# WebManager: Transforming a Network Management Application Into a Component-Based Framework

Jacques Philippe Sauvé
Antonio A. T. R. Coutinho, Rodrigo R. de Almeida, Ayla D. D. de Souza, Alexandre N. Duarte

Departamento de Sistemas e Computação - UFPB/Campus II
Rua Aprígio Veloso, CEP 58109-970, Campina Grande - PB
Telephone: +55 (83) 310-1120 Fax: +55 (83) 310-1124
{jacques, coutinho, rodrigor, ayla, alex}@dsc.ufpb.br

## Abstract

This paper describes a network management tool. Visits to organizations of all sizes reveal that more than 95% of them are empty-handed as far as network management tools are concerned and pursue ad hoc management techniques. We argue that this sad state of affairs is due to the lack of cheap, easy to use tools. Although we don't solve the problem completely by providing such a tool, we show how a black-box component-oriented framework can be the basis for constructing it. We report on results of actual use of the tool to manage a small network and show that the postulated requirements (flexibility, scalability, ease of use and portability) are met.

## 1 Motivation

This paper describes a network management tool. Like all tool builders, we build new tools because we think adequate tools are missing for a particular purpose. But do we really need new tools for network management? Are current tools not adequate? In this section, we explain why we believe new tools are needed. Sections 2 and 3 describe the tool we built from the perspective of Goals and Requirements and Architectural Design. Results are presented in section 4 and future work is described in section 5.

Why do we say that adequate tools are missing for network management? Are not mature platforms such as HP's Open View or Cabletron's Spectrum, coupled with the plethora of plugin applications adequate to manage current networks? A short answer is "yes and no", but we will concentrate on the "no" aspects in what follows. As part of our day-to-day activities, we have been in contact with many organizations over the years and certain facts have become clear concerning these organizations' networks and management policies. We summarize: A quick head count reveals that well over 95% of organizations lack adequate network management. As network management professionals, it behooves us to find reasons for this dramatic situation. We think that *tools*, or the lack thereof, is at the heart of the problem. Are there no tools for network management? There are, of course, but the calamitous situation outlined above is primarily due to the two following tool-related factors:

- Good network management tools are too expensive; and/or
- Network management tools are hard to use; that is, they require hard-to-find expertise to be useful.

The last statement is generally true of *all* network management tools. Installing and configuring the tools is difficult, at least for the level of expertise shown by typical network support personnel. Also, tools are either not extensible or very hard to extend. As an example, how would you extend your current network management tool to provide full end-to-end e-commerce site management? It is probably impossible, or very hard, or very expensive.

We have started a project aiming to produce very-easy-to-use network management tools. In our experience, producing *really* easy-to-use software is difficult. The model we aim at is what we call the "big red button". Ideally, network personnel should see a network management tool as a big red button. Push the button and the tool will work on its own and provide painless network management. This is obviously a tall order, undoubtedly unattainable, but the metaphor serves well to show what we are aiming for. We also understand that, if we are successful, the final tools will probably only be adequate for small networks (of hundreds of managed nodes).

A first tool, called WebManager, was developed [Sauvé 1999] and is currently being used to manage 6 networks, ranging from a tens to hundreds of managed nodes, including campus and wide area networks. We do not claim that WebManager, Version 1 (wmv1) is easy to use. However, we have gathered experience through the tool, especially in the areas of performance and fault management. The success we have had with this tool has prompted us to pursue our efforts further and improve it. Specifically, wmv1 suffers from the following major shortcomings:

- The tool is *difficult to install and configure*. A text-based configuration file must be edited with full details about network topology, interface descriptions and management policies.
- The tool is *not scalable* and barely has the performance to manage a network consisting of a few hundred managed nodes, using a run-of-the-mill Pentium computer.
- The tool is *not extensible*. Factoring in new functionality requires an understanding of all the source code and can currently only be done by the tool's creators.

## 2 Goals and Requirements

We want to solve two important problems: *scalability* and *extensibility*. Why are these two immediate goals more important than others? First, it has become clear that the final goal - the "big red button", extremely easy-to-use network management tools - will not be achieved with wmv2. Very easy-to-use network management software is very difficult to produce. It may require 10 or more versions to reasonably approach that goal. Many modifications will occur over time and it is therefore imperative that intermediate steps be based on an extremely flexible and extensible architecture, able to accommodate change. Secondly, we believe that scalability is achieved by good architectural design and not by tweaking badly designed software after it is found not to be scalable. It is thus also imperative to include scalability since the beginning so that adequate architectures may be tested as early as possible. Finally, a third major goal is to make wmv2 substantially easier to use than wmv1. In fact, it is a permanent goal of the overall project that ease of use should steadily increase from version to version, even as functionality increases.

This said, we may now list our major requirements:

1. All functionality available in the previous tool (wmv1) must be available. Details ate available in [Sauvé 1999]. This includes:
   - A Web interface;
   - Support for SNMP versions 1 to 3;
   - Navigation using hierarchical network maps;
   - Color-coded device status available on the network maps;
   - Ease of generating historical performance graphs for any MIB variable;
   - Configurable alarms with alarm logging, and notification through mail messages or similar mechanism.
2. The tool should be much more scalable. It should be possible to manage thousands of network elements using standard PC hardware (not multi-processed, not

clustered, run-of-the-mill Pentium-based PCs with a standard memory configuration of, say, 64 MBytes to 128 Mbytes.)

3. The tool should have an extensible architecture, allowing the addition of functionality by a third party without requiring the tool's source code. As an example of extensibility, it should be straightforward to introduce new event correlation algorithms and generate alarms accordingly.
4. The tool should be much easier to install and configure;
5. The tool should be portable and should run unaltered on Linux and Windows NT platforms.
6. The tool should be freely available and thus should use freely distributable accessory software. Cost should not be an issue.

## 3 The Proposed Architecture

Our solution to the problems outlined above can be described at several levels of detail. However, due to the nature of the three major goals (extensibility, scalability and ease-of-use), we deem it sufficient to outline the architecture of the tool, without delving into detailed design issues.

### *3.1 Extensibility through a component-based framework*

Let's tackle the first and most difficult requirement: extensibility. It is a difficult requirement because, by its very nature, it refers to *unknown* future functionality. Planning for the unknown is always tricky. Over the past few years the discipline of software engineering has shown several ways to achieve extensibility, one of the most important of which is, in our opinion, the use of *component-based frameworks* [Fayad 1998, Szyperski 1999].

Frameworks can be classified as *white-box* and *black-box* frameworks. White-box frameworks rely on class inheritance to complete the framework. The programmer extends certain abstract classes provided by the framework in order to provide the missing pieces. By contrast, a black-box framework relies on object composition to plug in missing pieces and complete the applications. In a fully mature black-box framework, the composition would be done using a visual builder [Roberts 1997] and without needing access to the objects' source code. Also, objects would be configured declaratively (that is, without programming). In such a situation, we say that the objects being composed are *components*. In other words, components are configured and composed together using a visual tool to form complete applications.

We want wmv2 to be a component-oriented-framework. This provides a very flexible and extensible architecture for two reasons:

- The selection of which components to plug into the framework, where to plug them in, and how to configure them provides the flexibility to build different applications;
- Extensibility is achieved by simply producing new components. For example, new event correlators or "e-commerce management components" could be produced and plugged into the framework, as long as the components' functionality can seamlessly be integrated into the architecture.

### *3.2 The overall n-tier architecture*

Before delving into the design of the framework itself, it is important to decide exactly what aspects of the overall solution the framework will cover. We chose to remove any consideration of user interface from the framework so as to maintain simplicity and the flexibility of providing any type of interface, including an HTML/XML Web browser-based interface, very thin clients such as WAP-enabled devices, GUI consoles, etc. We therefore base the solution on a standard n-tier architecture, as shown in figure 1. The only other major decision also taken at this point is to leverage Java technologies to gain portability and ease of development. This implies that framework components will be JavaBeans.

In figure 1, presentation services rely on browsers, WAP-enabled clients, etc. in the first tier. The second tier would typically be a Web server serving HTML, XML or WML pages produced from Java Server Pages (JSPs). These pages instantiate components called Proxy View Components. They are proxy components since their function is merely to communicate with the true view components running on the Network Management Station through Remote Method Invocation (RMI). The framework itself, in the third tier, offers view components that provide "object-oriented views" into the network management information available. These components exist only to provide information to be displayed to the user through the chosen interface. The framework obtains information from the network (which we may consider to be another tier) and keeps management data in yet another tier, the persistent data store. This overall architecture thus retains the ability to deal with thick, thin and very thin clients, thus providing the required flexibility in user interface.

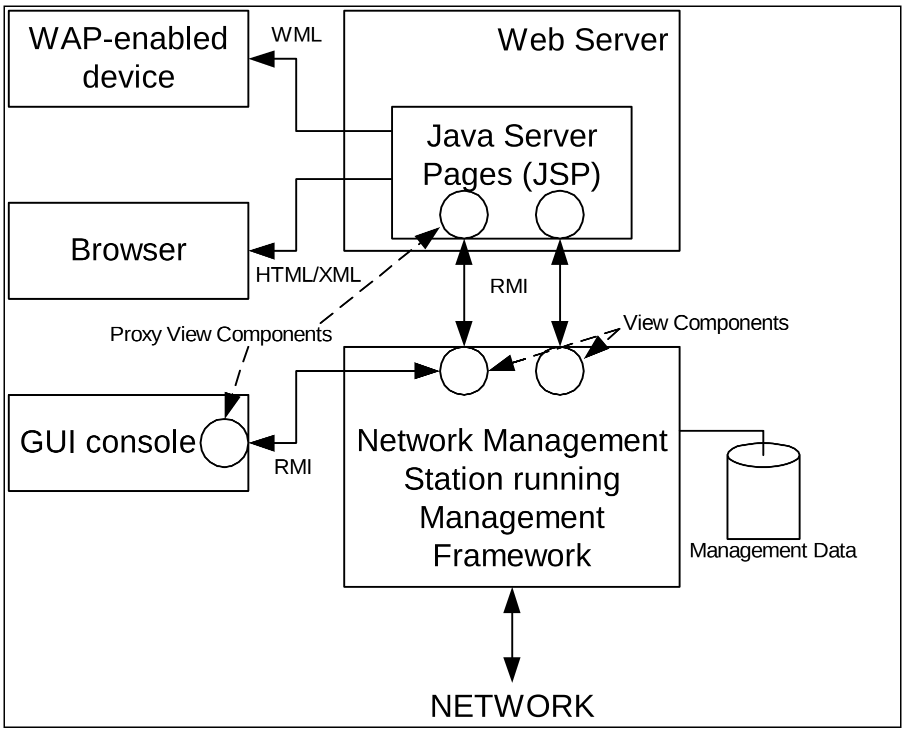


Figure 1. The overall n-tier architecture for WebManager, Version 2

We now describe the management framework itself, the most important part of the architecture.

### 3.3 The problem domain model

First, we emphasize that the framework is aimed specifically at the areas of *performance* and *fault* management, which are of particular interest to us.

Let us look at a general model for fault management, in figure 2, adapted from [Huntington-Lee, 1997]. The figure shows clearly that the structure of a fault management solution is essentially a pipeline of information, where the type of information flowing in the pipeline depends on the particular location. In the figure, we can identify:

- *Network activity*, as a result of SNMP polling or trap reception, for example;

- *Network events*, indicating special conditions such as a particular management variable crossing a preset threshold;
- *Network alarms*, obtained after filtering events, such as would be done by an event correlator;
- *Trouble tickets*, representing alarms that must be tracked until the problem is resolved.

We may now ask: what would change in figure 2 if performance management were included? Performance management deals with two major issues: performance problems and trending for capacity planning. Identifying performance problems requires no change to figure 2: it is simply a matter of defining appropriate threshold and grouping filters. In order to allow trending to be done, it is sufficient to log network activity (and not only network events as shown in figure 2) and make it available to trending analysis modules. We may therefore adopt figure 2, along with a network activity log, as the problem domain model that must be captured in the framework we wish to design.

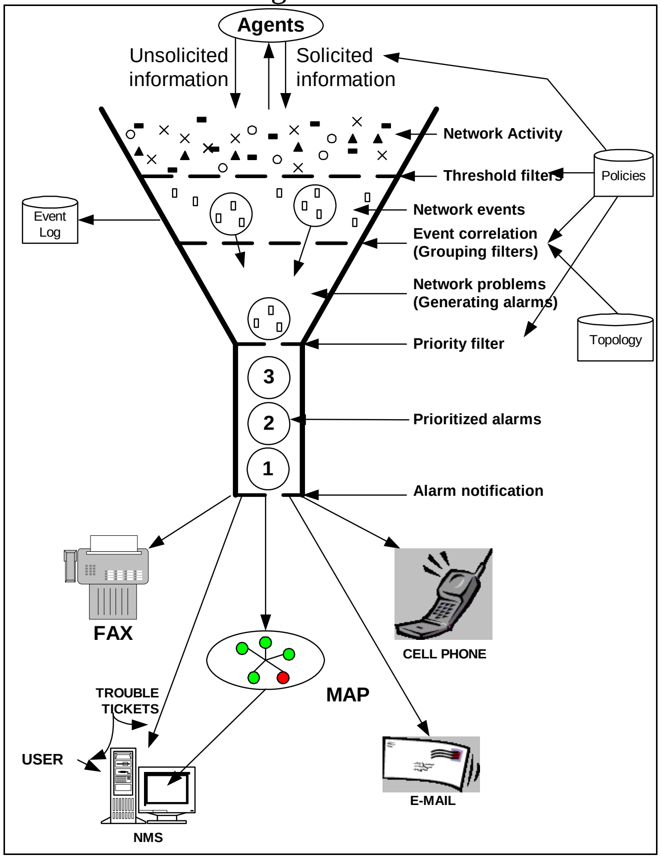


Figure 2. A model for fault management

Having decided on a problem domain model, we may now proceed to the design of the desired management framework.

### 3.4 Information producers and consumers

In transforming the problem domain model to a software architecture, we again observe that the model is essentially a pipeline of information with elements performing the transformation of information from one form to another along the pipeline. We may thus think of producers and consumers of information and use these basic entities in the software architecture. As an example, a threshold filter is a consumer of information (representing network activity) and also a producer of information (representing network events). *The basic architectural decision is to represent each information producer and consumer as a software component (a JavaBean).*

Information producers and consumers must be connected to one another. In other words, the information produced by one component (such as a threshold filter) will usually be sent to one or more components (such as event correlators) which will consume the information and possibly produce new information for yet more consumers. Exactly which components will be connected to which others is totally dependent on management policies that must be implemented by the framework. Since management policies are established by the organization, and since they also depend on the network topology, they are dynamic and the framework must therefore support ways of connecting the components at *run-time* rather than at *compile-time*. This, essentially, is what a black-box component-based framework allows us to do. But we still have to decide on the way in which components are made aware of one another.

An object-oriented design pattern called *Observer* is ideally suited to the situation [Gamma 1995]. Using this design pattern, an object that is interested in an event produced by another object registers its interest in the event with the producing object. Any object that produces events has an interface (a set of methods) through which it accepts the registration of interested objects. When an event occurs in this producer object, it promises to inform all interested objects by calling an appropriate method. The advantage of the Observer design pattern is that the coupling between objects is not statically coded but established at run-time and may be dynamically altered. This is ideal for our situation, since we will be able to design and code all components without regard to the final connections they will have with other components. This yields an extremely flexible architecture.

We use even more powerful way of establishing component connections. It resembles the Observer pattern and decouples components even more. We call it the Databus architecture, similar to Sun's Infobus [Sun 1999]. A Databus is a software bus that works similarly to a hardware bus: any information put on the bus is distributed to all components connected to the bus as receivers (consumers). Thus, instead of connecting components to one another, producers will put information on an Databus and interested consumers will connect themselves to the Databus and receive the information. This minimizes coupling still further, since components are no longer coupled to one another but to Databusses.

Many Databusses may exist in a typical instance of the management framework. Some will be present in any application based on the framework. Details will be given in the next section. However, it is important to observe that there must be a simple way of creating *new* Databusses, rather than having only fixed Databusses in the framework. There must also be a way of dynamically connecting components to the appropriate Databusses, either as producers or as consumers. In order to achieve this effect, each Databus has a *name* and any component can define any number of *Databus interfaces*. An Databus interface has two attributes: the name of the Databus to which it should connect and a type (PRODUCE or CONSUME) either to produce on the bus or consume from it. Figure 3 shows a component receiving information from an Databus called "EVENT" through a CONSUME-type interface and producing information on another Databus called "ALARM" through a PRODUCE-type interface. Observe that a component may have fewer or more than two Databus interfaces. Typical components will have one Databus interface (sources and sinks) or two Databus interfaces (filters).

## ***3.5 The framework***

The framework is shown in figure 4. All circles are components (JavaBeans). Three standard Databusses are always part of the framework, although more can be created for special situations. The standard Databusses are used to distribute Network Activity information, Network Event information and Network Alarm information. Databusses are also JavaBeans so that the whole framework really consists of pluggable components. This fact is what makes the framework of the black-box type.

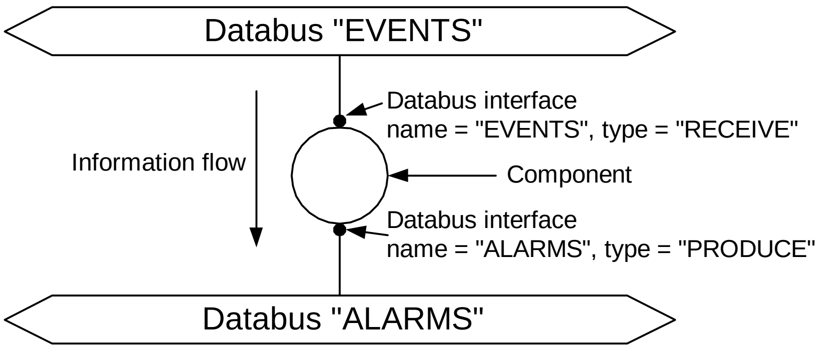


Figure 3. A component with two Databus interfaces

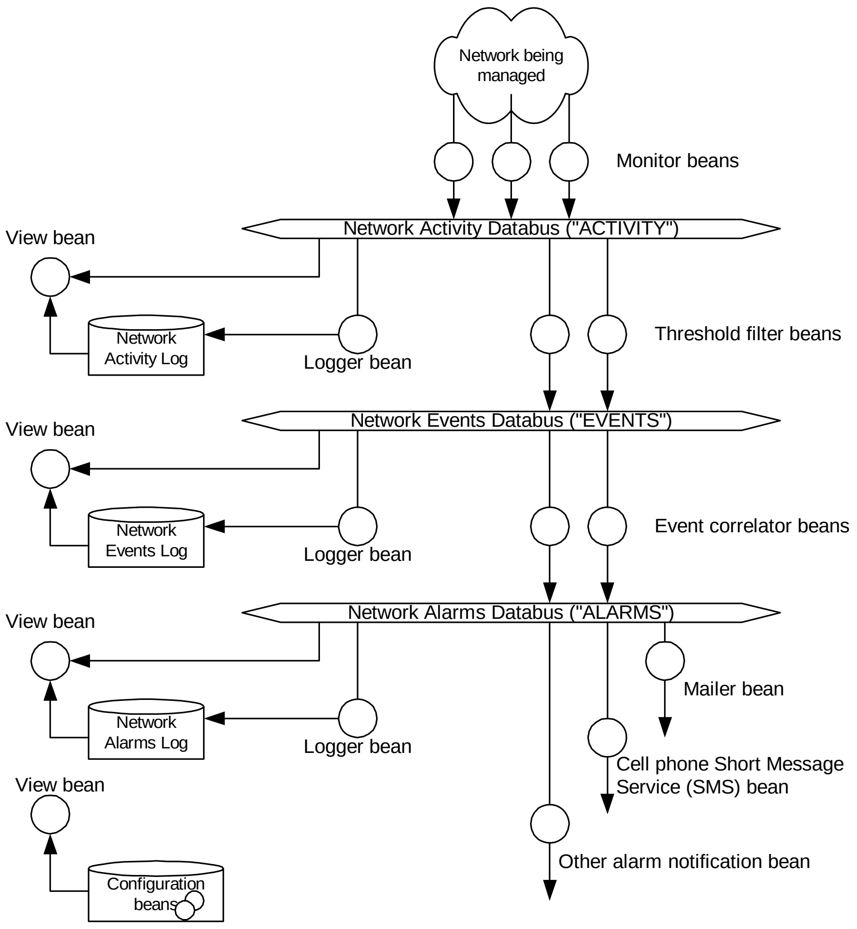


Figure 4. The network management framework

Producers for the "ACTIVITY" Databus are *Monitor beans*. These components are responsible for obtaining management information from the network and making it available on the Databus. All Monitor beans have a single Databus interface. We have developed three types of Monitor beans. One uses SNMP to monitor SNMP variables, another receives SNMP traps, while the last monitors round-trip time using the ICMP protocol (as the *ping* command does). Other types of Monitor beans could be develop, as discussed in the Results section.

The next major type of bean represents *threshold filters*. These beans have two Databus interfaces, one to consume from the ACTIVITY Databus and one to produce information on the EVENTS Databus. We have implemented a single type of threshold filter, one that uses hysteresis (with a *fire* threshold and a *rearm* threshold). Properly setting the two threshold values and whether events are produced as each threshold is crossed allows one to adequately treat all conditions necessary for fault and performance management.

Also shown in figure 4 are *event correlator beans* with two Databus interfaces (between the EVENTS and ALARMS Databusses). This is where the real intelligence of network management can reside. So far, we have produced few such components. One is a pass-through component that is used when each event should generate an alarm. Another is a component used for preventive network maintenance that inhibits event generation between two instants in time.

*Logger beans* possess a single Databus interface and are used to log Databus information to a file or database. Three different logger beans are necessary since the information flowing on each Databus is different. Logged information is important to supply the user interface with appropriate values of historical data, device status, alarm tracking, etc.

Notification of network alarms can be done in several ways. We have developed two *notification beans*, one which sends e-mail when an alarm occurs (*Mailer bean*) and one which send a message to a cell phone using the Short Message Service (SMS) available from phone companies. Currently, our SMS bean sends a DigiMemo to any BCP-owned cell phone in Brazil.

Topological information is contained in beans stored in a configuration database. There is basically one bean per network equipment being managed.

Notification can also be performed by changing the information on network maps. However, since network maps are generated by a pull model (browsing to a particular JSP page will pull information from the server to the browser), we cannot use the same solution as other notification beans which operate on a push model. This is the reason for the existence of *View beans*. These beans are passive in the sense that they do not push information further down the pipeline. Their methods are called by the JSP pages (or other clients) that provide a graphical user interface for the application and access the appropriate databases to return the required information. In this sense, they furnish an "object-oriented view" of all management data. They are accessed indirectly through Proxy View beans, as shown in figure 1, since clients are running in a different Java Virtual Machine than the framework (in a Web server, for example). We have developed several types of View beans, as described below:

- Status bean: this bean knows the current status (up/down) of all managed devices and may be queried to update network maps with the appropriate color coded information (red for down, green for up, and so on)
- Config bean: providing configuration information such as a list of network elements, their network interfaces, descriptions, names, polling intervals, community names, etc.
- Map bean: capable of drawing network maps and generating GIF-format files for network navigation in a browser or WAP client environment.
- Graph bean: capable of drawing statistical graphs of management information versus time. These graphs can be used for trending or to obtain a vision of network operation over time.

### 3.6 Scalability considerations

One of the major goals of this version of WebManager is to achieve a scalable architecture. In our view, achieving scalability requires two conditions to be met. First, in a centralized architecture, heavy use of threads is necessary to avoid sequential execution. In our architecture, every managed element has a Monitor bean associated with it to monitor the element's agent and each Monitor runs in a separate thread. Thus, network monitoring is not sequential but done independently for each element.

### *3.7 Application assembly: configuring the framework*

We have briefly described a component-based framework for creating network management applications. Summarizing, we have a set of components and an architecture for connecting these components to one another to perform a particular management task. We now have to discuss how *application assembly* is done. When using a black-box component-based framework, applications are created by assembly, that is by instantiating components and plugging them into the framework. This means that we have to decide:

- Which components exist;
- What their properties are (a property is a component attribute that may be altered declaratively, without programming, usually through a Visual Application Builder); for example, a Monitor bean has a property called "Polling frequency", another called "Community name", and so on.
- How components are connected to Databusses;

Since the connection of components to Databusses is done merely by setting the component's properties (the Databus name and interface type - CONSUME or PRODUCE), we must therefore indicate which components to instantiate and what values their properties should assume. *We stress that this is all that must be done to produce a full network management application.*

We have chosen to represent this information through an XML configuration file. The use of XML makes the configuration information easy to edit, generate automatically, store, browse, etc. Figure 5 shows a small part of an XML configuration file. We do not give a full example due to space restrictions. For ease of use, the configuration file is edited with a graphical XML editor.

```
<NetEquipment>
   <name>sw8273</name>
   <hardwareDescription>RouterSwitch IBM 8273</hardwareDescription>
   <functionalDescription>POP-PB main router</functionalDescription>
   <criticality>9</criticality>
   <host>sw1.pop-pb.rnp.br</host>
   <port>161</port>
   <protocol>0</protocol>
   <retries>0</retries>
   <timeout>4000</timeout>
   <pollInterval>5</pollInterval>
   <pollsToSave>3</pollsToSave>
   <community>RNP</community>
   <writeCommunity>private</writeCommunity>
   <instanceIndex>0</instanceIndex>
   <statusOID>.webmngr.ping.PingUp</statusOID>
   <allGraphics>
         <variableGraphic>
               <title>Ping time</title>
               <oid>.webmngr.ping.PingTime</oid>
               <label>ms</label>
               <limit>1000</limit>
         </variableGraphic>
   </allGraphics>
<NetEquipment>
```

Figure 5. Application configuration in XML

When the framework's main method starts, it reads the XML configuration file, instantiates all components, sets their properties according to the XML tags found in the file, and connects them together according to the Databus interfaces defined in the properties. Most of this work is done by another framework (called JCONFIG) specially created to ease the configuration of component-based applications.

### *3.8 Application installation and deployment*

Installing an application based on an n-tier architecture can be complex. Not only must the framework-based application be installed and configured but the other tiers must also be configured. This is especially tricky since we need a Web server with support for JSP and all this must be properly configured. Remembering that we have a major requirement to enhance ease-of-use, we have elected to package all needed modules in a single installation package and not depend on external Web servers, JSP plug ins, etc. Currently, a single installation script installs and configures everything, including a Jakarta Tomcat servlet/JSP server [Apache 2000] listening on a special port.

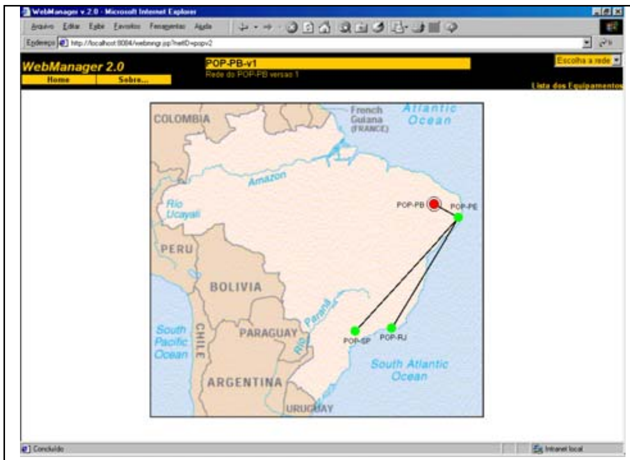
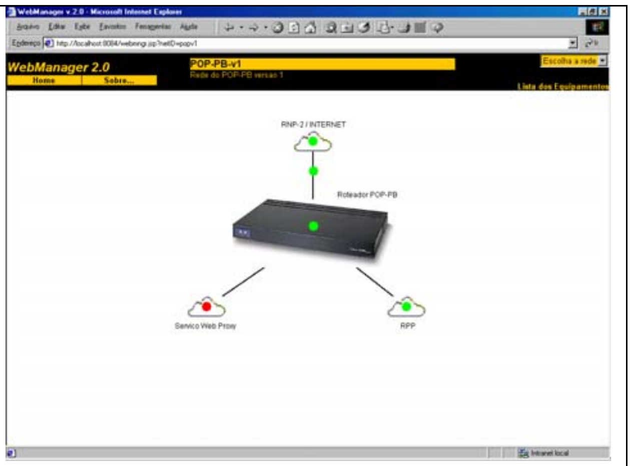
## 4 Results

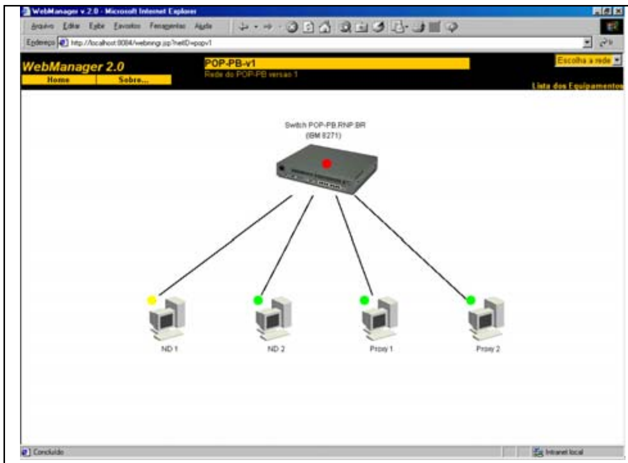
We have implemented the framework in Java and are currently using it to manage a small network consisting of about 10 managed nodes. We are substituting wmv2 for wmv1 for all 6 networks still being managed with WebManager, Version 1. Figure 6 shows six screen shots of our application when managing a small network. Figure 6.1 is the main network map; clicking on the red spot (indicating a severe problem) yields figure 6.2. By further clicking on the red trouble spot, we reach, in succession, figures 6.3, 6.4 and 6.5. This figure is no longer a network map but a detailed view of statistical information available about the network equipment. Clicking on a particular graph yields figure 6.6. Several types of reports may easily be generated using JSP pages such as a list of managed elements together with their status, an alarm browser, etc. We also have an operational WAP version but space restrictions prohibit use from showing screen shots.

How do we validate our work? How do we measure success, or lack thereof? We must turn to our requirements and see how fully they were attained.

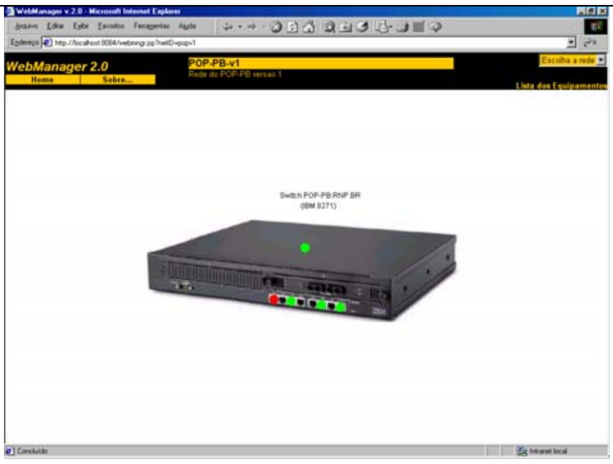
### *4.1 Portability*

The requirement for portability was fully achieved, mainly thanks to Java and to the portability of the Jakarta Tomcat server. The solution is working under Linux and Windows NT. It will probably easily work under other platforms, although this has not been tested.

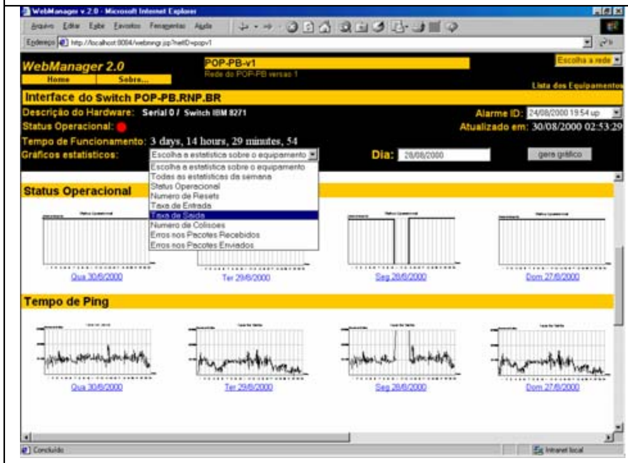
| | |
|---|---|
|   WebManager v.2.0 - Microsoft Internet Explorer<br>Arquivo Editar Exibir Favoritos Ferramentas Ajuda<br>Endereço http://localhost:8084/webmngr.jsp?netID=popv2<br>WebManager 2.0<br>Home<br>Sobre...<br>POP-PB-v1<br>Rede do POP-PB versao 1<br>Escolha a rede<br>Lista dos Equipamentos<br>COLOMBIA<br>French Guiana (FRANCE)<br>Atlantic Ocean<br>Amazon<br>Rio Ucayali<br>PERU<br>BOLIVIA<br>PARAGUAY<br>CHILE<br>South Pacific Ocean<br>ARGENTINA<br>URUGUAY<br>Paraná<br>Rio<br>POP-PB<br>POP-PE<br>POP-SP<br>POP-RJ<br>South Atlantic Ocean<br>Concluído<br>Intranet local |   WebManager v.2.0 - Microsoft Internet Explorer<br>Arquivo Editar Exibir Favoritos Ferramentas Ajuda<br>Endereço http://localhost:8084/webmngr.jsp?netID=popv1<br>WebManager 2.0<br>Home<br>Sobre...<br>POP-PB-v1<br>Rede do POP-PB versao 1<br>Escolha a rede<br>Lista dos Equipamentos<br>RNP-2 / INTERNET<br>Roteador POP-PB<br>Servico Web Proxy<br>RPP<br>Intranet local |
| **6.1 Main network map for managing a small network** | **6.2 Detailed network map for a small network** |



**6.3 Detailed network map for Web proxy**



**6.4 Detailed network map for a switch**



Interface do Switch POP-PB.RNP.BR

Descrição do Hardware: Serial 0 / Switch IBM 8271

Status Operacional: 🔴

Tempo de Funcionamento: 3 days, 14 hours, 29 minutes, 54

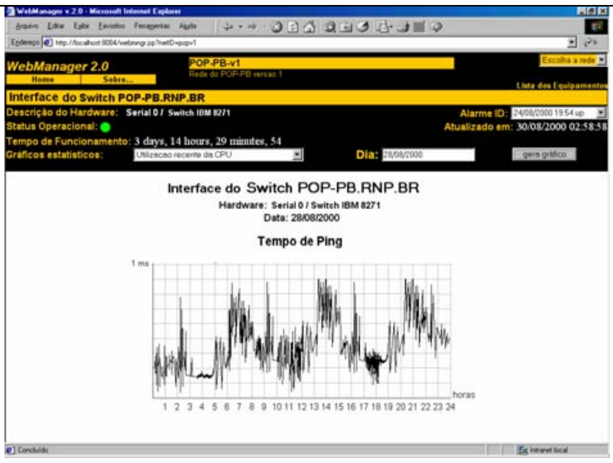
Gráficos estatisticos: Escolha a estatistica sobre o equipamento

Alarme ID: 24/08/2000 19:54 up

Atualizado em: 30/08/2000 02:53:29

Dia: 28/08/2000

**6.5 Detailed information about a network switch**



Interface do Switch POP-PB.RNP.BR

Descrição do Hardware: Serial 0 / Switch IBM 8271

Status Operacional: 🟢

Tempo de Funcionamento: 3 days, 14 hours, 29 minutes, 54

Gráficos estatisticos: Utilizacao recente da CPU

Alarme ID: 24/08/2000 19:54 up

Atualizado em: 30/08/2000 02:58:58

Dia: 28/08/2000

Interface do Switch POP-PB.RNP.BR

Hardware: Serial 0 / Switch IBM 8271

Data: 28/08/2000

**6.6 Statistical time lime graph for round-trip time**

Figure 6. Web Interface for managing a small network

### 4.2 Ease of use

We understand that we have not achieved the "big red button" simplicity we mentioned in the introduction, nor was this meant to happen in this version of the tool. That level of ease-of-use will be very difficult to attain. However, wmv2 is *much* easier to configure than wmv1, even though it is much more powerful. That is a step forward.

More important, we must verify that very easy configuration will eventually be achievable without changing the architecture. We believe this is possible. We will need to provide more components and an intelligent auto-discovery module that will not only discover devices and topology but automatically choose which components to include in the configuration, and what properties they should have. This would also have to include network maps. Although this is not a simple task, it is achievable and does not affect the architecture. On the contrary, the framework's highly modular architecture eases the task.

### 4.3 Scalability

Using a 300 MHz Pentium machine with 64 Mbytes running Linux and Java Development Kit 1.2.2, our tests indicate that 100 network elements can be polled in 5 seconds. Up to this number of elements, performance is strictly linear. Although we have not yet confirmed the saturation point where a linear performance behavior stops, we believe that our solution will easily allow the polling of thousands of elements in a 15-minute time frame, commonly used in network management, using standard, inexpensive, PC hardware.

### 4.4 Extensibility

The last major requirement is especially important since it is what will let us easily evolve toward our final goal: very easy-to-use tools. Since the goal is still distant, we need flexibility and extensibility. We have achieved tremendous success with this requirement. Any change we have thought of implementing is easy to merge into the framework. Consider how easy it is to implement the following extensions:

- Trouble ticketing: in order to interface our applications with a trouble ticketing system, all we need to do is implement a new notification bean that would consume events on the ALARMS Databus and interface to the trouble ticketing system.
- Non-SNMP agents. As an example, we currently are implementing a new Monitor bean that interfaces to an ATM switch through a telnet interface to gather information through "screen scraping". This is necessary since the desired information is not available in the switch's MIBs.
- Notification of alarms on network maps is currently done through a pull model (the JSP servlet calls a proxy view bean that obtains the information from the management station). We could easily implement a push-model notification by having a new notification bean, similar to the Mailer bean.

## 5 Future development

Although much remains to be done to achieve our final goals for WebManager, the tool is already useful and can, in fact, be employed to manage small networks. We intend to pursue further work in the following areas:

- *Ease of use*. We are still pursuing the "big red button" metaphor. This will involve the creation of many new components and, most important, auto-configuration algorithms. We will also probably need to design a user interface framework, since, in the current version, there is a lot of manual work involved in creating the JSP-based interface.
- *Service management*. We have concentrated our efforts in managing the network infrastructure. However, we believe that the framework is also applicable to manage services. We will validate this idea by building a full E-commerce management solution.
- *Distributed management*. As the tool improves, it may eventually be used to manage large networks (thousands of nodes). This situation leads us to examine a distributed Databus.

## 6 References

[**Apache 2000**] Apache Software Foundation, *Jakarta Project*, On-line: http://jakarta.apache.org/

[**Fayad 1998**] Fayad, M. E., D. C. Schmidt, Johnson, R. E., Building Application Frameworks - Object-Oriented Foundation of Framework Design, Wiley, 1999.

[**Gamma 1995**] Gamma, Helm, Johnson, Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[**Huntington-Lee, 1997**] Huntington-Lee, J. D. Case, *HP's OpenView*, McGraw Hill, 1997.

[**Roberts 1997**] Roberts, D., R. Johnson, *Evolving Frameworks: A Pattern Language for Developing Frameworks*, Addison-Wesley, 1997.

[**Sauvé, 1999**] Sauvé, J. P., *WebManager: A Web-Based Network Management Application*. LANOMS – Latin American Network Operations and Management Symposium, Rio de Janeiro, Brazil, December 1999.

[**Sun 1999**] Sun Microsystems Inc., On-line: http://www.javasoft.com/beans/infobus.

[**Szyperski 1999**] Szyperski, C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley, 1999.