katz, "SNMP4J - Free Open Source SNMP API for Java," 2014. [Online]. Available: http://www.snmp4j.org/. [Accessed 26 June 2014].

[13] Oracle Corporation, "Trail: RMI," 2014. [Online]. Available: http://docs.oracle.com/javase/tutorial/rmi/. [Accessed 26 June 2014].

[14] Oracle Corporation, "Java Management Extensions (JMX) Java Platform Standard Edition version 7," August 2008. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/tutorial/tutorialTOC.html. [Accessed 26 June 2014].

[15] Oracle Corporation, "keytool - Key and Certificate Management Tool," 2014. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html. [Accessed 26 June 2014].

[16] IETF, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)," December 2002. [Online]. Available: http://tools.ietf.org/html/rfc3414. [Accessed 26 June 2014].

[17] IETF, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)," December 2002. [Online]. Available: http://tools.ietf.org/html/rfc3415. [Accessed 26 June 2014].

[18] Tibbo Technology Inc., "AggreGate Documentation: Network Management and Monitoring," 2014. [Online]. Available: http://aggregate.tibbo.com/docs/en/index.htm?network_management.htm. [Accessed 26 June 2014].

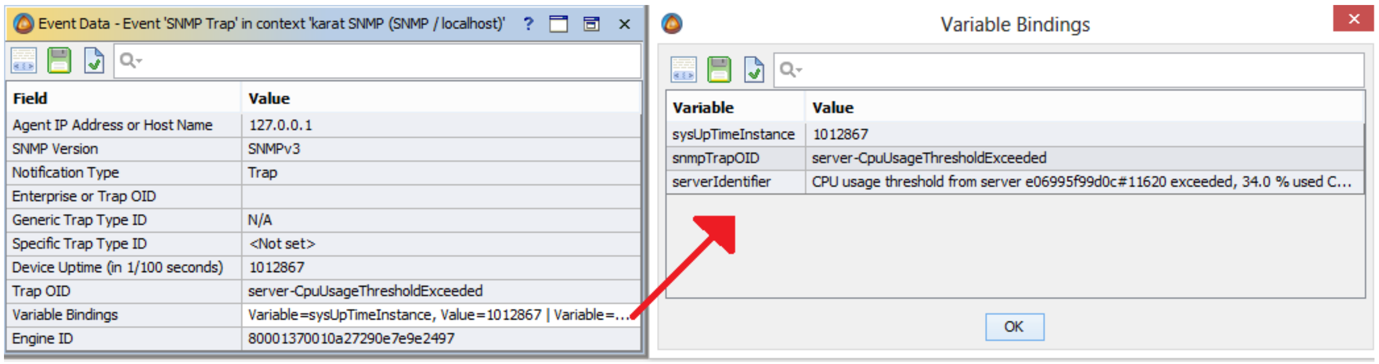
[19] Paessler AG, "PRTG Network Monitor - User Manual," 2014. [Online]. Available: http://www.paessler.com/manuals/prtg. [Accessed 26 June 2014].

[20] AppDynamics Inc., "Monitor JMX MBeans - AppDynamics Pro 3.8 Documentation," 2014. [Online]. Available: http://docs.appdynamics.com/display/PRO14S/Monitor+JMX+MBeans. [Accessed 26 June 2014].

[21] Oracle Corporation, "VisualVM," 2014. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/visualvm/. [Accessed 27 June 2014].

APPENDIX

A1. SNMP Notification Screenshot

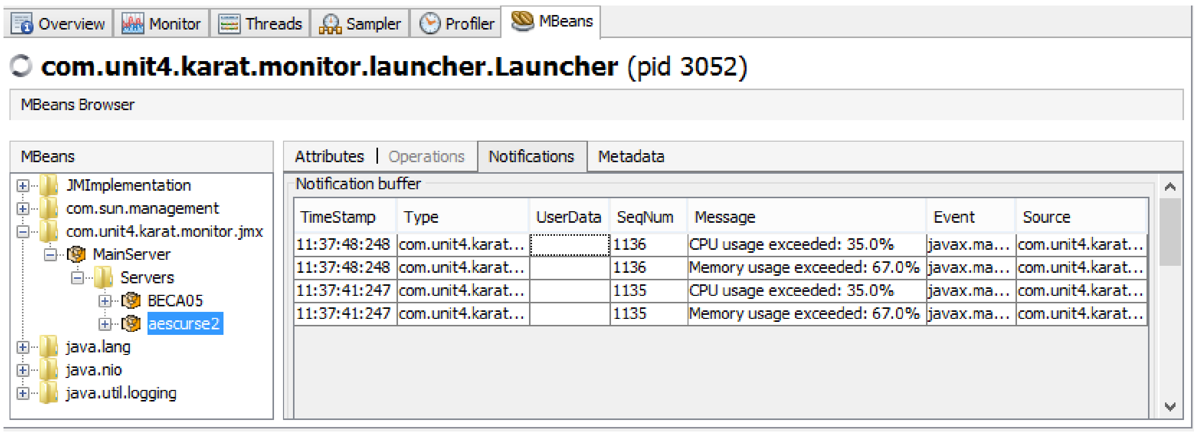

Event Data - Event 'SNMP Trap' in context 'karat SNMP (SNMP / localhost)'
Variable Bindings
OK

Field	Value
Agent IP Address or Host Name	127.0.0.1
SNMP Version	SNMPv3
Notification Type	Trap
Enterprise or Trap OID	
Generic Trap Type ID	N/A
Specific Trap Type ID	<Not set>
Device Uptime (in 1/100 seconds)	1012867
Trap OID	server-CpuUsageThresholdExceeded
Variable Bindings	Variable=sysUpTimeInstance, Value=1012867 | Variable=...
Engine ID	80001370010a27290e7e9e2497

Variable	Value
sysUpTimeInstance	1012867
snmpTrapOID	server-CpuUsageThresholdExceeded
serverIdentifier	CPU usage threshold from server e06995f99d0c#11620 exceeded, 34.0 % used C...

Left panel shows SNMP notification information: agent hostname, SNMP version, notification type, trap OID, variable bindings among others; right panel shows the original variable bindings, in this case: system uptime (mandatory), trap OID (mandatory) and the server identifier variable

A2. JMX Notification Screenshot



TimeStamp	Type	UserData	SeqNum	Message	Event	Source
11:37:48:248	com.unit4.karat...		1136	CPU usage exceeded: 35.0%	javax.ma...	com.unit4.karat...
11:37:48:248	com.unit4.karat...		1136	Memory usage exceeded: 67.0%	javax.ma...	com.unit4.karat...
11:37:41:247	com.unit4.karat...		1135	CPU usage exceeded: 35.0%	javax.ma...	com.unit4.karat...
11:37:41:247	com.unit4.karat...		1135	Memory usage exceeded: 67.0%	javax.ma...	com.unit4.karat...

Left panel shows the MBeans managed, the right panel shows the notifications received and their information: timestamp, type, sequence number, message, event and source. In this case we are receiving notifications from server *aescurse2*.