MonALISA

ing. B. Dorlandt, ing. H.J. Blok

4th February 2005

Contents

Introduction

In the past four weeks we have done research on MonALISA for the UvA and SARA as part of our study, System and Network Administration, at the University of Amsterdam (UvA). MonALISA stands for Monitoring Agents in A Large Integrated Services Architecture. The developers describe MonALISA as following:

> The MonALISA system provides a distributed service for monitoring, control and global optimization of complex systems. MonALISA is based on a scalable Dynamic Distributed Services Architecture (DDSA) implemented using Java / JINI and Web Services technologies.

We investigated the components, the communication and the security in MonALISA. We also have set up a test environment to monitor nodes in the LightHouse lab.

First we'll discuss the design of MonALISA, followed by the communication it uses. After that, we write some thoughts on the security of MonALISA next we give some notes on the installation and configuration. The last chapter we give our conclusion of this project including our recommendation for further investigation.

Thank You

We want to thank the people who have helped us during this project.

- Freek Dijkstra & Bas van Oudenaarde & Paola Grosso, for guidance and support during the project
- Iosif Legrand & Adrian Muraru, developers of MonALISA

1 MonALISA Design

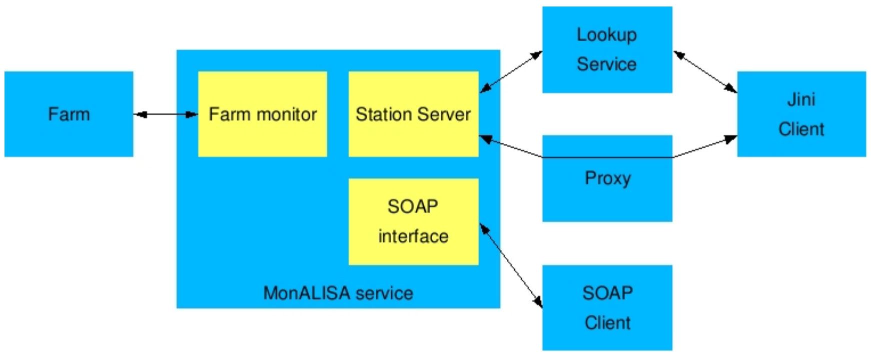

Figure 1: Station Server

As figure 1 shows, MonALISA consists of different kind of services which we'll discuss in the next chapter. First we'll mention the hart of MonALISA, the lookup service. Next we'll discuss the station server and the farm monitor which together will give the information to the clients, mostly these are called together as the MonALISA service [1].

1.1 Lookup Service

As said before, the lookup service is the hart of MonALISA. It controls everything from client to the monitoring information service. Each MonALISA service registers itself with a set of *Lookup Services* as part of a group[1], and having a set of attributes. How the registration is done will be explained in chapter 2.1.1.

MonALISA is developed with reliability in mind. This starts by registering services to a set of lookup services. A lookup service is also a service of MonALISA and therefore it also registers to at least one lookup services. These lookup services will replicate their information to one-other making this a reliable network for registration of services. Besides that, MonALISA is also set up dynamically, it uses a leasing mechanism. How this mechanism works will be explained in chapter 2.1.2.

1.2 Station Server

A station server is a provider for different kind of services. Through this station server information is transfered to the clients. Before this is possible the station server needs to register itself with the lookup service. As a result it downloads the necessary code and parameter data[2] (the yellow dots in the figure), from a JavaSpace. At the same time the station server sets itself up as a remote listener, to keep updated with events. This way each station server keeps an updated dynamic list of active station servers. This communication goes via the proxies. By making use of remote events the station servers will keep up-to-date of changes. The communication is schematically shown in figure 2.

1. see appendix A
2. Information for the communication

The station servers keep a list of other active station servers to create a network that hosts dynamic services. This framework allows services to access the information they require from the entire system and can also interact with other services.

The lookup service is aware of the attributes a specific station server has. According to these attributes a client/service or other service can match these attributes to find the information the client is looking for. After a match has been found the client wishes to communicate with the service. The client downloads a proxy from an URL, which it receives from the lookup service and communicates through it with the station server. How this works will be explained in chapter 1.4.

MonALISA, isn't just a monitoring tool. Because it uses standardized messages and information to communicate with other services it is able to use these messages/information to do something intelligent. For example, if some information is requested by a service, the returned information can be used, if matched by a filter, to take actions. These actions can be used to improve the grid, like load balancing.

The filter mentioned in the previous paragraph, is launched by the *code mobility paradigm*. How code mobility is used is explained in chapter 2.1.7

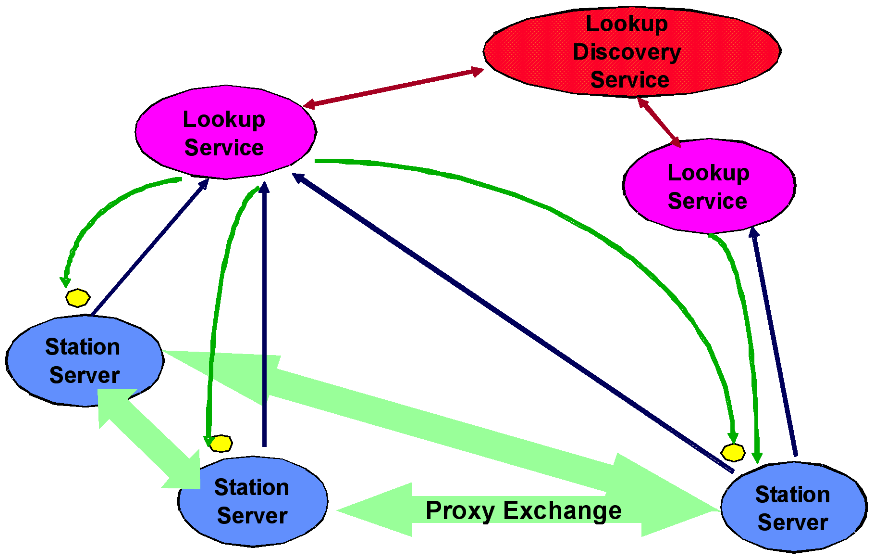

Figure 2: Station Server

1.3 Farm monitor

The data collection part in MonALISA is called the farm monitor, see figure 3. This farm monitor monitors the nodes in a farm [1] [3]. It is possible to use different data collection methods to monitor parameters from the nodes, for example you could use SNMP to monitor parameters from these nodes. The farm monitor can also use an existing monitoring application to monitor parameters from the nodes in a farm.

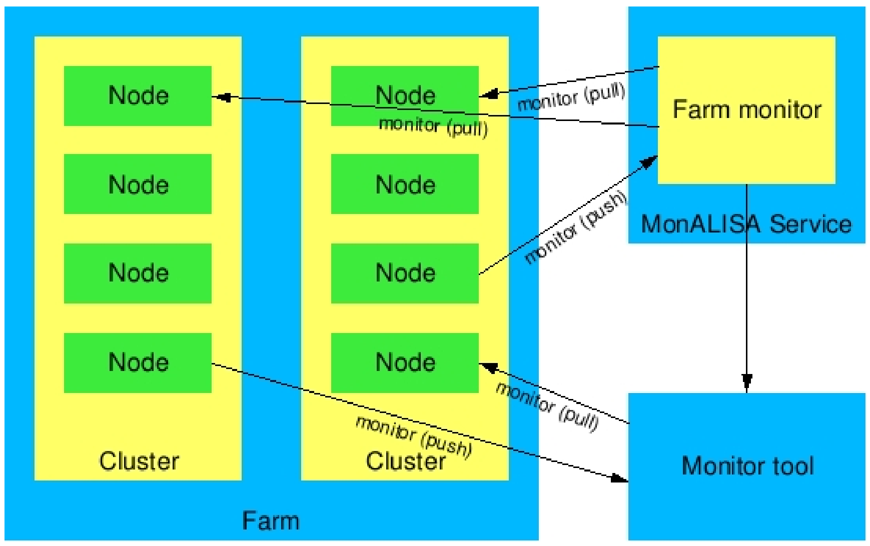

Figure 3: Monitoring parameters in MonALISA.

To be able to monitor the parameters from the different nodes independently and in parallel, the farm monitor is build as a multi-threaded system, consisting of a dedicated control thread and a number of monitor threads (see figure 4). This thread pool is created dynamically. The dedicated control thread is used to control the monitor threads. In this dynamic thread pool, a monitor thread is reused after a monitor task has completed. So the dynamic thread pool only has to be created once, which reduces the load on the system.

To perform the monitor tasks, monitoring modules are used. These monitoring modules can be loaded from a file-system or can be (dynamically) loaded into the farm monitor from a web-server (using HTTP). Each monitoring module can be used to monitor parameters just once or with a given frequency from one or more nodes. In order to monitor, a monitoring module executes a script, runs a program or queries a node using SNMP. It is possible to develop your own modules but there are already monitoring modules developed for:

- querying nodes using SNMP.
- monitoring the local proc.
- use data collected with other monitoring-tools like Ganglia or MRTG (Multi Router Traffic Grapher).

Another possibility is to use ”push methods” (like SNMP traps) to monitor parameters from the nodes. Instead of the farm monitor querying the nodes to monitor the parameters the nodes report the results periodically back to the farm monitor.

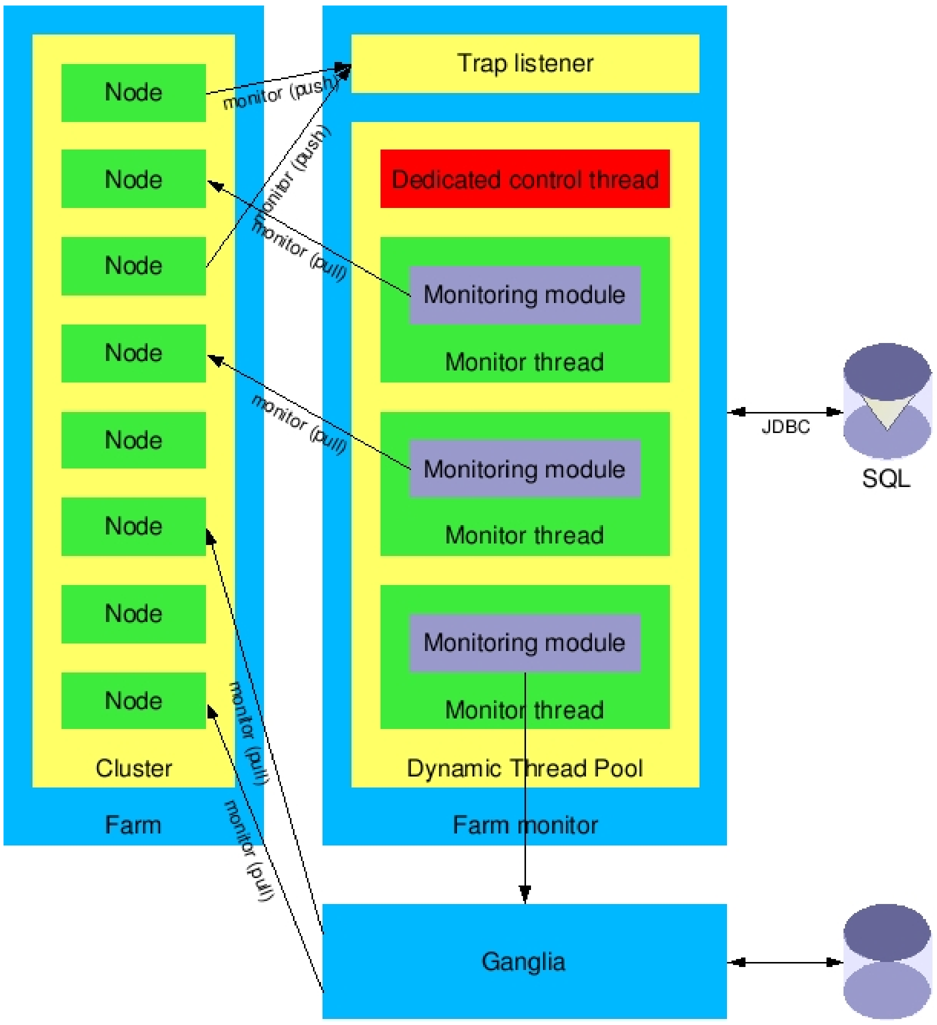

Figure 4: Farm monitor in detail.

When a monitoring task is executed the monitoring module is loaded into the thread. After completion of the task the thread is reused for another monitoring task. The advantage of this multi-threaded system is that if a monitoring task fails or hangs (due to I/O errors) the other monitoring tasks can continue their tasks without being delayed or disrupted. Hanging tasks are properly closed by the dedicated control thread. The dedicated control thread also takes care of rescheduling those tasks that have not been successfully completed.

1.4 Clients

When a client connects to MonALISA, it uses the discovery service to find all the services from a list of user predefined groups. According to this groups a proxy is downloaded via the lookup service that communicates between the client and the specific monitoring service. This monitoring service is the station server and the farm monitor that goes with it.

A client can request for real-time or historical data. It uses a so called predicate and filter mechanism for requesting or subscribing to selected measured values. In case of historical data it uses the predicate mechanism to query the SQL database.

The client also makes use of event notification. By registering itself to the lookup service to specific groups it will receive event updates of these groups.

In chapter 3.2 will be explained how it is possible to administer the MonALISA service via the GUI administration interface.

1.4.1 Pseudo-clients

With the pseudo-client it is possible to have access and gain information through, for example, your mobile phone. It communicates the same way as a normal client, through the lookup service. The pseudo-client also subscribe to specific service with a list of predicates and filters.

We haven't gone in deeper at this subject, because we found it not relevant to this project.

2 Communication

In this chapter we'll describe the communication used in MonALISA. MonALISA is based on a scalable Dynamic Distributed Services Architecture (DDSA) which is designed to meet the needs of physics collaborations for monitoring global Grid systems, and is implemented using Jini/JAVA and WSDL/SOAP technologies [4].

MonALISA services are able to register, discover, receive events and subscribe to each other autonomously by making use of self-describing protocols. MonALISA clients can do the same to the services. With these cooperating services they can access each other seamlessly and adapt rapidly in a dynamic environment. This is possible thanks to Jini (Java Intelligent Network Infrastructure) [5], in the next chapter we will discuss the Jini mechanism.

2.1 Jini

The most important concept within the Jini architecture is that of a service. A service is an entity that can be used by a person, a program, or another service. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user [8].

The services within Jini communicate by using a service protocol, which is a set of interfaces written in Java. This service protocol is RMI, Remote Method Invocation. RMI provides mechanisms to find, activate, and garbage collect object groups.

2.1.1 Lookup Service

The lookup service is the main service in Jini and therefore also in MonALISA. It is the central point of contact between the user and the rest of the system. The main purpose of the lookup service is to find and resolve services. A service is added to a lookup service by a pair of protocols called *discovery and join*. First the service locates an appropriate lookup service (by using the discovery protocol), and then it joins it (by using the join protocol) [15].

The protocols used for discovery can be [6]:

- The multicast request protocol, used to discover one or more lookup services on a local area network (LAN)
- The multicast announcement protocol, used to announce the presence of a lookup service on a local network
- The unicast discovery protocol, used to establish communications over a local area or wide area network (WAN) with a lookup service whose address is known in advance.

In figure 5 is shown how the communication via the lookup service works. The MonALISA service registers itself with one or more lookup services (step 1). At the same time it sets itself up to receive remote notifications. Why this is done will be described in chapter 2.1.3. The lookup services have registered themselves with each other before the MonALISA services do, in step 2 they replicate their information to make MonALISA more reliable.

When a client wants to receive information from MonALISA it uses the lookup discovery service to find all the MonALISA services (step 3). Depending on the client attributes it downloads a proxy from an URL that is specified by the lookup service. The clients connect to the MonALISA service through the proxy (step 4).

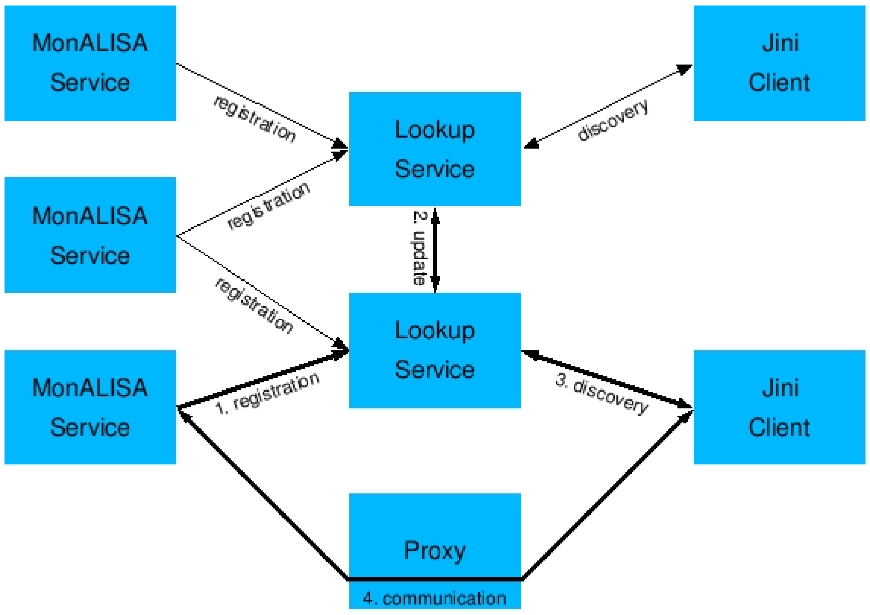

Figure 5: Communication in MonALISA

2.1.2 Leasing Mechanism

A lease is a grant of guaranteed access over a period of time. The lease is negotiated between the user of the service and the provider of the service. If the lease isn't renewed before it expires, the resource can be freed. This can happen in case of client or network failure, the service is no longer needed or the lease is not permitted to be renewed.

Jini has taken thought about the leasing mechanism. Because of working in an distributed system, with clients and servers all over the world, you can encounter problems that you won't have on a single computer. For example, the time. The time will never be exact the same on each computer. Therefore, the lease mechanism works with a countdown system instead of specifying a timestamp in the future.

2.1.3 Remote events

Within Jini it is possible to get notified if changes occur. This can be a service that is unavailable or if a network link is down. These changes are called events. If a service or a client register interest in events, they will receive notification of the occurrence of such an event.

2.1.4 Transactions Manager

The transaction manager doesn't really do more than any other transaction manager. The transaction manager makes sure that the operation(s) are performed correct. Only if the operation is performed correct the result is written.

More information about the transaction manager specified in Jini can be viewed at the Jini homepage [9].

2.1.5 JavaSpace Service

This service supports an ensemble of active programs, distributed over a set of physically dispersed machines. While each program is able to execute independently of the others, they all communicate with each other by releasing data (a tuple) into tuple spaces containing code as well as data. Programs read, write, and take tuples (entries) from tuple spaces that are of interest to them [2].

This JavaSpace can be seen as a cloud around the services of Jini. The other services take care of the communication like the discovery and join, and the remote events. The advantage that is created by this mechanism is the presentation to the clients. To the clients it looks like MonALISA is only one system.

The JavaSpace communication mechanism was heavily influenced by the concept of a tuple space that was first described in 1982 in a programming language called Linda [10]. A similar implementation from IBM that can be also used in the Jini architecture is named Tspaces [11]. An extended, high performance implementation which also supports replication is provided by Gigaspaces [12].

2.1.6 The Mailbox Service

This service can be used to provide asynchronous communications (based on any type of messages) between distributed services. [2]

2.1.7 Code Mobility

Agents or Filters are java objects (code) that a client (or an other service) can deploy on one or all the services. These pieces of code can process data locally at each service and send back to the client processed information or are used to take action when predefined condition are detected. An agent or filter can communicate back with the client (or service) that deployed them in different ways (RMI, client server, channels). For security, these objects that are dynamically deployed are digitally signed.

Code Mobility is pretty vague. There is not very much documentation on it and time was not at our hand to do more research. For more information about Code Mobility, see [13] [14].

2.2 Soap

Beside the Jini interface MonALISA also has a SOAP interface. SOAP (Simple Object Access Protocol) is a simple XML based protocol to exchange information between applications in a computer understandable format. With the MonALISA SOAP interface you can retrieve the values of the monitored parameters and the monitoring configuration from the farm(s) the MonALISA service monitors.

SOAP is one of the protocols used in the so-called Web Services.

> The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services [7].

The MonALISA SOAP interface can be used to build higher level services which require this monitored parameters or the monitoring configuration from the farm(s).

There are three SOAP methods available to get information from the monitored farm(s).

getValues This method is used to get the monitored parameters in a specific time interval. To get this parameters you should specify in your request:

- the farm name (* means all monitored farms).
- the cluster name (* stands for all monitored clusters).
- the node name (again * means all monitored nodes).
- the parameter name (* returns all monitored parameters).
- the time interval, this can be specified using absolute values (in ms since 1970) or relative values (in ms).

The method returns an array of the requested monitored parameters, specifying for every monitored parameter the absolute time (ms since 1970) when this value was registered.

getConfiguration Get the farm configuration(s) in a specific time interval. In your request you should specify the time interval, using absolute time values (in ms since 1970).

getLastConfiguration Use this method to get the latest configuration from a farm. In your request you only have to specify the farm name you want to receive the configuration from (or * to see the configuration from all monitored farms).

To be able to communicate with this SOAP interface it is required to know the interface definition. WSDL (Web Service Description Language) is a simple XML based protocol used to specify the SOAP interface in a computer understandable format. It is not necessary to hardcode the interface definition in your program because you can access the WSDL (and thus the SOAP interface) during runtime.

By default the MonALISA WSDL is published on `http://<your_hostname>:6004/axis/services/MLWebService?wsdl` and the SOAP interface at `http://<your_hostname>:6004/axis/services/MLWebService`.

Chapter 4.2.3 explains how to start the MonALISA Web Service. The online manual [16] explains the Web Service interface in a more detailed manner.

3 Security

In this chapter we'll discuss the security and perhaps the insecurity of MonALISA. First we'll discuss differences between the MonALISA versions, following the Jini security. After that the administration interface will be discussed and finally the group registration.

3.1 MonALISA versions

After some communication with the developers of MonALISA we heard about MonALISA version 2. We haven't found any documentation about it, so we were surprised to hear this. The question that lead to this information was about the security of MonALISA version 1. After reading information about MonALISA version 1 and about Jini, we were interested if MonALISA was using the *net.jini.security* package.

About the security in MonALISA version 1, according to the developers:

> There are no security issues in Monalisa v1 as much as we know

We took this for granted because we didn't have the time to do a research about its security. Though we have read about the package net.jini.security. This package check the confidentiality and the integrity of, for example, the downloaded code. If you think of it, wouldn't that declare MonALISA version 1, insecure. . .

3.1.1 Jini & Security

We won't go in to deep on this subject and we can't say what the perfect way should be for Jini/MonALISA to set up their environment. Nevertheless, we can mention somethings that could look insecure.

When a client communicates to MonALISA, after the discovery procedure, it communicates through the downloaded proxy. It uses the X.509 certificates to authenticate the client and the server. Unfortunately, this doesn't determine if the proxy is trustworthy. The client doesn't want to give its private key to the "wrong" proxy.

Downloaded code is always a security problem, because you don't want malicious code to be run on your system. Even if MonALISA version 1 is secure, are the permissions of the downloaded code correct and does this changes in case someone will use the administration interface (chapter 3.2).

After considering this, are remote events transmitted secure? Is the integrity of these events guaranteed? Should the leasing mechanism be secure? Should the same be considered for the transaction mechanism?

3.2 Administration interface

The MonALISA GUI has an administration interface which you can use to administer the MonALISA service. Administering is done via a SSL connection which is based on X.509 certificates. The server (the MonALISA service) has a public certificate from every user who is authorized to administer the MonALISA service. If the user can prove knowledge of the corresponding private certificate the user is allowed to administer the MonALISA service. The certificates are stored in the so-called ”keystore”. One keystore (with the public certificates) is stored on the server, the other keystore (containing the public and the private certificate(s)) is stored on the client, see figure 6. Chapter 4.2.4 explains in detail how to create certificates and how to store them in the keystore.

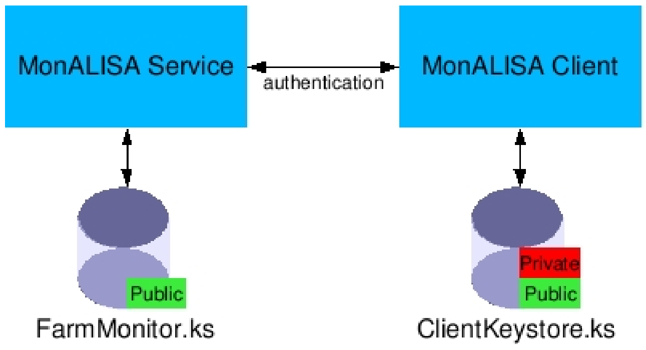

Figure 6: MonALISA certificates and keystores.

3.3 Group registration

When you register yourself with a lookup service you have to register yourself as part of a group. It is not possible to register yourself in any group you like. There are freely available groups (like ’test’) and there are secured groups. Registration in these secured groups is only possible using X.509 certificates. This X.509 certificate has to be signed by a trustworthy Certification Authority.

4 Installation and configuration

4.1 Installation

The online manual [16] explains how to install MonALISA. Just *read* the preface, chapter 1.1 and chapter 1.2 and *follow* the steps described in chapter 1.3 (only read chapter 1.3.1).

To find the Latitude and Longitude of the server use the Maporama site [17] instead of the suggested site Geotags [18].

4.2 Configuration

The online manual [16] explains how to configure MonALISA. The basic configuration is done with the install script used to install MonALISA. MonALISA uses three configuration files to configure its environment. The first configuration file is used to specify how to start MonALISA.

ml_env In this file settings like your Java path and the username from which you will run MonALISA are specified. The file is located at *$MonaLisa_HOME/ Service/CMD/*

The next two configuration files are used by MonALISA itself and are located at *$MonaLisa_HOME/Service/YOURFARMNAME/*

ml.properties The global farm properties are specified in this file. Settings like your farm name, your farm location, which monitoring modules should be loaded and which database should be used.

YOURFARMNAME.conf Settings like cluster names, node names and the monitoring modules which should be used to monitor a node are specified in this configuration file.

The default for *YOURFARMNAME* is *myFarm*.

4.2.1 MonALISA group

The MonALISA group in which you want to register is specified in the *ml.properties* file. As explained in chapter 3.3 you can't register in any group you like. By default your farm is registered in the (freely available) *test* group. With the line

```
lia.Monitor.group=test
```

you can change the group you want to register with. You can only register in groups that already exist. At this moment only the developers are able to start a new group.

4.2.2 Farm monitoring configuration

The parameters from your farm that should be monitored are initially configured in the farm monitor configuration file *YOURFARMNAME.conf*. When MonALISA is running, you can change this configuration file and restart MonALISA, but you can also dynamically change the farm monitor configuration using the administration interface.

As explained in appendix A.2 a farm consists of one or more clusters and on their turn consists of one or more nodes. In the farm monitor configuration file (*YOURFARMNAME.conf*) a new cluster is defined with the line:

```
*cluster_name
```

Some special cluster names are used to be able to collect information shown on the map in the MonALISA GUI.

WAN The monitored parameters from the nodes in the WAN cluster are used to show the information for the WAN links on the map. To be able to show the WAN links on the map, the MonALISA developers have to know the endpoints of the WAN links before they appear on the map. Example
```
*WAN
```

PN PN stands for 'Processing Nodes' and the monitored parameters from the node(s) in this cluster are used to show the pie chart views containing the cluster usage on the map. To be able to show this information on the map the cluster name has to start with PN. Example
```
*PN_yourname
```

To define a monitored node in this cluster you should specify the complete node name (or IP address) optionally following on the nodes short name.

```
>node_name.domain.com
```

or

```
>short_name node_name.domain.com
```

Monitoring modules are used to collect the data (see chapter 1.3). Several modules are shipped with MonALISA to collect data for example using SNMP or the local proc. Usually monitoring modules collect more than just one parameter from a node. Which monitoring modules should be used to monitor a node with a given frequency is specified with the line:

```
module_name%30
```

The monitoring frequency (*%30*) is specified in seconds. It is possible to use more than one monitoring module to collect parameters from a node.

A complete farm monitor configuration file looks like this

```
*PN_cluster
>node0 node0.domain.com
monProcLoad%30
monProcStat%30
monProcIO%30

>node1 node1.domain.com
monPing%30
SSHProcLoad%30

*Routers
>router0 router0.domain.com
monPing%30
```

MonALISA ships with some monitoring modules. Some of these supplied monitoring modules are:

monProc* The monProc* modules (monProcLoad, monProcIO, monProcStat) collect parameters from the local system (the system running MonALISA) using the local proc. It collects parameters like load average, I/O on network interface(s) and CPU load.

monPing This module sends a ping from the system running MonALISA to the specified node and collects parameters like round trip time and lost packages.

snmp_Load The snmp_* modules collect parameters from a node using SNMP messages. This module (snmp_Load) is used to monitor the load average from a node.

snmp_IO Monitors the I/O on the network interface(s).

snmp_CPU Monitors the CPU usage from a node.

snmp_Mem Memory usage can be monitored using this module.

snmp_Disk A module to monitor disk usage from a node.

snmp_IOpp The snmp_IOpp and the snmp_IOpp_HC modules can be used to monitor router and switch interfaces using SNMP. The snmp_IOpp uses 32-bit SNMP counters, the snmp_IOpp_HC uses 64-bit SNMP counters. The configuration (in *YOURFARMNAME.conf*) for this module is:

```
snmp_IOpp{1=if1_desc;2=if2_desc;3=if3_desc}%30
```

> WARNING
> At the moment of writing the snmp_IOpp module has some known problems with the parsing of these strings. Therefore there should be no space after the `;`.

1 is the SNMP interface number, *if1_desc* is your own interface description. To discover the SNMP interface number you can use the *snmpwalk* utility.

```
snmpwalk -v2c -c public <routerIP> .1.3.6.1.2.1.2.2.1.2
```

This will lists all the interface and their numbers.

```
IF-MIB::ifDescr.1 = STRING: GigabitEthernet 0/0
IF-MIB::ifDescr.2 = STRING: GigabitEthernet 0/1
```

The MonALISA user guide [16] describes SNMP in a more detailed manner.

snmp_IOpp_HC This module is the 64-bit equivalent of the snmp_IOpp module. The configuration for this module is the same:

```
snmp_IOpp_HC{1=if1_desc;2=if2_desc}%30
```

> WARNING
> At the moment of writing the snmp_IOpp_HC module has some known problems with the parsing of these strings. Therefore there should be no space after the `;`.

It is also possible to write your own modules. The modules are written in Java, examples are located at *$MonaLisa_HOME/Service/usr_code/*. During our project we have written our own module to collect parameters from a nodes proc using SSH. See appendix B for the source code. To use your own module you have to specify the class URL from your module(s) in the *ml.properties* file, so MonALISA can load the modules.

```
lia.Monitor.CLASSURLs=file:${MonaLisa_HOME}/Service/
usr_code/SSHProc/
```

You can now use your modules in the farm monitor configuration file (*YOURFARMNAME.conf*). For example, to use your own module *SSHProcLoad.class* use the line

```
SSHProcLoad%30
```

4.2.3 Web Service configuration

The settings for the MonALISA Web Service are configured in the *ml.properties* file. To start the Web Service you should change the line

```
lia.Monitor.startWSDL=false
```

into

```
lia.Monitor.startWSDL=true
```

You can configure the port on which the Web Service is running with the line

```
lia.Monitor.wsdl_port=6004
```

By default MonALISA uses port 6004.

After a restart MonALISA offers a SOAP interface on `http://<your_hostname>:6004/axis/services/MLWebService` and its WSDL on `http://<your_hostname>:6004/axis/services/MLWebService?wsdl`.

4.2.4 Basic certificate management

Administering the MonALISA service can also be done using the MonALISA GUI (see chapter 3.2) [19]. Authenticating users is done with SSL (X.509) certificates. The certificates are stored in the so-called "keystore". One keystore (with the public certificates) is stored on the server, the other keystore (containing the public and the private certificate(s)) is stored on the client see figure 7. MonALISA uses the *Java keytool* utility to generate and store X.509 certificates. MonALISA provides some scripts to ease the use of the *Java keytool*.

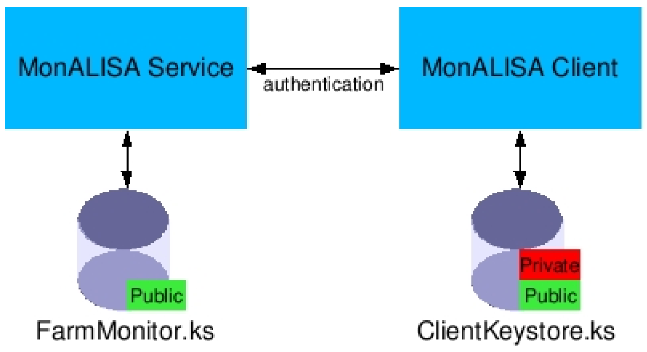

Figure 7: MonALISA certificates and keystores.

The *Java keytool* is located at *$JAVA_HOME/bin/keytool*, the MonALISA scripts are located at *$MonaLisa_HOME/Service/SSecurity/*.

To set up certificates based authentication for administering the MonALISA service you have to follow four steps:

1. Create the client keystore with a public and private certificate.
2. Export the client public certificate.
3. Import the client public certificate into the servers keystore.
4. Restart the MonALISA service.

The first step uses the *genKey* script to create a new X.509 certificate pair (a private and a public certificate) and to store this new certificate pair into a keystore. This keystore, containing a certificate pair (public and private certificate), is used by the client and thus should be stored there (ClientKeystore.ks in figure 7). The keystore and the certificate(s) are password protected. This *genKey* script takes two arguments:

1. The name of the keystore. You can add the new certificate into an existing keystore or store the certificate into a new keystore. If the keystore doesn't exist it will be created.
2. An alias name for the certificate.

When you run the script *genKey* you will be asked to enter the keystore password and some personal details for creating the X.509 certificate.

> WARNING
> Choose the same password to protect your certificate and to protect your keystore, because the MonALISA GUI isn't able to handle different passwords for them.

```
$ $MonaLisa_HOME/Service/SSecurity/genKey \
> newkeystore.ks certaliasname

Enter keystore password:  ******
What is your first and last name?
  [Unknown]:  M.Y. Name
What is the name of your organizational unit?
  [Unknown]:  Unit
What is the name of your organization?
  [Unknown]:  Organization
What is the name of your City or Locality?
  [Unknown]:  City
What is the name of your State or Province?
  [Unknown]:  State
What is the two-letter country code for this unit?
  [Unknown]:  CC
Is CN=M.Y. Name, OU=Unit, O=Organization, L=City, ST=State,
C=CC correct?
  [no]:  yes

Enter key password for <certaliasname>
        (RETURN if same as keystore password):  ******
```

This keystore with the public and the private certificate should be stored *on the client*.

Another keystore is located at the server (FarmMonitor.ks in figure 7), this keystore doesn't contain certificate pairs but only contains public certificates from the users which are authorised to administer the MonALISA service. This keystore is located at *$MonaLisa_HOME/Service/SSecurity/FarmMonitor.ks*. The second step is to get the users public certificate so you can import this certificate into the (servers) keystore. The user has to export his public certificate out of his keystore (ClientKeystore.ks in figure 7). You can use the supplied *exportCert* script to store the users public certificate into a file. This script takes three arguments:

1. The keystore in which the public certificate is stored.
2. The alias name for the certificate that should be exported.
3. Into which file the (public) certificate should be stored.

```
$ $MonaLisa_HOME/Service/SSecurity/exportCert \
> newkeystore.ks certaliasname public.cert

Enter keystore password:  ******
Certificate stored in file <public.cert>
```

Importing this public certificate into the servers keystore (FarmMonitor.ks in figure 7) is the third step. To do so you use the supplied *importCert* script. This script takes two arguments:

1. An alias name for the public certificate.
2. The location of the public certificate.

If you trust this (public) certificate the script imports the public certificate into the keystore (FarmMonitor.ks).

```
$ $MonaLisa_HOME/Service/SSecurity/importCert \
> publiccertaliasname public.cert

Owner: CN=M.Y. Name, OU=Unit, O=Organization, L=City,
 ST=State, C=CC
Issuer: CN=M.Y. Name, OU=Unit, O=Organization, L=City,
 ST=State, C=CC
Serial number: 41efb162
Valid from: Thu Jan 20 14:25:54 CET 2005 until: Wed Apr 20
 15:25:54 CEST 2005
Certificate fingerprints:
 MD5:  39:54:2D:A1:8F:C2:23:13:64:F7:68:DF:78:7E:E3:68
 SHA1: AE:08:98:7D:04:BE:8F:C3:68:52:A8:FE:D2:2D:82:6E:
  2C:19:5C:69
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

The last step is to restart the MonALISA service.

To administer the MonALISA service the user has to load his keystore (containing the public and the private certificate) into the MonALISA GUI.

1. Choose *Keystore* from the menu *security*.
2. Select the location of the keystore and enter the keystore password.
3. Click *OK*.

If you now display the farm properties window from a farm you are authorised to administer, the administration buttons will appear.

> NOTE
> It could take some time before the buttons appear.

4.2.5 Advanced certificate management

Beside creating, exporting and importing certificates into a keystore it is also possible to list the stored certificates or to delete a certificate from a keystore. To create, export and import a certificate you can use the supplied scripts located at *$MonaLisa_HOME/Service/SSecurity/*, other certificate management tasks can be accomplished with the *Java keytool* located at *$JAVA_HOME/bin/keytool*. The servers keystore is located at *$MonaLisa_HOME/Service/SSecurity/FarmMonitor.ks*.

To list all the certificates in the servers keystore use the command

```
$ keytool -list -keystore FarmMonitor.ks
```

The keystore is password protected, the default password is "monalisa".

The command

```
$ keytool -delete -alias aliasname -keystore FarmMonitor.ks
```

deletes the certificate with the alias name "aliasname" from the servers keystore.

> NOTE
> It is required to have at least one private key in the servers keystore (FarmMonitor.ks) to be able to set up SSL communications.

5 Conclusions

We hope it is clear that the MonALISA service is more than just monitoring software. MonALISA truly is a monitoring *service*. You can build higher level services upon this MonALISA service using the Jini or the SOAP interface.

Within Lambda context the SOAP interface can be useful in retrieving monitoring information used in some higher level (lambda) service. Because the used format (SOAP) is computer understandable the higher level (lambda) service can adapt itself dynamically when changes occur in which nodes are being monitored and how they are monitored.

The Jini interface can also be used to accomplish this task, but we suppose that the Jini interface can do even more. We suppose that you are able to use the Jini interface to dynamically change which nodes are being monitored. Our supposition comes from the fact that you can use the administration interface in the MonALISA GUI to dynamically change which nodes are being monitored. We suppose the MonALISA GUI uses Jini to accomplish this task.

Being able to dynamically change which nodes are being monitored can be useful for example when a higher level services changes the underlying structure. The higher level service is than also able to change which nodes are being monitored and how they are monitored. This may be subject of further investigation.

Another point of further investigation are the (security) differences between MonALISA version 1 and 2.

Our opinion

We think the MonALISA service is a good an innovative concept. We have enjoyed researching MonALISA although in our opinion the documentation on MonALISA could be expanded. We hope our report contributes to this. We also hope this report can help the society also working on this project.

References

[1] *MonALISA Design*
http://monalisa.caltech.edu/design.html

[2] *Data Intensive Grids for High Energy Physics*
http://ultralight.caltech.edu/gaeweb/Grid2002_HEPChapterFinal_071902.doc

[3] *MonALISA Design.*
http://www.sinica.edu.tw/~jzlee/LCG/Design/

[4] *MonALISA: A Distributed Monitoring Service Architecture*
http://monalisa.caltech.edu/documentation/MOET001.pdf

[5] *Jini homepage*
http://www.jini.org

[6] *Discovery and Join*
http://java.sun.com/products/jini/2.0/doc/specs/html/discovery-spec.html

[7] *Web Services Activity.*
http://www.w3.org/2002/ws/

[8] *Jini Specifications Archive - v2.0*
http://java.sun.com/products/jini/2_0index.html

[9] *Transaction*
http://java.sun.com/products/jini/2.0/doc/specs/html/txn-spec.html

[10] *Maximum Performance Through Parallel Execution*
http://lindaspaces.com/products/linda_overview.html

[11] *What is TSpaces?*
http://www.alphaworks.ibm.com/tech/tspaces/

[12] *Gigaspaces Enterprise Application Grid*
http://www.gigaspaces.com/

[13] *Understanding Code Mobility*
http://sunset.usc.edu/classes/cs599_2002/Week11_c.ppt

[14] *Understanding Code Mobility*
http://www.elet.polimi.it/upload/picco/papers/icse00tut.pdf

[15] *JINI Lookup service*
http://java.sun.com/products/jini/2.0/doc/specs/html/lookup-spec.html

[16] *MonALISA - Service User Guide.*
http://monalisa.caltech.edu/documentation/ml_ser_ug.html

[17] *Maporama*
http://www.maporama.com

[18] *Geotags*
http://geotags.com/

[19] *MonALISA User Guide*
http://www.sinica.edu.tw/~jzlee/LCG/Documentation/userguide.html

[20] *GRID3*
http://www.ivdgl.org/grid2003/

A Definitions

A.1 Groups

MonALISA is diveded into several groups (see figure 8), every group contains one or more farms. Mostly the groupname indicates to which organisation[3] the farms belong. A farm can also span several groups. An example is the ”grid3” group [20]:

> The Grid3 collaboration has deployed an international Data Grid with dozens of sites and thousands of processors. The facility is operated jointly by the U.S. Grid projects iVDGL, GriPhyN and PPDG, and the U.S. participants in the LHC experiments ATLAS and CMS.

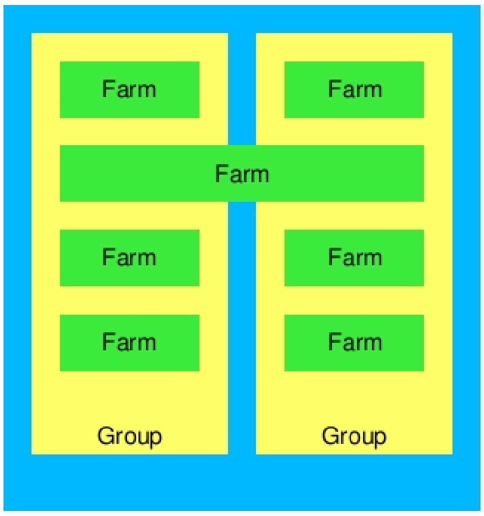

Figure 8: The MonALISA ’world’.

A.2 Farm, cluster and node

A node in MonALISA is a ”piece” of hardware that *can* contain one or more processors, for example a computer or a router. A clusters is a collection of nodes and a farm contains one or more clusters. See figure 9.

A.3 MonALISA service

To include your farm into the MonALISA world you have to install software on one of your nodes (this node can be part of the farm or can be a dedicated machine). This piece of software takes care of the communication between your farm and the MonALISA world and takes care of monitoring the parameters from your nodes. We call the node running this piece of software the MonALISA service.

3. When two or more organisations collaborate this groupname indicates to which virtual organisation the farms belong.

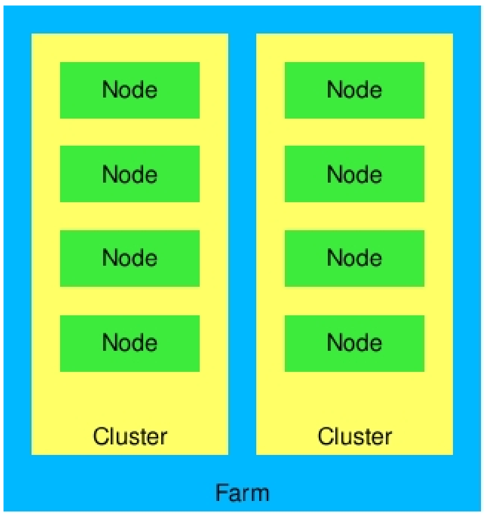

Figure 9: A farm in MonALISA.

B SSHProcLoad

This is the source code for the monitoring module using SSH to monitor the load average from a node. The module uses the load average information from */proc/loadavg*. It is required to set up password less, key based SSH login to the monitored node.

```
import lia.Monitor.monitor.*;

import java.io.*;
import java.util.*;
import java.net.InetAddress;


public class SSHProcLoad extends cmdExec implements  MonitoringModule {

  // The name of this module
  static public String ModuleName="SSHProcLoad";
  // What parameters do we monitor?
  static public String[]  ProcResTypes = {"Load5", "Load10", "Load15" };

  static public String OsName = "linux";
  double load5, load10, load15;
  String user = "user";


  public SSHProcLoad () {
    super( ModuleName);
    info.ResTypes = ProcResTypes;
    System.out.println ( "Start the Interface to SSH proc." );
    isRepetitive = true;
  }


  public String[] ResTypes () {
    return ProcResTypes;
  }

  public String getOsName() { return OsName; }

  public MonModuleInfo getInfo(){
    return info;
  }


  //Get and process the results
  public Object   doProcess() throws Exception {
    System.out.println ("Connecting to host " + Node + " as " + user + ".");

    //Command to read remote proc using ssh:
    // ssh -n -2 -x -t -C  user@node ’ cat /proc/loadavg’
    String cmd = "ssh -n -2 -x -t -C " + user + "@" +
                              Node + " ' cat /proc/loadavg'";

    //Get the results
    BufferedReader buff1 = procOutput ( cmd );

    if ( buff1  == null ) {
      System.out.println ( " Failed  for " + full_cmd );
      throw new Exception ( " Proc output is null for " + Node.name);
    }

    //Process the results
    String lin;
    StringTokenizer tz;

    try {
      for ( ; ; ) {
        lin = buff1.readLine();

        if ( lin == null ) break;
        if ( lin.equals("") ) break;

        tz = new StringTokenizer ( lin );
        load5 = (new Double(tz.nextToken().trim())).doubleValue();
        load10 = (new Double(tz.nextToken().trim())).doubleValue();
        load15 = (new Double(tz.nextToken().trim())).doubleValue();
      }
      buff1.close();
    } catch ( Exception e ) {
      System.out.println ( "Exception in Parsing SSH proc output  Ex=" + e );
      throw e;
    }

    //Store the results
    Result result  = new Result ( Node.getFarmName(),Node.getClusterName(),
                                 Node.getName(), ModuleName, ProcResTypes );
    result.time =  ( new Date()).getTime();
    result.param[0] = load5; //load5
    result.param[1] = load10; //load10
    result.param[2] = load15; //load15

    System.out.println ( " Result = " + result );
    return result;
  }

  static public void main ( String [] args ) {
    String host = args[0];
    SSHProcLoad aa = new SSHProcLoad();

    String ad = null ;
    try {
      ad = InetAddress.getByName( host ).getHostAddress();
    } catch ( Exception e ) {
      System.out.println ( " Can not get ip for node " + e );
      System.exit(-1);
    }

    MonModuleInfo info = aa.init( new MNode (args[0] ,ad,  null, null),
                                         null, null);
    try  {
      Object bb = aa.doProcess();
    } catch ( Exception e ) {
      System.out.println ( " failed to process " );
    }
  }

}
```