WebManager: A Web-Based Network Management Application

Jacques Philippe Sauvé
Departamento de Sistemas e Computação
Centro de Ciências e Tecnologia - Universidade Federal da Paraíba
Av. Aprígio Veloso, 882 - Bodocongó
58109-970 - Campina Grande - PB - Brazil
Email: jacques@dsc.ufpb.br

Abstract

The implementation of a three-tier, Web-based network management application called WebManager is discussed. It is implemented in a UNIX environment, accesses management data through SNMP and is written in *perl* and C. Its main features include a Web-based interface, hierarchical network navigation maps, display of graphical statistics for any *Management Information Base* (MIB) variable, automatic color status indications for devices and sub-networks, full configurability and automatic alarms. This tool is currently being used in two production environments to manage networks with hundreds of nodes.

1 Introduction

Traditional network management solutions based on centralized *Network Management Stations* (NMS), running on UNIX environments and using the Motif *Graphical User Interface* (GUI) suffer from many problems [1,2,3]. One of these problems is their high cost which runs to the tens of thousands of dollars for anything but very small-scale networks due to the cost of the management platform added to the cost of the management applications. A second and more important problem is that of mobility: the management software is only available through the GUI on the NMS hardware console. This lack of user mobility seriously hinders management tasks: network managers want to access management applications from anywhere on the network, not just the NMS console.

Web technology is being used to solve these two problems. The use of a Web browser as a universal client to access management applications completely eliminates the mobility issue and alleviates the costs incurred in the final solutions. Web-based network management solutions can be grouped in four classes: two-tier solutions whereby an HTTP server is embedded in network devices as a *telnet* replacement [4], three-tier solutions in which the applications' GUI is replaced with a Web browser while maintaining the NMS [5], and, finally, two recent initiatives aiming to replace the SNMP-based Internet Standard Management Framework with an object-oriented data model and a new management protocol. These efforts are called *Web-Based Enterprise Management* (WBEM) [6] and *Java Management Extensions* (JMX) [7].

This paper describes the implementation of a three-tier Web-based network management application called WebManager. Section 2 presents the motivation behind the work, the objectives and requirements of the solution. Section 3 gives a tour of the final product. We believe the rest of the presentation will be clearer after the reader has had a chance to see the management application in action. Section 4 provides implementation details and constitutes the main part of the paper. Section 5 critically evaluates WebManager. Finally, section 6 concludes by indicating the ways in which the solution is evolving.

2 Motivation, Objectives and Requirements

2.1 Motivation

Four motives led us to develop WebManager. In the first place, the author is currently investigating Web-Based Management as an alternative to traditional management techniques. We felt that implementing a tool would lead to a better understanding of the issues involved.

Secondly, we aim to use a future version of the tool as a platform for implementing management applications as part of our research effort in fault and performance management of computer networks. The current version of the tool is not yet usable in this sense since it is a management application and not a platform. However, as explained further on, version 3 will be a full management platform.

Third, our institution needs tools to implement production network management applications. Before our solution was concluded, we were using a traditional, expensive, platform-based network solution to manage our campus network. However, severe budget restrictions did not allow us to expand the solution, buy and integrate new applications, deploy new management consoles, etc. We therefore intended to build a production tool that could substitute this platform. While it is true that the cost of the manpower involved can easily surpass the high cost of expanding the platform, one has to keep in mind that, at a university, human resources are often more readily available than hard cash. We understand that this may not be the case at other institutions.

Finally, we wanted to solve the mobility problem exhibited by the traditional approach. A Web-based solution where the user interface is a Web browser is the natural way of solving this problem, as can be seen by the number of intranets being deployed world-wide for all types of applications.

2.2 Objectives

Our main objective was to implement a Web-based network management tool in three stages. The **first** stage would be a basic management application used to try out some ideas and rapidly become a production tool at our institution. This stage has now been successfully completed and has met all expectations, goals and requirements. The **second** stage, currently under development, will expand the first tool to make it much easier to configure, introduce better graphical representations of the management data, make it cross-platform so the NMS runs on a Windows-NT server, and package it as a shareware product to be offered on the Internet. The **third** stage, currently in the requirements phase of development, will finally transform the product into a management *platform* that will allow one to plug-in management applications especially developed for it.

This paper describes the result of the first stage of work which has been satisfactorily concluded.

2.3 Requirements

The first stage, baptized *WebManager Version 1*, was implemented with the following requirements in mind:

1. The solution should be a management *application* and not a platform. In other words, it is *not* required that separate applications be able to plug themselves into the final result.
2. The solution must use the Internet Standard Management Framework and use the *Simple Network Management Protocol* (SNMP), versions 1 and 2.
3. A Web browser must be the only tool available to access the application's interface.

4. The interface must allow for hierarchical network navigation through position-sensitive maps. In other words, a high-level network map should show the whole network; clicking on a part of the network should provide a new image for that part of the network, and so on.
5. The network maps need to show the status of each part of the network through an appropriate visual mechanism such as a color code. This allows for a quick view of the presence and location of any trouble spots in the network.
6. The application should basically allow *Management Information Base* (MIB) browsing with access to historical statistics through daily graphs. Only monitoring should be allowed; in other words, due to the weak security of SNMP versions 1 and 2, no active control of the network should be done.
7. Graphs presented in the interface can be as much as one hour old. However, there has to be a mechanism whereby up-to-date statistics can be obtained with a response time of at most 15 seconds for small networks.
8. Configurable alarms need to be provided. Alarms should be based on any mechanism able to detect the crossing of thresholds in the value of MIB variables. Furthermore, the use of arbitrarily complex expressions of MIB variables to generate alarms is desirable. An arbitrary command (such as sending mail) should be configurable as an alarm's action.
9. Configuration of the tool can be done manually with a text editor. However, the configurability must be very high. The whole solution must be data-driven, with no topological information or visual element coded into the programs. This includes HTML pages, graphs, network maps, etc.

3 Using WebManager Version 1

This section leads the reader through a quick tour of the final solution. We believe this to be the most efficient way of describing the tool before dealing with its implementation in the next section. The advantage is that this approach is much less tedious than giving a full or even partial product specification.

Pointing the browser at the starting *Uniform Resource Locator* (URL) yields the image shown in Figure 1. This figure shows a screen shot of the Web browser. The main part of the screen is a network map showing the whole network composed of four sub-networks, a firewall, a router and the internet. Each part of he network has a small dot whose color indicates the operational status of that part of the network. The small hub doesn't have such a dot as it is not a manageable device (it doesn't speak SNMP). The top part of the screen exhibits two links pointing to summary reports for the network. We will have more to say about these reports later. Finally, the bottom part of the screen allows the operator to purge old statistical graphs and old data from the system.

Figure 1 shows, through a red dot, that the Main Office part of the network has a problem. Since the network map is position-sensitive, clicking on the Main Office displays the screen shown in Figure 2. This screen shows the Main Office part of the network and shows that there is a problem with the connection to the *Remote Access Server* (RAS). Clicking on the Main Office Stackable Hub yields Figure 3 which shows that the fifth repeater group has a problem.

Finally, clicking on group 5 leads to the screen of Figure 4 showing that port 19 is *down*. It should be clear from these screens that network device status has actually propagated up the hierarchy so that, for example, the top image (Figure 1) shows that network problems exist at lower levels.

After navigating through network maps, one finally reaches the "leaves" of the network hierarchy: network devices and interfaces. By clicking on the image representing the device (or the small dot indicating the status of a link), a screen similar to Figure 5 is shown. The screen shown here is for a 256 Kbps Wide Area Network access link. The screen has two frames: the top one

shows the hardware name, operational status, alarms, etc. The second part of the screen shows several thumbnail graphs of statistics for several days (usually the last 5 days). Scroll bars (not shown here) allow one to access all statistics for all available days. Finally, a full graph may be exhibited by clicking on any thumbnail. For example, Figure 6 shows the percentage of collisions obtained from a Remote Monitoring (RMON) probe on an Ethernet. The graph clearly indicates that the network should be segmented, since the percentage of collisions is well above the recommended maximum of 5%.

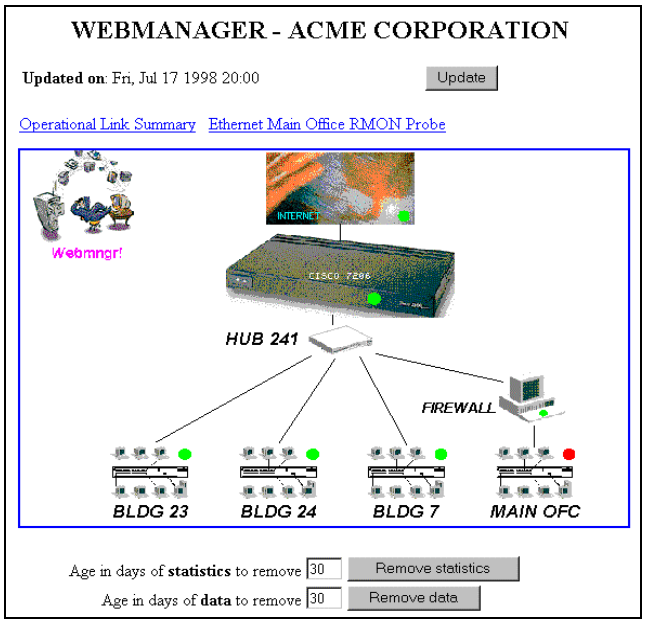

Figure 1: The Whole Network as Seen by WebManager

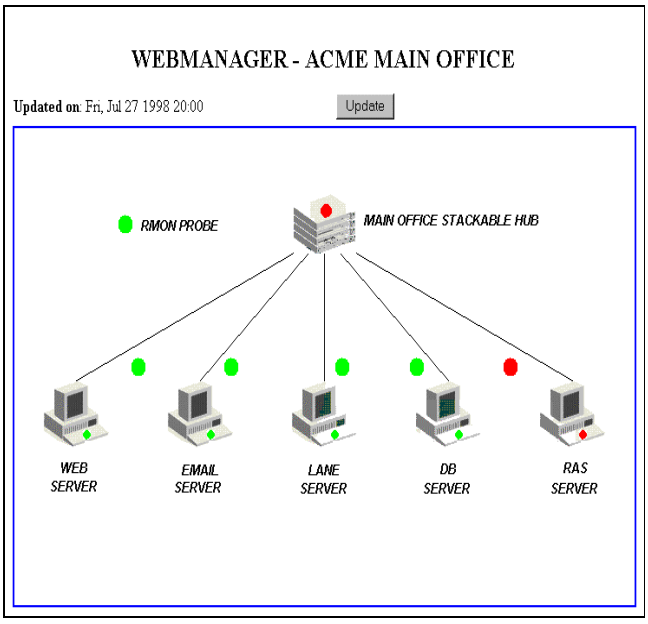

Figure 2: Main Office Part of The Network

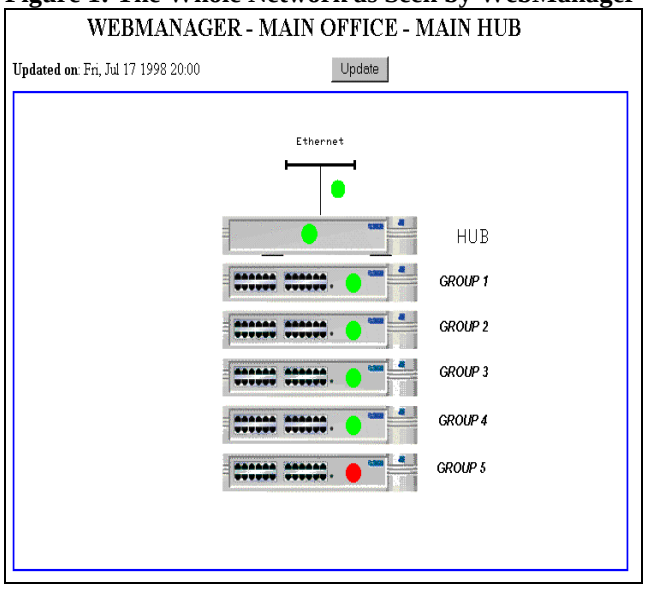

Figure 3: Main Office - Main Stackable Hub

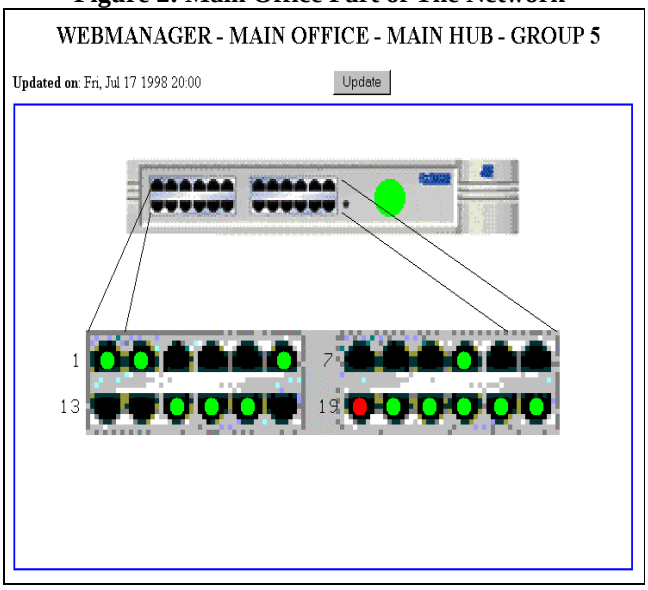

Figure 4: Repeater Group 5

Several of the graphs show Update and other buttons that a activate *Common Gateway Interface* (CGI) program used to update all values, images and graphs shown on the screens. Another important function which is not shown on the screens is that any alarm detected causes an action to be executed. For example, a mail message could be sent to a network operator.

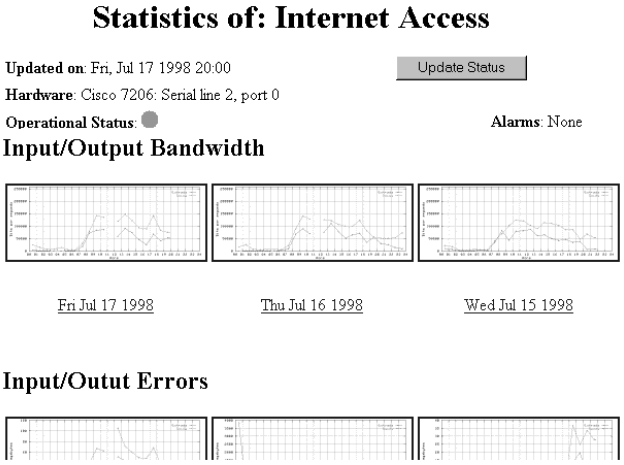

Fig. 5: Thumbnail Statistical Graphs of a WAN Link

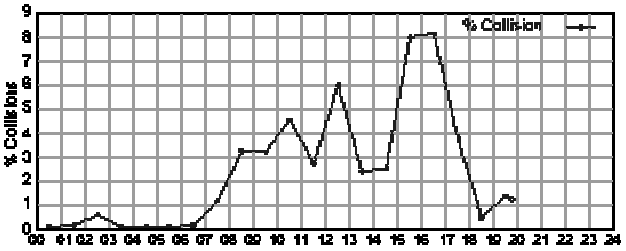

Fig. 6: Percentage of Collisions from an RMON Probe

4 Implementation Details

This section discusses certain details of the WebManager implementation. Specifically, we cover the architecture, the name space for managed objects, network device monitoring, and both offline and online HTML page construction.

4.1 WebManager Architecture

WebManager's architecture is a typical three-tier Network Management Station (NMS) [5] as seen in Figure 7. The first tier consists of network devices such as routers, hubs, switches, etc. containing SNMP agents. The second tier is the NMS. It obtains data from tier one and stores it in a database. At appropriate times, this data is converted to HTML format (as well as GIF and MAP files) which allows the browser in the third tier to access the management information.

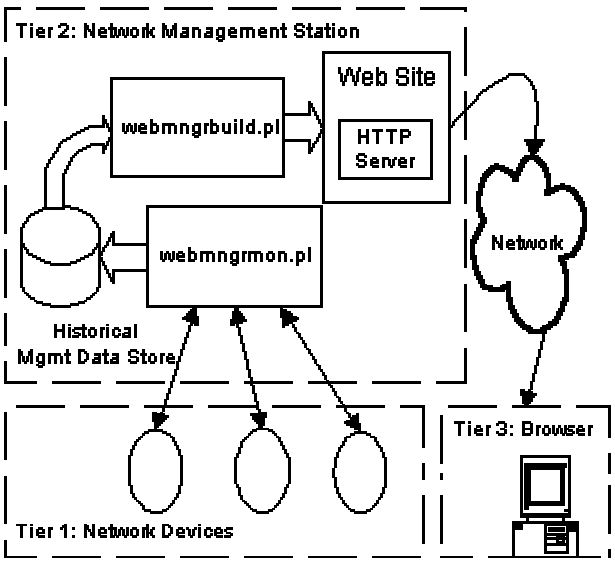

Figure 7: WebManager Architecture

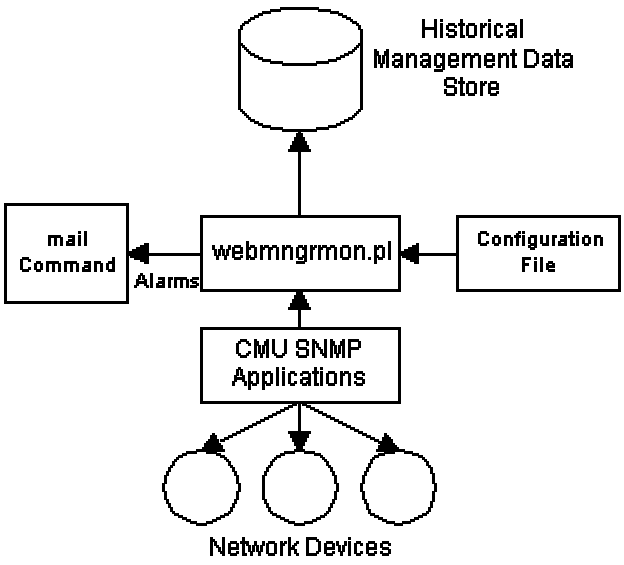

Figure 8: Managed Object Monitoring

The various parts of Figure 7 are described in more detail further on. A brief description follows. The program called `webmngrmon.pl` (WebManager Monitor) runs permanently in the background and periodically polls the network devices using SNMP and stores the MIB variable

values in a database. It also checks whether alarms have occurred (variable values may have crossed thresholds, for example) and takes appropriate actions when they do. The program called `webmngrbuild.pl` (WebManager Builder), on the other hand runs once an hour (through the UNIX `cron` facility) and converts the database values into appropriate HTML pages, GIF images (network maps, thumbnails and full graphs), and HTML MAP files (for position-sensitive GIFs).

For the sake of simplicity, Figure 7 does not show the CGI program which is also part of the NMS. A CGI program allows one to include some interactivity in the user interface. It is through the CGI that images, status values, graphs, etc. may be updated at any time through the operator's browser. The CGI works by simply calling `webmngrmon.pl` and `webmngrbuild.pl`. The fact that the CGI program calls these two programs is the only reason for having all programs from the second tier running on the same machine (the NMS). Otherwise, since communication between `webmngrmon.pl` and `webmngrbuild.pl` and the rest of the system is through files, these programs could have run on separate machines connected through a distributed file system.

4.2 Name Space for Managed Objects

Two requirements from section 2.3 lead to the adoption of a hierarchical structure for naming the managed objects. These are the need to offer hierarchical network navigation through maps and the need to propagate status information (the colored dots) along this hierarchy. For this reason, all managed objects are organized in a hierarchy. Using the "!" as a hierarchy separator, section 3 has navigated to the following HTML pages: `!.html` (the root of the hierarchy containing all managed objects), `!main.html` (containing all objects from the main office), `!main!mainhub.html` (containing all objects below the main office's main hub), `!main!mainhub!group5.html` (containing everything in the fifth repeater group of this hub), `!main!mainhub!group5!port19.html` (a leaf object in the hierarchy which was not visited in the tour), and so on.

As said above, this hierarchy allows the proper propagation of status information. When port 19 goes down (its status becomes red), a red dot will appear on port 19 and will be propagated up to also appear as the status of the aggregate objects `!main!mainhub!group5`, `!main!mainhub`, and `!main`. Furthermore, this hierarchy is used when clicking on the update button on any HTML page. The update must include all objects of the hierarchy situated on or below the object under consideration. For example, clicking on the update button of Figure 3 would force `webmngrmon.pl` to poll all ports on all repeater groups for the hub, the hub itself and the Ethernet network interface. `webmngrbuild.pl` would then be used to update all HTML pages, maps and images for these objects and all network maps above these objects in the hierarchy (to propagate visual status information).

4.3 Managed Object Monitoring

This section describes the `webmngrmon.pl` program and how it interacts with other components of the architecture.

4.3.1 General comments

`webmngrmon.pl` monitors network devices[1] through the SNMP protocol at configured intervals, usually around 5 to 15 minutes for critical devices and 60 minutes for less critical ones.

[1] Actually, anything with an SNMP agent and MIB support can be monitored, including HTTP servers, DNS servers, database servers, printing systems, UPS, etc.

The values obtained from the devices are stored in a database. Observe that `webmngrmon.pl` does not interface directly with the user and does not produce any GUI output. A second task performed by `webmngrmon.pl` is to check for alarm conditions and perform suitable actions when such alarms are set off. `webmngrmon.pl` is written in *perl*. It is started on the NMS at boot time and runs in an endless loop. Figure 8 shows details of its operation and interaction with other components of the architecture.

4.3.2 Configuration file

All of WebManager's configuration is contained in a single text file. This file is edited manually in this version of the product. This means that the devices to be monitored, the MIB variables to poll for each device, etc. are all statically defined in this file. It is thus not currently possible to use WebManager as a dynamic MIB browser, in the sense of dynamically choosing which MIB variable to poll for, with what frequency, etc. The configuration file will not be fully described here. The following list of items contained in the file and related to the monitoring task will be sufficient to give a general idea.

- Installation information such as file and directory names.
- Equipment classes such as Cisco routers, Ethernet interfaces, Repeater Groups, etc. along with the MIB variables to be monitored for each class of equipment.
- A description of physical devices in the network topology.
- A description of logical devices along with their criticality, polling frequency, and corresponding physical devices. Observe that separating physical and logical devices allows one to inquire about, say, the "Internet Access Link" without worrying about the physical router port used. If the router port is changed, the operator would still ask about the "Internet Access Link" and wouldn't be aware of the port change.
- A description of alarm levels, actions and specific alarms to test for.

4.3.3 SNMP commands

When it needs to access the network devices using SNMP, `webmngrmon.pl` uses the CMU SNMP package [8]. This package includes the `snmpget` command which retrieves MIB variables using SNMP's "get" operator and the `snmpwalk` command which does the same but using SNMP's "get-next" operator. `snmpget` is used to get simple variables while `snmpwalk` is faster when retrieving whole MIB tables. This is be the case, for instance, when retrieving information about *all* of a router's interfaces with a single command.

4.3.4 The historical management data store

After retrieving MIB variables using the CMU SNMP package, `webmngrmon.pl` stores the results in a database for further processing. For the moment, we are using simple text files to hold this data, since the networks being managed are still reasonably small and since storing the information as text allows easy parsing in the *perl* language. Separate files are used to hold the data for each separate day of monitoring. Each record contains a time stamp as well as the values of the MIB variables polled. Some pseudo-MIB variables are also polled and stored in the database. This includes, for example, the "ping" variable indicating the round trip time to the managed object in milliseconds, the "snmpup" variable indicating whether the SNMP agent is responding, and so on.

Values obtained from the network devices are stored in the database once every hour. For some MIB variables, the values obtained are averaged over the last hour before being stored. This would be the case for, say, measures of free memory, values for ping, etc. For other MIB variables (uptime or cumulative error counts, for example), the last value obtained is stored. In other cases, both an

average and the last value are important. Consider the value of a communication link's operational status. The average value indicates the fraction of time the link was up over the past hour while the last value indicates the last known status: both measures are important.

4.3.5 CGI support

Recall that `webmngrmon.pl` runs permanently in the background and periodically polls devices. Let us call this the "normal" execution of `webmngrmon.pl`. But we also know that the CGI program must call `webmngrmon.pl` to update values at the operator's request. This is done by having two special arguments accepted by `webmngrmon.pl` and used when it is called by CGI. The first is a flag (`-u`) which indicates that `webmngrmon.pl` should run only once and exit. The second is a list of arguments saying which particular devices should be polled. In this way, `webmngrmon.pl` may be executed through the CGI while the normal execution goes on. The normal execution will poll all devices regularly while the second execution (through the CGI) will poll only those few devices necessary to update the current HTML page and then exit.

4.3.6 Support for alarms

Since the first version of WebManager does not support asynchronous SNMP traps, alarms are detected by `webmngrmon.pl` through the normal synchronous polling process. Alarms are configurable in two ways. First, several severity levels with associated actions may be given in the configuration file. This is shown in the following example configuration description taken from the configuration file.

```
ALARMS
  DEFALARM Critical "mail netop@acme.com"
  DEFALARM Normal   "cat >>/web/webmngr/log"
  DEFALARM Ignore   "cat >/dev/null"
  ALARM    !main!mainrouter
    ALARMTYPE  Critical
    ALARMTEST  &StatusDown("!main!mainrouter")
    ALARMMSG   $ObjectName{"!main!mainrouter"} . "is down"
ENDALARMS
```

This information defines three alarm levels and the associated action. For example, when a critical alarm occurs, a mail message will be sent to the `netop` user. The example also defines a single alarm associated with the logical object `!main!mainrouter`. The `ALARMTEST` line indicates what boolean condition must be met to set the alarm off. In this case a *perl* function (`StatusDown`) is called. This function, is part of `webmngrmon.pl` and checks to see whether the status of the object has gone through a transition from "up" to "down". The mail message received by the network operator is defined in the `ALARMMSG` line. `ObjectName` is a *perl* array containing the full name of all objects. It is important to observe that the configuration file contains *perl* code in certain choice locations. For example, we decided that the most general way of handling alarm conditions would be to embed any *perl* expression in the configuration file. These *perl* expressions are evaluated at run time and can access any of the variables and functions in the `webmngrmon.pl` program. This is exactly what can be seen in the above example.

4.4 HTML Page Construction

This section describes the `webmngrbuild.pl` program and how it interacts with other components in the architecture.

4.4.1 General comments

Up to this point in the discussion, we may assume that `webmngrmon.pl` has loaded the database with historical values for MIB variables (all entries in the database are timestamped). The second major component of the WebManager architecture is `webmngrbuild.pl` which is responsible for transforming this data into a Web interface using HTML, image maps and GIF files representing network maps as well as thumbnails and full graphs. A browser is then used to access the resulting files. The `webmngrbuild.pl` program is written in *perl* and runs on the NMS once an hour through the UNIX *cron* facility. It thus produces its results offline. Figure 9 shows a detailed view of the `webmngrbuild.pl` component and its interfaces inside WebManager.

Three types of information must be produced by `webmngrbuild.pl` and correctly installed as Web pages in the management Web site. They are: HTML pages, network maps and statistical graphs. We consider how to produce each of these in turn.

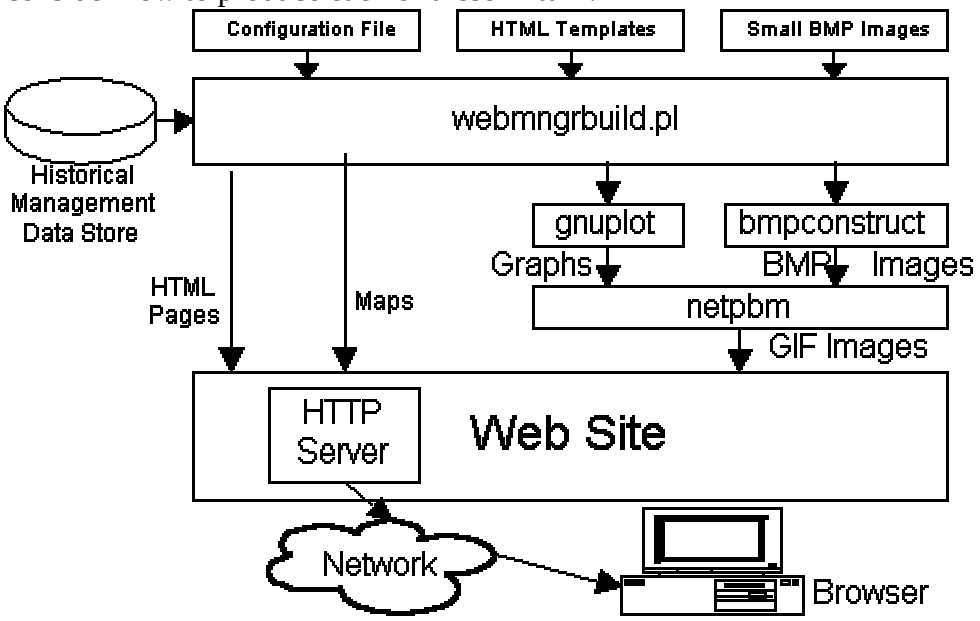

Figure 9: Construction of HTML Pages and Associated Information

4.4.2 Producing HTML pages

The HTML pages accessed by a browser, a few examples of which were seen in section 3, are *not* static pages. For this reason, HTML *templates* are used to produce the pages. For example, from hour to hour, the times given on the pages will change. If the status of a device changes, the GIF file indicating the status may change from, say, "greendot.gif" to "reddot.gif". Furthermore, from day to day, the daily graphs exhibited will change completely. As a result, the HTML pages exhibited are constantly changing. A stronger reason for using templates is that several devices may be very similar (consider two routers of the same model) and would have their pages produced from a single template. Using templates was dictated by the requirement to have a data-driven solution. The data comes from the configuration files, template files and the data base containing management data values. `webmngrbuild.pl` processes all this data and produces the final HTML pages.

In order to generate the variable data inside HTML pages, *perl* code was embedded in the HTML template files. In other words, by processing and expanding the HTML file,

`webmngrbuild.pl` will actually call any appropriate *perl* functions used to prepare whatever visual object may be needed by the HTML file. This is a very powerful mechanism and has drastically simplified the development effort and made maintenance an order of magnitude simpler.

A measure of this technique's success may be judged when one considers that, in a production system using WebManager, a mere 15 templates are used to generate the more than 10000 HTML files present on the Web site to manage a 500-node network with 200 managed objects.

4.4.3 Producing network maps

We have said above that the `BuildMap` function is responsible for producing network maps such as those shown in Figures 1 to 4. This section describes how this important function works. A *network map* is made up of a GIF file containing the image and an associated MAP file indicating where the hotspots are located on this image and what URLs are associated with those hotspots. Let us see how each of these files is built, in turn.

In order to build a GIF file, a BMP (bit map) file is first constructed. It is then converted to the GIF format using the *netpbm* package [9]. The BMP image is constructed through a program called `bmpconstruct`, written in C and called by `webmngrbuild.pl` according to instructions contained in the configuration file. This bit map constructor program basically knows how to build a larger BMP file from small ones.

The small BMP images may be associated with managed objects or not: in Figure 1, all small images are thus associated except for the "WebManager" image (added for the pleasing effect) and the hub image (this hub is not SNMP-manageable). When the images *do* represent managed objects, the `bmpconstruct` program may be told to change one color to another in order to reflect the device's status. For example, the same small BMP image was used for "BLDG 7" and "MAIN OFC"; but since the MAIN OFFICE has a red status to indicate a problem, `bmpconstruct` was told to add the small BMP and change all green pixels to red. This is how the network maps are made to dynamically reflect network status. Finally, `bmpconstruct` also can add visual sugar to the image. This includes text (in several sizes and colors), lines, boxes, etc. Finally, after the BMP file is ready, commands from the *netpbm* package are called to convert to GIF.

The second file needed for a network map is the MAP file indicating rectangular areas and associated URLs for all hotspots on the image. `webmngrbuild.pl` builds this file as it sets up the call to `bmpconstruct`.

We believe that the approach taken shows the power of the UNIX tools approach. Many small but well-thought-out commands are used in conjunction to produce the desired result. More importantly, all this image construction is done automatically, without human intervention. Finally note that everything described here is data driven; the results can be changed by editing only the configuration file and HTML templates.

4.4.4 Producing statistical graphs

Statistical graphs, shown as thumbnails in Figure 5 and as a full graph in Figure 6, are one of the most important aspects of WebManager's interface. These graphs can give a clear idea of trends, can be used for capacity planning and may be used to find recurrent problems in the network. WebManager can be used to produce daily graphs for any MIB variable such as uptime, packet errors, bandwidth consumption, resource utilization, etc. In order to produce graphs such as the one shown in Figure 6, `webmngrbuild.pl` extracts the appropriate data from the database and produces a *data points file* with x-y values for the desired curves. The curve drawing program *gnuplot* [10], called by `webmngrbuild.pl`, reads the data points and produces a *Portable*

PixMap representation of the graph. The *netpbm* package then converts this image to the GIF format.

Once more, the power of the tools approach is apparent. Many commands (`gnuplot`, `ppmtogif`, `giftopnm`, `bmptoppm`, `pnmscale`, `ppmquant`) are combined to yield the desired effect and do not require human intervention to produce the results.

4.4.5 Online HTML page construction

The Web interface includes `Update` buttons that may be used to update all information shown on the HTML pages, network maps and graphs. A CGI program, called `webmngr.pl.cgi`, is used to respond to these requests. It is also written in *perl* and basically works by calling `webmngrmon.pl` to update the database and `webmngrbuild.pl` to update the visual information. Updating may be applied to a single object (say a router) or to a whole section of the network (the whole Main Office, say).

5 A Critical Evaluation of WebManager

We feel that it is important to tally up our experience with WebManager and point out its strong points as well as its defects. Such an analysis will allow others to better learn from our work.

5.1 WebManager's Strong Points

On the implementation side, writing the 4000 lines of C/*perl*/HTML code was very fast. This is due to the use of the *perl* language, which is excellent in every way in dealing with applications involving text processing and this is what most of the work actually is. Using HTML template expansion with embedded *perl* code was also beneficial to the effort. Architecturally, the `webmngrbuild.pl` program is nothing more than a management application and it is relatively straightforward to write other applications.

WebManager has been used to manage two networks so far totaling 500 nodes and 100 nodes, about half of which are managed objects. It has proven useful as a management tool, especially with respect to graphical MIB browsing. We have already discovered many weaknesses of these two networks through the tool, as indicated by Figure 6 which was taken from actual operation. By spending some time in properly defining alarm conditions (which WebManager makes very general), reasonably automatic network management can be achieved, especially with respect to fault identification and performance management. The device configuration aspects of WebManager are non-existent but the HTML pages can easily include URLs to access embedded HTTP servers in the devices and thus access configuration pages provided by device manufacturers. In such a case, using *telnet* becomes a thing of the past.

With respect to speed, the pre-calculation of HTML pages, network maps and images allows very responsive browsing although the load on the NMS to generate this information is heavy for larger networks.

5.2 WebManager's Weak Points

From an implementation perspective, we made a mistake in imposing a hierarchical structure on the managed objects. A general graph rather than a tree would have been more useful. The lack of a MIB compiler forces the developer to provide more configuration information than is necessary, but this was a imposed by the short time span available to develop the tool. We feel that the statistical graphs part of WebManager is still too weak since it does not easily allow one to graph curves other than straight MIB variables. More general calculations should have been

possible. The method used to embed *perl* code in HTML templates should be redone and should use more general mechanisms such as extensible tags rather than *magic cookies*. Also, Java applets would have given us a more responsive interface although this also was precluded by time constraints. Finally, no aspects of security were considered while drawing up the requirements and the final solution suffers from this omission.

As far as usefulness for network management is concerned, WebManager suffers from a major fault: configuring the product for a particular network topology is extremely time-consuming. As an example, the configuration file for one our small networks is 2500 lines long. The process would be much improved by using auto-discovery tools and a graphical network map editor such as described in [11]. Better graphs should be available, especially when one wishes to combine statistics from several devices on the same graph for comparison purposes. Another weak point is that alarm correlation is not performed by WebManager. As an example, should the MAIN OFFICE HUB of Figure 2 go down, tens of alarms would go off at the same time with no clear indication of the common cause. Finally, no support for SNMP traps was provided and this could be used advantageously to minimize the normal polling traffic.

An evaluation of the speed of the tool indicates that updating the status for the whole network through he CGI may take tens of minutes for a medium-size network. This could be substantially minimized if threads could be used. As it stands, WebManager does not scale well, although this is mainly due to the SNMP-based framework which imposes the concept of a centralized NMS.

6 Conclusions

WebManager is a three-tier, Web-enabled and SNMP-based network management tool. It is currently being used to successfully manage two networks with hundreds of nodes. We are currently working on a second version of the tool. Our main objective is to build-in enough quality to make it useful to third parties. If we are successful, we plan to offer WebManager as shareware on the Internet. Building in the proper amount of quality is a demanding challenge. As a first step, the tool must be extremely easy to use, especially to install and configure, and we are still quite far from that goal. Furthermore, the tool will need to have a more dynamic interface and we plan to achieve this by the judicious use of Java applets. In fact, most of the negative points mentioned in section 5 will be removed. Finally, we intend to make WebManager platform independent, allowing it to run on various flavors of UNIX and on Windows NT. Version 3 of the product is currently in the requirements stage. It will be component-based allowing applications to be built visually.

References

[1] R.L. Ptak, “Managing Complexity Trends and Issues in Distributed Management”, *On-line*, URL http://www.summitonline.com/netmanage/papers/brown2.html.

[2] C. T. Corcoran, “Managing Network Ills Network Managers Tackle Management-Tools Deficiencies With Web Technologies”, *InfoWorld*, 18(42), 1996.

[3] M. Smith, “Enterprise Management Glue”, *On-line*, URL http://kinetworks.ki.com/WBEM/emglue.html.

[4] D. Hyde, “The New Paradigm for Network Management”, *On-line*, URL http://www.3com.com/technology/tech_net/white_papers/500627.html.

[5] A. K. Larsen, “The Next Web Wave: Network Management”, *Data Communications*, 25(01), 1996.

[6] Desktop Management Task Force, DMTF, *On-line*, URL http://www.dmtf.org.

[7] Sun, Java Management Extensions White Paper, *On-line*, URL http://java.sun.com/products/JavaManagement

[8] *Online*, URL http://www.net.cmu.edu/projects/snmp.

[9] *Online*, URL ftp://wuarchive/wustl/edu/graphics/graphics/packages/NetPBM.

[10] *Online*, Newsgroup comp.graphics.apps.gnuplot.

[11] J. Schönwälder, H. Langendörfer, "How To Keep Track of Your Network Configuration", *LISA VII*, Monterey (California), November 1993.