REMBASSY: OPEN SOURCE TOOL FOR NETWORK MONITORING

Vreixo Formoso, Fidel Cacheda, Víctor Carneiro, Juan Valiño
Área de Ingeniería Telemática, Dpto. de Tecnologías de la Información y las Comunicaciones
Universidad de A Coruña, Fac. de Informática, Campus de Elviña s/n
15.071 – A Coruña. Spain

ABSTRACT
Monitoring systems are an essential element in the management of information systems. However, in the context of Open Source, existent monitoring systems have not yet obtained the necessary level of maturity and professionalism. In this paper, the limitations of the current tools are analyzed, and a new architecture is developed, which in our opinion resolve the principal existing problems. The system which is presented incorporates important improvements, such as a centralized configuration via web, support for monitoring templates, a hierarchical structure of objects in which its design or flexibility is based, and support for centralized and distributed monitoring schemes. Its extension system, based on plug-ins, is especially innovative for its power and simplicity. Finally, this article studies its use in an authentic environment, which makes it possible to verify the importance of the improvements that have been developed.

KEYWORDS
Computer networks, Monitoring tools, Network management.

1. INTRODUCTION
Nowadays, monitoring tools have a fundamental importance by being essential for the maintenance and management of the organizations’ information systems. These tools monitor the state of the diverse components which make up an information system by notifying the user of distinct problems and incidences. This monitoring can be divided into two main groups, depending on if the system analysis is done to locate future problems (proactive) or if it is limited to locating existent problems (reactive).

In spite of the fact that quite a large number of current information systems are based on free applications, it is undeniable that there still exists a void within the Open Source framework as far as monitoring tools are concerned. In the majority of cases, the available applications supply a much-reduced set of functionalities, are neither very flexible nor easy to extend, and often have an excessively complex configuration.

In this paper, a new monitoring tool, which we have named rembassy [1], is proposed. This tool implies an important improvement in various aspects compared with the ones currently available.

In the first place, the use of the system has been simplified for the user. The complex configuration based on text files, which is used in the majority of existing applications, has been substituted by an intuitive Web interface. In addition, rembassy is extremely flexible, permitting both centralized and distributed monitoring schemes, or using agents for monitoring parameters not accessible via the network.

Important improvements have also been incorporated which make rembassy a system that is scalable to large networks, such as the hierarchical structure of objects, the monitoring templates at various levels, or its distributed monitoring characteristics. Its plug-ins system, which facilitates the extension of the system, is notably superior to existing tools.

Finally, it should be pointed out that a robust architecture has been developed, thanks to the use of numerous design patterns, the implementation of a large number of unity and acceptation tests, or the development methodology based on extreme programming [2].

This article is structured in the following manner: in the first place the most important characteristics of the existent Open Source monitoring systems are analyzed, with their major shortcomings being identified.
After that, the architecture proposed to resolve these problems is described; in section 4, a real study case is presented. Finally, the conclusions arrived at are described and the planned future lines of research are presented.

2. STATE OF THE ART

Currently, there are quite a few Open Source monitoring applications, which is not strange considering the important presence of free applications in the information systems.

In order to obtain a global vision of the state of the art in this dominion, various tools have been studied, among which we have pointed out the 10 most popular / complex. In Table 1, a list of them with their principal characteristics can be consulted.

Our study has consisted of the analysis of a series of common characteristics and functionalities in the monitoring systems, studying the tendency followed in the Open Source tools:

- Type of monitoring interfaces: the web interface is undoubtedly the most popular option among the applications studied to show information to the user. What is lacking, however, is that the majority of the tools do not also permit the configuration of the system from this interface. That option, without a doubt, would be enormously useful for the users.

- It is also interesting to distinguish between centralized and distributed monitoring interfaces. The centralized ones permit the monitoring of all the network hosts from only one point, while the distributed ones require the user to connect to each host in order to obtain the monitored values. As can be expected, the centralized ones are by far the most utilized because they are much more practical for the users.

- Type of inquiry: it can be either centralized, meaning that one group is responsible for consulting all of the monitoring parameters, or distributed, meaning that the monitoring responsibility is divided among various groups. Most of the tools which were studied do not offer the possibility of a choice to the user, and they simply implement one scheme or the other.

- Type of configuration interface: The majority of the systems analyzed use a configuration based on text files. This obligates the user to be familiar with the format of these files, which can be complex in certain cases. As was previously mentioned, a web interface would be a much more suitable option. Another problem is that those systems using distributed inquiry require the configuration to be distributed, which is extremely tedious. It would be interesting if the tools which use this type of inquiry could permit a centralized configuration.

Table 1. Open Source Monitoring Tools. Legend: C: centralized; D: Distributed

<table>
<thead>
<tr>
<th>Tool</th>
<th>Monitoring interface</th>
<th>Inquiry</th>
<th>Configuration interface</th>
<th>SNMP support</th>
<th>extensible?</th>
<th>historical?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nagios [7]</td>
<td>C Web/Wap</td>
<td>C Text</td>
<td>C Text</td>
<td>√, √</td>
<td>√, √</td>
<td></td>
</tr>
<tr>
<td>OpenNMS [8]</td>
<td>C Web, C Text</td>
<td>C Text</td>
<td>C Text</td>
<td>√, √</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pandora FMS [9]</td>
<td>C Web, C Text</td>
<td>C Text</td>
<td>C Text</td>
<td>√, √</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After that, the architecture proposed to resolve these problems is described; in section 4, a real study case is presented. Finally, the conclusions arrived at are described and the planned future lines of research are presented.

2. STATE OF THE ART

Currently, there are quite a few Open Source monitoring applications, which is not strange considering the important presence of free applications in the information systems.

In order to obtain a global vision of the state of the art in this dominion, various tools have been studied, among which we have pointed out the 10 most popular / complex. In Table 1, a list of them with their principal characteristics can be consulted.

Our study has consisted of the analysis of a series of common characteristics and functionalities in the monitoring systems, studying the tendency followed in the Open Source tools:

- Type of monitoring interfaces: the web interface is undoubtedly the most popular option among the applications studied to show information to the user. What is lacking, however, is that the majority of the tools do not also permit the configuration of the system from this interface. That option, without a doubt, would be enormously useful for the users.

- Type of inquiry: it can be either centralized, meaning that one group is responsible for consulting all of the monitoring parameters, or distributed, meaning that the monitoring responsibility is divided among various groups. Most of the tools which were studied do not offer the possibility of a choice to the user, and they simply implement one scheme or the other.

- Type of configuration interface: The majority of the systems analyzed use a configuration based on text files. This obligates the user to be familiar with the format of these files, which can be complex in certain cases. As was previously mentioned, a web interface would be a much more suitable option. Another problem is that those systems using distributed inquiry require the configuration to be distributed, which is extremely tedious. It would be interesting if the tools which use this type of inquiry could permit a centralized configuration.

- SNMP (Simple Network Management Protocol) support: SNMP is a standard for the monitoring of network devices [13]. Due to its limitations in the representation of the managed information, and its security problems, especially in the 1 and 2c versions, most of the applications are not based on this standard. Instead, they define their own protocols to obtain, transmit, and store the monitoring data. However, given that it is
still widely used in network monitoring, some applications permit the interaction with SNMP, generally by means of a specific plug-in. In most cases, this support is limited to the reception of SNMP traps. Among the analyzed tools, only OpenNMS is actually a full-blown SNMP management application.

- What also should be taken into account are the parameters the tool permits to monitor. The immense majority support the monitoring of TCP services. In addition, those based on agents permit the monitoring of many other parameters which cannot be obtained by means of the network, such as the use of CPU, memory…

However, the majority are extensible, which permit the users to develop new modules or plug-ins which will be integrated into the application. It is for that reason that very often the number of supported parameters is not as important as their extensibility.

- Historical storage. It is usual that many applications only permit having awareness of the state of the systems at the moment of the consultation, therefore lacking the possibility of storing historical registers of the monitored parameters.

- Supported platforms: The majority of the applications studied are intended for Unix systems, which is not strange considering the tight relationship between GNU/Linux and the Open Source movement, and they cannot be executed on other platforms.

Nowadays, the most popular tool is undoubtedly Nagios. It is a very flexible tool, and it supports a large number of monitoring elements. However, its configuration is extremely complex because it is based on text files. For the user, it would be much simpler to be able to configure the tool through the web interface. In addition, its design based on structured programming and the shell scripts make its maintenance and extension difficult.

3. ARCHITECTURE

In the previous section, some of the existent main problems currently present in Open Source monitoring applications were looked at.

The architecture proposed in this work, which is described in this section resolve many of these problems in a satisfactory manner.

3.1 General Overview

The basic unit of rembassy execution is the daemon or agent. A rembassy daemon functions in an automated way, planning and executing the configured monitoring activities.

However, the true capacity of rembassy stems from a potent communication mechanism that permits the rembassy daemon to communicate with a user client or other daemons. This system is based on the remote calling of methods by means of XML-RPC [14]. The choice of this protocol was determined by its simplicity compared with other alternatives such as SOAP [15], whose greater complexity involves an excessive, and in this case unnecessary, communications cost.

The clients communicate with the daemon not only to obtain the monitoring data, but also to change the configuration of the system. In this way, it is possible to construct powerful interfaces that permit a user to manage the rembassy daemon. The rembassy web interface is an example of this mechanism.

The communication among daemons makes it possible to have a distributed monitoring system. In this case, monitoring chains can be managed transparently from only one daemon. This distributed monitoring scheme with centralized management is not found in any of the applications studied.

The rembassy daemon is formed by two fundamental components: the core and the plug-ins. The core implements a set of basic functions, which are described in section 3.3. The plug-ins use these as a basis for developing more advanced capabilities.

A rembassy daemon maintains a series of hierarchically organized objects. Each one of these objects is responsible for storing one or various configuration parameters and / or consulting the monitoring state of an element. All of them are capable of attending to remote petitions. The rembassy daemon will delegate all the received petitions to one of those objects. The plug-ins extend the subset of objects defined in the core, permitting the creation of a large variety of different objects. This mechanism makes rembassay a very powerful and easily extendable system.
Each daemon maintains its own objects structure, but thanks to the XML-RPC petitions system, it is
possible to manage this from a different daemon. The “proxy” plug-in integrates the objects structure of a
remote daemon into the local objects structure in a way which is completely transparent to the user. The
management turns out to be, therefore, independent of the chosen monitoring model.

In addition, rembassy distinguishes between persistent and non-persistent objects. The non-persistent
objects are loaded into memory when the daemon is executed while the persistent objects are stored in a
relational database and they are loaded into memory when they are accessed.

3.2 Monitoring Objects

The proposed architecture contemplates three categories of monitoring objects: sensors, probes, and services.

The sensors are the lowest level monitoring objects and they have as a mission to return the monitoring
state of an element in function of a series of parameters which can be either obligatory or optional. Normally,
they are non-persistent objects that merely respond to each petition without saving any type of internal state.

The probes behave similarly to sensors, but as opposed to them, the probes are persistent objects which
maintain information. The probes delegate the monitoring in a sensor, which permits the fixing of certain
parameters beforehand. In a way, they are like higher level sensors, and in fact, they can be used as if sensors
were being dealt with. The principal utility of the probes is the creation of monitoring templates.

Finally, the services are the highest level monitoring objects. Just like the probes, they delegate the
monitoring in a sensor, but they are different in that all of the mandatory parameters have to be established
prior to their use. Therefore, a service can consult its state without the need for the user’s interaction, which
permits its automatic execution by the rembassy daemon. In this way, the user merely has to indicate the
verification interval for each service, and rembassy automatically plans its execution in the designated
intervals.

Another difference between services and probes is the possibility of fixing parameters associating probes
to monitored entities.

A monitored entity is a type of object that saves information relative to a determined entity (for example,
a host). If this information changes, the associated services automatically proceed to use the new values.
Thanks to the use of monitored entities, in case of changing the value of an attribute (for example, the IP of a
host), it won’t be necessary to modify this information in each one of the services.

The relation among the various monitoring objects can be more easily understood in the following
example. Suppose we want to monitor the state of a series of machines in the network using “ping”. If any of
the machines does not respond, we would like to receive an error message. In addition, we are also interested
in being warned in case the response time is especially long.

With rembassy, this situation could be resolved in the following way. In the first place, there would be a
sensor responsible for realizing the “ping” of a machine. This sensor could also take as parameters the IP
address and optionally the maximum desired response time. In spite of the fact that various machines are
going to be monitored, we most probably would like the same response time for all of them. Instead of
copying the same value in each case, a probe can be created to set the response time to the value desired. This
probe would serve as a template for the services. Due to the fact that rembassy automatically plans the
monitoring of the services, we have to introduce beforehand the value for the other sensor parameter: the IP.
However, instead of introducing it directly, it is suitable to create a monitored entity for each host, and link it
to the IP parameter. In this way, a possible change in a host IP will only need the change of its value in the
corresponding monitored entity. If we are using this value in various services, the usefulness of this
mechanism is considerable.

3.3 The Rembassy Core

As has been commented on previously, the rembassy daemon is formed by two fundamental components: the
core and the plug-ins.

The core offers the basic tools that will permit the plug-ins to develop their functionalities. It is made up
of the following elements (see Figure 1):
- Log Manager. It manages the system logs, where all the events produced during the execution of the application are registered. The level of detail is configurable, permitting the user to choose among different log levels, which range from the mere registration of the execution errors until the registry of abundant debug information.

- XML-RPC Server. It permits the communication of the daemon with external clients, using the XML-RPC protocol. It also tolerates the use of SSL (Secure Socket Layer), which permits the guaranteeing the security of the communications. The petitions that the server receives are really remote calls to methods of the objects managed by the Object Manager, which will delegate its execution through them.

- Database. It is responsible for managing the system DB at a low level. It is used to save information in a persistent manner. This component utilizes the SQLAlchemy [16] object-relational mapper in order to simplify access to data, which favours the extensibility of the application. Each persistent object is only responsible for defining the tables where its state will be saved and for indicating the relation between the DB columns and the object attributes to the mapper.

- Object Manager. It manages the hierarchy of the system objects, being concerned with its creation, its upgrading, and its elimination. This hierarchy is based on containers and sub-containers that come from a root and store the system objects. In the case of persistent objects, it will delegate these operations in the Database component. The non-persistent objects are created upon initiating the rembassy daemon under the virtual container “sys”. A big advantage of this system compared with ones used by other tools or defined in the Standard SNMP is the possibility of adding or eliminating monitoring objects during execution time.

- Security Manager: It is the component responsible for guaranteeing the security of remote calls. It carries this out by authenticating the user and verifying if the permits are the correct ones for calling in a specific way. The rembassy security system is extremely flexible. Each object associates one or more permissions to its methods. It is possible to create specific permits per method, or else permits which affect various methods. The rembassy administrator can alter these permits by modifying the corresponding configuration files. From the point of view of the user, the security mechanism is based on roles. A user will have one or various roles, each one associated to a set of permits. In order to execute a method, the user will have to possess a role with some of the permits associated with the method. Rembassy offers two predefined roles: admin (allows the system configuration) and users (only permits viewing the created objects), although it also permits the creation and configuration of new roles. The authentication is based on passwords. Each user has an associated password which is stored in the database and encoded with the SHA-1 hashing algorithm.

- Scheduler: It is the object responsible for planning the periodical controls of the configured services that are to be monitored. It utilizes an algorithm which distributes the charge during the whole time and it is concerned that the services are verified in the selected intervals but without overloading either the monitored or the monitoring machine.
3.4 The Plug-ins

The plug-ins system permits the extension of the basic functionality offered by the core. The rembassy architecture is designed to be easily extensible, delegating in the plug-ins the majority of the system’s monitoring activities.

Special emphasis has been placed on the necessity of having a simple and powerful extension mechanism. As was previously commented upon, this is one key aspect in the design of a good monitoring system. In rembassy, the plug-ins design is extremely simple. It is sufficient to extend the appropriate objects of the core. Complex questions such as the persistency, the remote invocation of methods, the management of objects, or the planning are automatically managed by the core.

The core offers a set of interfaces and basic objects which can be extended to simplify the creation of plug-ins. For example, suppose we want to implement the sensor of the previous example, which realizes a “ping” in a machine, and also verify the response time. To do this, we extend the base sensor defined in the core. This base sensor implements the logic that manages the relation between the sensor and the rest of the core, which simplifies the creation of the new sensor. We only need to implement two methods: one which returns the parameters that support the sensor (IP and the maximum response time), and the other one which receives these parameters and returns the corresponding state, after making ping in the indicated machine.

3.5 User Interface

The interaction between the user and rembassy is realized by means of a Web interface. The fact that this behaves like any other client, communicating with the daemon by means of XML-RPC guarantees the independence of this from the system core, and facilitates the future elaboration of other types of user interfaces.

The designed Web application permits a complete rembassy management, including both the monitoring and the configuration of the system. This implies a clear advantage over the usual configuration based on files in Open Source monitoring tools. The Web interface greatly facilitates the rembassy configuration, resolving in this way one of the principal problems found in the current existent applications.

In order to facilitate the integration of the plug-ins in the Web interface, a system of adaptors has been designed. The adaptors are used for personalizing the graphic aspect of the part of the Web interface that permits the user to interact with the plug-ins. The developers who want to define a specific interface for their plug-ins have to implement the corresponding adaptor and register it in the system. Just like the plug-ins, the implantation of adaptors is extremely simple. In addition, it is totally optional, because rembassy possesses an interface by default which permits the user to execute the methods of any plug-in.

4. SYSTEM IMPLANTATION

The development of rembassy was motivated by the need to monitor the network of the Department of Information and Communications Technology of the School of Computer Science at the University of A Coruña. In this section, we briefly present the steps followed in the configuration of rembassy in the monitoring of this environment, in which the following servers are going to be monitored:

- A web applications server (tenca) which gives support to the network of the School of Computer Science at the University of A Coruña and to various management applications, all of them implemented with J2EE technology. Tomcat is used as a J2EE applications server, and Apache as an HTTP server. The persistence in the web applications is implemented by means of a remote DB, which is accessed through JDBC (Java Database Connectivity).

- A web server (marraxo) which hosts a small web page implemented with LAMP technology. It operates an Apache server and a MySOL data base, used as persistent support of the web page’s dynamic information. It contemplates public access to the web via HTTP and private access via SSH (Secure Shell) in order to manage the contents.

- An Oracle Data Base server (lubina), which gives support to the persistent information of the tenca web applications.
An NAS - Network Attached Storage- (carpa) which is used to store the backup files of the other three servers. A script in lubina server is responsible for the backup process, which gains access to the carpa hard disk via SAMBA.

In our study, we will monitor the correct functioning of these four systems as well as the services available in each one of them. A distributed monitoring scheme will be used, centralizing management from a Linux client. In this case, three types of sensors are used, depending on the parameters to be monitored:

- Local parameters: Disk, Memory, etc.: These parameters cannot be consulted directly by means of the network, and, therefore, they need the installation of a rembassy agent in each system. Utilizing the “Proxy” plug-in, it is possible to manage these types of parameters in a centralized way. Rembassy will be responsible for sending the remote daemon requests to the user in a transparent fashion.

- Standard TCP services: The sensors can gain access to the monitored parameters by means of the network. In the majority of cases, this permits a centralized monitoring, using available sensors in the monitoring system, without having to install a rembassy daemon in each monitored system.

- Applications which are accessible through the network: Just like the aforementioned ones, they consult the state of the service to be monitored by means of the network. The principal difference is that while the aforementioned ones monitor standard services, which are well known and supported by the operating system, these ones are going to need the installation of drivers or specific clients in the monitoring system. For example, this is the case with the Database.

The probes permit the defining of the monitoring templates, and are very useful in the case of monitoring similar services in multiple machines. In this particular case, there are two http servers, meaning one probe will be created to group the common parameters from both of them (sensor, return code…).

Once the necessary probes are created, it is possible to define the services which are going to be used to monitor our system. As has been previously mentioned, the services are going to be executed by the scheduler, meaning that all the parameters have to be introduced beforehand.

Rembassy’s Web interface permits the creation of a service, the choice of a sensor in which the monitoring will be delegated, and the establishment of fixed values for all the parameters.

Upon establishing the verification interval, rembassy automatically schedules the execution of the service. Once all the services are created, the user’s interaction is no longer necessary. Rembassy is in charge of monitoring all the scheduled services, and storing a record with the changes in the state, which the user can verify at any moment.

Finally, rembassy makes it possible to analyze the data stored in the registry. This is useful in case we are monitoring parameters not only for their correct state but also if we are interested in following a certain related numerical value. For example, if we are monitoring a system’s memory, we are not only interested in being warned when the percentage of utilization surpasses a certain threshold, but we also want to verify how this value evolves throughout time. The analysis of this evolution permits the anticipation of future problems.

![Figure 2: Rembassy permits the representation of graphics from the registry](image)

This function is still in the developmental process, even though graphics are already available (see Figure 2) so that the user can analyze the evolution of certain parameters.
5. CONCLUSIONS AND FUTURE RESEARCH

In this research, a novel architecture in the area of applications monitoring has been developed. It resolves the principal problems and deficiencies of the existent monitoring applications in the Open Source world. Rembassy contributes functional and design characteristics which were previously unknown in this environment.

The architecture has been designed with extensibility and scalability in mind. The plug-in system is especially innovative because of its capacity and simplicity, as well as its hierarchical structure of objects in which its design is based on.

In the area of configuration and use, rembassy incorporates important improvements over the existent alternatives, such as the centralized configuration from the web interface or the system of monitoring templates at various levels. Its flexibility is also an innovation, permitting the implementation of various monitoring schemes and their multi-platform characteristics.

As far as future applications and improvements are concerned, it should be pointed out that the development of the system is continuing, and in the future, the following aspects will be concentrated upon:

- Notification of incidences. A system will be implemented for real-time notification of possible incidences by certain means such as electronic mail or SMS. Some Open Source tools already support these types of notifications. In rembassy, this functionality is currently in development, and it will be implemented with a new family of objects (notifier) which they will be able to associate with services according to its alert state.

- Improve the analysis capabilities. At the moment, the rembassy representation system is limited. The development of a more advanced system that permits the comparison of various parameters or a statistical analysis of the data would be enormously useful.

- Dependencies among services. The creation of dependencies among services facilitates the monitoring, because in case one of them fails, you avoid checking the state of others that depend on it. The majority of Open Source tools do not have this characteristic.

- Development of plug-ins. Up until now, the principal work objective has been the development of a suitable architecture to resolve the principal existent problems. Once this task is completed, the focus will be more and more on the elaboration of plug-ins to increase the capacity of the system and its utility in different environments.

REFERENCES