A Generic Web-Based Network Management Application

Yehuda Afek* Yoram Cohen†
Computer Science Department,
Tel-Aviv University, Israel 69978.

Abstract

A new generic, web based approach to network management is introduced. In traditional network management each management application is designed and implemented from scratch by its vendor in a specific proprietary way that depends on the vendors private MIB and implementation. Here we suggest a standard platform for the design of management applications. In the same way that the Web and HTML are a standard platform for people to design remote graphic displays and interfaces, our platform together with a new language called NMML (Network Management Markup Language) suggests an easy way to build and design complex and yet flexible management applications. The new approach greatly simplifies the development of network management applications, and paves the road for making a generic network management application able of managing any communication device. This approach requires, at most, the addition of few MIB variables in the device's agent.

*Telephone: +972-3-6408952,Fax: +972-3-6409357, e-mail: afek@math.tau.ac.il
†Telephone: +972-3-6459468,Fax: +972-3-6472675, e-mail: yoramco@math.tau.ac.il and yoramco@radmail.rad.co.il

1 Introduction

The process of developing management applications for new networking devices is currently expensive, time consuming and delays the time products time to market. The main reason for the high cost of this process is the lack of standardization and the use of many proprietary data structures and code. Three major ingredients mix together in each management application: (1) The SNMP protocol (Simple Network Management Protocol) [CFSD90, Ros91b, Ros94] is used for communication between the management application and the managed devices. (2) The set of MIB variables used by the management application to interact with the different devices (since each application uses mostly the variables defined in the private MIB of the device). [Per93, Ros91a] (3) The user interface, embedded in each management application (hence the details of the front-end user interface) and the way it uses the MIB variables, inside its code. Because of these characteristics, management applications for different devices look different, and entail a huge investment in the development of each. This fact, together with the fact that for each manageable device a new complementary management application must be developed causes vendors to spend a large amount of resources in the development of new management applications.

In this paper we introduce a new approach to the development of network management applications that eliminates most of the tedious work, and reduces development costs and time. We suggest a standard and *generic* network management application that manages all communication devices, using a small extension in each manageable device. Our suggestion has three major components: a standard language for describing management applications, a generic GUI (graphics user interface) management application which is like a browser into manageable devices, and a description in the above language of specific management application of the manageable device, which we call *the device_control_page*. The device_control_page that includes the specification of how the network management application interacts with the user and how it uses the MIB variables, may reside either in the agent or in a public location (e.g., on the web) instead of in the application. The generic management application would read this page from the agent, and build the proper management application for the device, on the fly. The generic management application would find a pointer to the device_control_page in an agreed upon form and location, in the agent.

The device_control_page may be implemented using a new MIB, or using a file that resides either in the agent or in the manager workstation, or in a third location such as a World Wide Web (WWW) site or diskette.

In our approach vendors implement only the device_control_page using a standard language, and place it in their agents, relieving them from the need to develop a management application for each new device. We strive of course, for an extension that is easy to implement, and efficient in terms of space and computing-power requirements. Thus the task of managing communication devices would become an easy step in the process of developing new manageable communication devices.

2 Management applications and the problem

A network management application is software that network managers use to manage any manageable device. It is a sophisticated browser to the agent MIB, in the same way that WWW browsers, like Mosaic and Netscape, are sophisticated browsers to the data found in HTTP servers.

The major parts of each management application are, user interface, device interface, and detailed description of the managed device(s). The device interface is an SNMP layer and is the same in all management applications. However, the user interface and the device description (including the MIB) are different from device to device. Moreover, both the user interface and the device MIB strongly influence the design and development of the network management application that manages the device.

The user interface part of management applications employs four different means of interaction with the user:

1. A Graphical view of the managed device including its physical and logical objects.
2. A Menu system enabling the invocation of desired operations.
3. Dialogs for viewing and changing the device parameters.
4. Lists for viewing and changing parameters of the device which are tabloid in nature.

The way each of the four means of user interfacing is used is strongly correlated to the structure of the device MIB. Therefore, the strong influence of the MIB structure on the management application design and behavior.

In developing a communication device vendors go through the following four steps: First, the device itself is developed, second the corresponding private MIB is defined, third the agent that supports the MIBs is implemented and finally the management application is written. Usually the forth step may not begin before late in the process and thus delaying the time the device meets the market.

2.1 The problem

These facts cause management applications for each device to be different, and entail a lot of tedious work to be done again and again for the development of different management applications. Together with the fact that for each manageable device a complementary management application is developed, it makes vendors spend a lot of resources in this area. It also prolongs the development time of devices, thus delaying their availability in the market.

3 The generic web based approach

Our solution suggests a use of a *generic* network management application to manages all communication devices. The generic application would find the description of each device's MIB and user-interface in a special file stored either in the device agent or in an alternative location. A newly suggested standard language is used to describe all of these in that file. The description for each device is called the device_control_page.

Each communication device holds a different device_control_page. The content of this device_control_page tells the generic application how the device wants its management application to look like. This is the heart of the idea and its uniqueness. The device itself, *and not the management application*, determines how its management application looks and behaves.

This is similar to the role HTML files have in the WWW world. In the same way that standard HTML files are understood by different WWW browsers, this description would have a standard form and will be understood by any generic management application.

The generic management application is a GUI (Graphic User Interface) application containing menus that invoke operations, dialogs, and lists. The device_control_page must include, in a standard way:

- A description of the graphical views of the device.
- A description of the menu system to be offered by the application. This includes the structure of the menus, the names of the menu items and the action to be invoked by each menu item.
- A description of the MIB variables to be set in case of a SET action.
- A description of the MIB variables to appear in the dialogs to be invoked, and their attributes.
- A description of the MIB variables to appear in the lists invoked, and their attributes.

The generic application would read the device_control_page from the agent, and build the management application on the fly, in the same way WWW browsers build the screens that appear in their window, on the fly, according to the corresponding HTML file.

The device_control_page is in an ASCII Hyper-Text-like form. It may be stored in the agent in the form of a tabular MIB variable of type STRING, which the manager would download using the SNMP protocol. Alternatively, a standard MIB variable (likely to be system.sysDescr) may be defined holding a URL that points to the location of the device_control_page on the intranet or Internet. The ASCII form of the device_control_page enables us to easily extend the language in which it is written and thus enables extending the capabilities of the generic application. In addition, space consuming elements in the device_control_page (like images of the device and its elements) may be referred to using URLs, instead of being held inside the agent.

3.1 Implementation

There are many ways to implement this solution, and they vary in the language used for the device_control_page, and in the way the generic application is written.

Our solution (see Figure 1) uses a special Management Browser (the generic management application) which is written in JAVA. It also uses the SNMP protocol for the actual management operations. The device_control_page is written in a special language, oriented towards management of communication devices, compact in space, and enables fast and easy building of management applications. The device_control_page may be retrieved using either SNMP or FTP or HTTP protocol. The device_control_page can reside in one of two places: In the device itself, or in an HTTP Server located in the vendor's headquarters or in a local organization HTTP server which holds the device_control_pages of all devices managed in the organization. The agent holds a pointer to the device_control_page in the form of a URL whose value can be retrieved using an SNMP request to a well-known MIB variable.

We suggest a new language for the device_control_page which is object-based, and is called **NMML** (see Section 4).

This Management Browser could be executed as a JAVA applet from WWW Browsers if security administration, like in Sun's HotJava WWW Browser, is implemented. The architecture of the suggested generic network management application is given in figure 2.

4 The NMML language

NMML (Network Management Markup Language) is a special language designated for describing network management applications. Using NMML, network management applications can be easily developed.

By using NMML, one can build a network management application, with the same ease one builds a Home Page on the Internet using HTML. In the same way that HTML enables us to build impressive, nice looking and graphical web-pages, NMML enables developers to build powerful, easy-to-use management applications. Following is a short description of the language. For a complete example, see Section 6.4 .

Developing management applications using NMML consists of describing two groups of entities:

1. Classes of manageable objects which comprise a hierarchy of objects representing the device to be managed by the application.

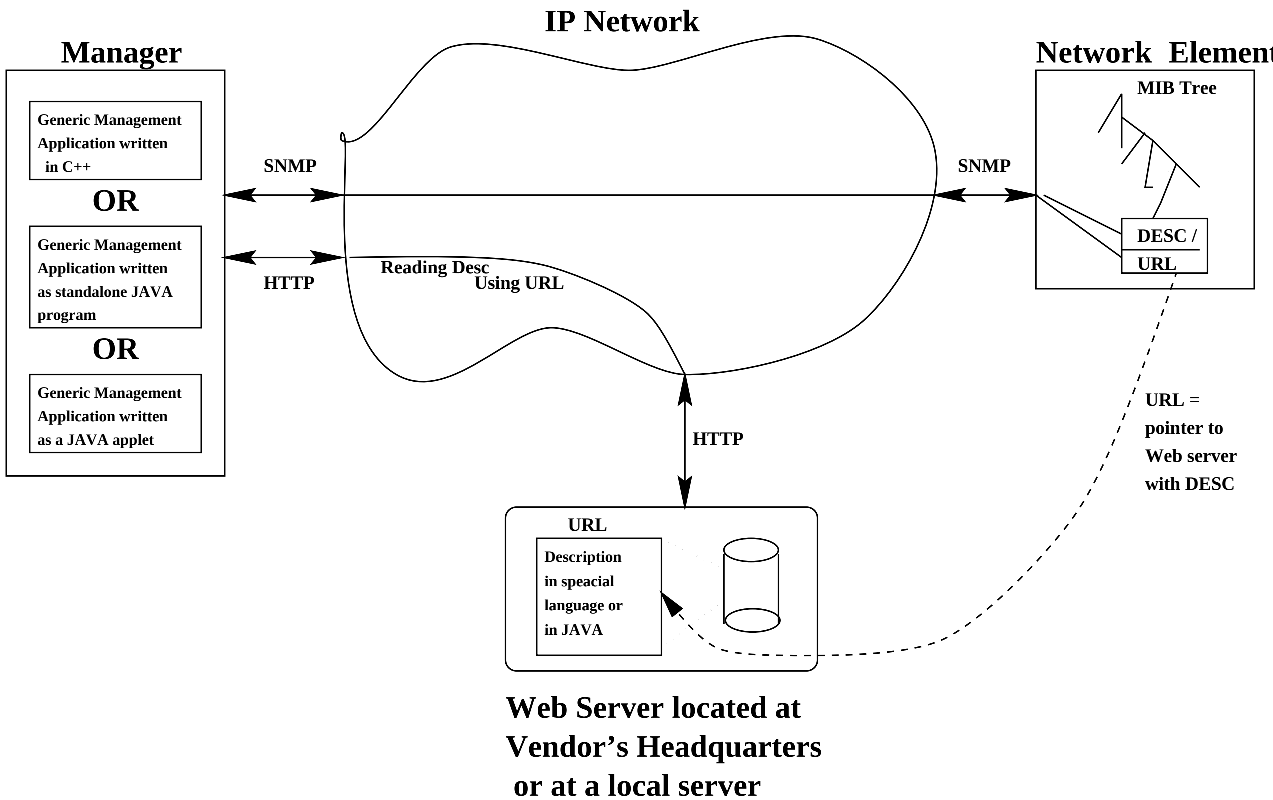


Figure 1: The Solution Model - Our Implementation

2. Menu items and their associated actions which actually implement the management task.

4.1 Describing management classes

Each class of manageable objects is defined using the *ObjectDef(class name)* { *attribute statements;* } statement, where *class name* represents the class name and *attribute statements* represent the set of statements describing the attributes associated with this class. For example:

```
ObjectDef(RAD-TokenRing-Hub)
{
        BitmapList = (http://www.rad.co.il/NMML/TokenRing/Hub/hub.gif);
        [ ObjectStatus = .1.3.6.1.2.1.2;                                   ]
        [ Description = <RAD token-ring Hub with 3 ports>;                  ]
        [ NumberOfChildren = 6;                                            ]
        [ ChildType = ( PORT, PORT, PORT, LED, LED, LED);                  ]
        [ ChildType = .1.3.6.1.2.1.3;                                      ]
        [ ChildrenCoordinateList = ( <10, 10>, <110, 10>, <210, 10>,
                                     <10, 50>, <110, 50>, <210, 50> );     ]
        [ MenuList = ( AlarmListMenuItem, HistoryLogMenuItem, ResetHubMenuItem ); ]
}
ObjectDef(PORT)
{
  ...
  ...
}
ObjectDef(LED)
{
  ...
  ...
}
```

Each of the possible attribute statements can only be specified once per class definition. All attribute statements but the first are optional. Only one of the ChildType statements can be used. The possible attribute statements are:

1. `BitmapList = (list of bitmaps separated by comma);`

Specifies the set of bitmaps which represent the possible states of the class, e.g., if the class represents a led then it contains two bitmaps, one for the *On* state and one for the *Off* state. The list must contain at least one bitmap. The bitmaps are numbered from 1 and upwards, and the default bitmap number is 1 (the first one). For example:

```
BitmapList = ( http://www.rad.co.il/NMML/leds/led_on.gif,
               http://www.rad.co.il/NMML/leds/led_off.gif );
```

2. `ObjectStatus = MIB-Variable;`

Provides a MIB variable whose integer value specifies which bitmap from the list of bitmaps is to be used in order to represent the current state of the object. E.g., if this class represents a led, this MIB variable tells if the led is on or off (a value of 1 stands for the on state and a value of 2 stands for the off state) For example:

```
ObjectStatus = .1.3.6.1.2.1.2;
```

3. `Description = < description of the class >;`

Describes the class literally. For example:

```
Description = <Cisco 1100 Router>;
```

4. `NumberOfChildren = Num | MIB-Variable;`

Specifies the number of child objects that this class contains (e.g., how many ports a slot contains). If a constant Num is specified then Num is used, otherwise the MIB-Variable's integer value is used. For example:

```
NumberOfChildren = 4;
NumberOfChildren = .1.3.6.1.2.1.5.6.7;
```

5. `ChildType = ( list of class names separated by comma);`

Specifies a static list of the class's children by their names (as it appears in the ObjectDef(class name) statement). E.g., if a certain device has two leds, and the led's class name is LED, the statement used is:

```
ChildType = ( LED, LED );
```

The number of children specified in the list of class names in this statement, must match the number of children defined by the NumberOfChildren statement, i.e., in the above example the NumberOfChildren must be 2.

6. `ChildType = MIB-Variable;`

Provides a MIB-Variable whose content is a dynamic list of the class's children. The MIB variable is used with different instances, one per child. The MIB-Variable ASCII-string value specifies the name of the child class. That is, if a class has N children, the MIB variable is queried N times, using a different instance per child, and the ASCII-string value of each instance specifies the desired class for this child. For example:

```
ChildType = .1.3.6.1.2.1.5.6.1; // Value might be "LED"
```

7. `ChildrenCoordinateList = ( <x1, y1>, <x2, y2>, ..., <xi, yi>, ..., <xn, yn> );`

Specifies the location of each child bitmap relative to this class location. For example:

```
ChildrenCoordinateList = ( <10, 20>, <110, 20>);
```

8. `MenuList = ( list of menu-item names separated by comma);`

This statement specifies the list of menu items associated with this object. This list specify the set of actions which can be invoked on the object. Every object for which this attribute is specified, can be selected on the display using the mouse, and thus make the object selectable. Once the object is selected the menu bar is updated according to the MenuList statement, so that the list of menu items in this statement appears in the menu bar. If this attribute is not specified then no actions are available for that object, (a Led normally has no actions) and it is not selectable (it can only be observed). The menu items are specified using their respective names (explained below). For example:

```
MenuList = ( AlarmListMenuItem, HistoryLogMenuItem, ResetHubMenuItem );
```

4.2 Describing menu items

As we have seen above, each management class can be associated with a group of menu items. Menu items are defined using the special keyword
MenuDef(menu item name) { *attribute statements* },
where *menu item name* specifies the name of the menu item, and *attribute statements* represent the set of statements describing attributes associated with this menu item. For example, the following menu definition will cause an SNMP SET Operation of the integer value 1 to the MIB-Variable .1.3.6.1.2.1.4.0 :

```
MenuDef(ResetHubMenuItem)
{
        MenuItemName = Configuration|Reset Hub;
        Action = SetAction;
        MibVarList = {
          .1.3.6.1.2.1.4 Integer(2) wo 1;
        }
}
```

Each menu item definition must contain a name (specified by the MenuItemName statement), an action (specified by the Action statement) and a list of MIB variables participating in this action (specified by the MibVarList statement). The possible attribute statements are:

1. `MenuItemName = list of strings separated by '|' ;`

Specifies the menu item's name and hierarchy in the menu system. The | sign allows us to specify cascaded menu items. For example:

```
MenuItemName = Fault|Alarms|List;
```

In this example `Fault` appears in the menu bar. Selecting `Fault` causes a menu to pop down from the `Fault` entry, in which `Alarms` is found. Selecting `Alarms` causes another menu to pop down, this time from the `Alarms` entry, and in this menu, `List` appears. This is the actual menu entry whose selection causes the invocation of an action.

2. `Action = SetAction | DialogAction | ListAction | ExecAction;`

Describes the action invoked when this menu item is selected. A SetAction action invokes an SNMP SET operation for all the variables in the list of MIB variables. A DialogAction opens a dialog containing the list of MIB variables in a Form like manner. This action is suitable for scalar MIB variables (variables that do not have many instances for a certain object). A ListAction action opens a List dialog containing the list of MIB variables in a Table like manner. This action is suitable for tabular MIB variables, e.g., a list of active alarms for a certain port. An ExecAction action causes the execution of a new command. For example:

```
Action = DialogAction;
```

3. `MibVarList = { list of MIB-variable statements, each in a separate line }`

Specifies the list of MIB variables that takes place in the action invoked by this menu item. Each MIB-variable statement describes a MIB variable and its suitable attributes in the action to be invoked. The syntax of the MIB-variable statement is:

```
ObjectID type access label|value;
```

For example:

```
.1.3.6.1.2.1.2 String(25) rw SystemName;
```

Where:

(a) `ObjectID` contains the ObjectID (SNMP Object Identifier) of the MIB variable, e.g., .1.3.6.1.2.1.5 .

(b) `type` can be one of:

   i. Integer(length) Specifies that this MIB variable holds an integer value. The length parameter is used by the DialogAction and ListAction actions, and it specifies the length in digits that is reserved for this field in the dialog or list that is opened. The use of the length parameter is the same in the Hex-type and the String-type.

   ii. `Hex(length)` Specifies that this MIB variable holds a hexadecimal value.

   iii. `String(length)` Specifies that this MIB variable holds a string value.

iv. `boolean(onVal, offVal)` Specifies that this MIB variable holds a boolean flag with the suitable *on value* and *off value*. In a dialog this MIB variable is represented by a check box.

v. `Enum( <name1, val1>, <name2, val2>, <name3, val3>, ... )` Specifies an enumerated MIB variable with the appropriate names for each possible integer value. In a dialog this MIB variable is represented by a choice list. For example:

`Enum( <Major, 1>, <Minor, 2>, <Marginal, 5> )`

(c) `access` can be one of `rw`(read-write), `ro`(read-only) and `wo`(write-only).

(d) `label` A text describing the meaning of this MIB variable. It is used in dialogs and lists.

(e) `value` specifies the value to be set to the MIB variable in case of a SetAction action.

5 Generic Network Management Application Structure

The following figure describes the architecture of the *generic web-based management application* implementation. It contains the following elements:

1. Engine - The control logic of the application.
2. SNMP - The SNMP layer of the application.
3. HTTP - The HTTP layer of the application.
4. NMML Parser - The element which parses the NMML language.
5. Application Generator - It compiles the parsed code into a set of (JAVA) objects that together make the management application.
6. Management application - The specific management application which is generated on the fly by the application generator.

All the elements of the generic application are written in JAVA. The input parameter to the application is the ip_address of the device to be managed.

The management application uses SNMP to interact with the device, and HTTP to get the NMML code and graphic images. The user interacts with the management application through the management console.

As is usual in network management, the device must have an SNMP agent to communicate with the management application. In addition as part of the generic scheme, a MIB variable, *system.sysDescr* in the device must contain the URL pointing at the NMML device_control_page.

The elements of the generic application are explained below:

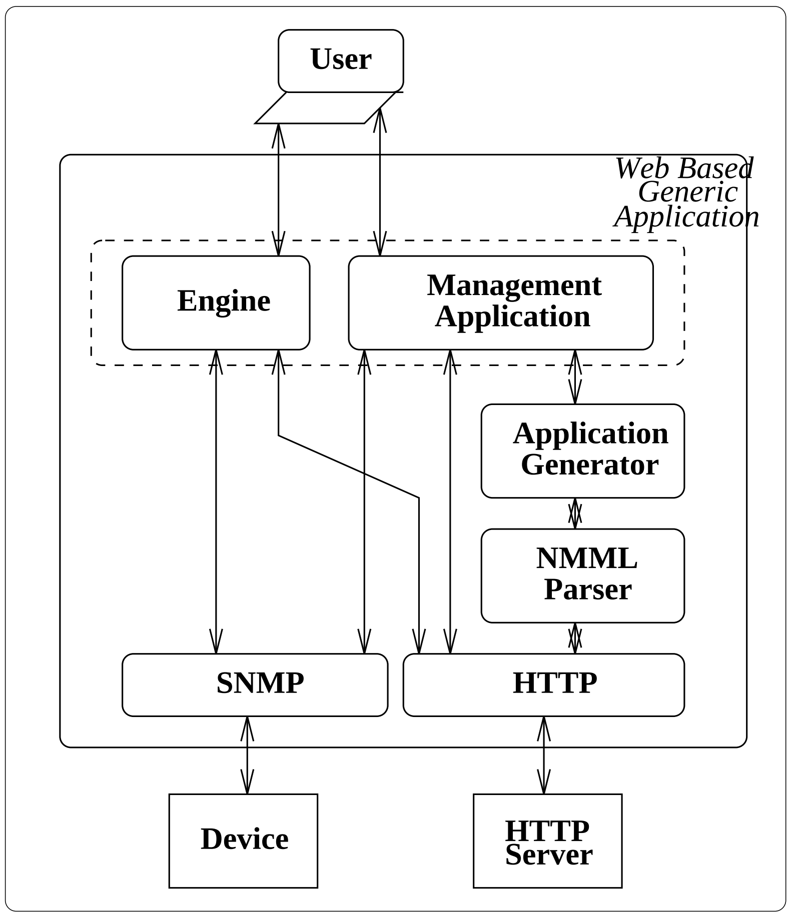


Figure 2: The Generic Web-Based Management Application Architecture implementation.

The Engine The engine of the generic application is the part that activates, controls and coordinates the other elements in the application in order to carry out the task of the generic application.

The engine first reads the content of the system.sysDescr MIB variable, using the SNMP layer, and by this obtains the URL pointing at the device_control_page (that contains the corresponding NMML code) . It then passes the URL to the application generator, using the management application.

The SNMP layer The SNMP layer allows the sending and receiving of SNMP packets in JAVA, and is based on the *Advent Network Management, Inc.* JAVA SNMP implementation.

The HTTP layer The HTTP layer allows the retrieval of data using the HTTP protocol, and is actually the set of JAVA API routines which allows the opening of HTTP connections and then reading data using those connections.

The NMML Parser This part of the application understands the NMML grammar. It takes as an input NMML code, parses it, and feeds the information to the application generator. This part is a piece of JAVA code generated almost automatically by a JAVA-lex and JAVA-CUP products.
The JAVA-lex was written by Elliot Berk (ejberk@princeton.edu) . JAVA-lex takes a JAVA-lex specification file and generates JAVA classes which are used to lexicographically analyze the NMML code.
The JAVA-CUP is a LALR Parser Generator for JAVA named CUP, (serves the same role YACC does for C programmers) and was written by Scott E. Hudson. JAVA-CUP takes a JAVA-CUP specification file and generates a Parser in JAVA.

We combined the output of this two products to generate a comprehensive Parser for the NMML language.

The Application Generator This part of the application generates the specific management application. After receiving the URL from the engine, this element starts reading the device_control_page from the HTTP server using the HTTP layer. The contents of the control_page is fed into the NMML parser. The output of the NMML parser is read by the application generator and it generates the specific management application by creating instances of classes with a parent-child relationship, which together form the specific management application.

The Management Application This is the specific management application generated by the application generator. It is composed of instances of classes with parent-child relationship according to the objects found in the NMML control_page. These classes use SNMP queries to control the device, and use the HTTP layer in order to retrieve data needed for their display.

6 An example

We present an example, describing an imaginary device, its imaginary private MIB, and its device_control_page.

6.1 The device

This device is an Ethernet hub having three Ethernet ports. This device also has a token ring interface. Each port can have a workstation connected to it. All the workstations which are connected into this hub can talk to each other using their Ethernet interface. Each port can be in one of two states, off or on. For each port there is a led, which specifies if the port is on or off. The device also counts the number of erroneous frames that has passed in the last minute through each of the ports. This device is managed using the token-ring interface. The management station manages the device on this interface, using the SNMP protocol.

6.2 The private MIB of the device

The private MIB exploits most of the device attributes, so that the manager has as much control as possible over the device. Assume that the manager can change the state of the port (to off or on), and she/he can also view the amount of errors for each port. The private MIB of this device contains the following MIB variables:

- `exampleHub.portCommand` - a tabular (one instance per port) integer MIB variable which can be set to one of two values, 1(on) or 2(off).
- `exampleHub.portStatus` - a tabular integer MIB variable which describes the port status (1(on) or 2(off)).
- `exampleHub.portErrors` - a tabular integer MIB variable that contains the number of erroneous frames inserted by the connected workstation into the port in the last minute.

6.3 The management application

We would like the management application to support the following:

1. Enable the manager to change the state of each of the ports.
2. View the status of each of the ports.
3. View the number of errors in each of the ports.

We would also like to see a picture of the device having 3 ports, with 3 leds and we would like to see the color of the leds according to their state in the hub. We would like to enable the manager to change the state of the port to on or off, through a menu entry in the application, and to be able to see the status of the port and the number of errors in the port in the last minute. To be more specific, we would like to see a menu entry in the menu bar, for the ports. This menu entry has three menu items:

1. Enable Port (setting it to an on state).
2. Disable Port (setting it to an off state).
3. Port Status and Errors.

6.4 The device_control_page in this case

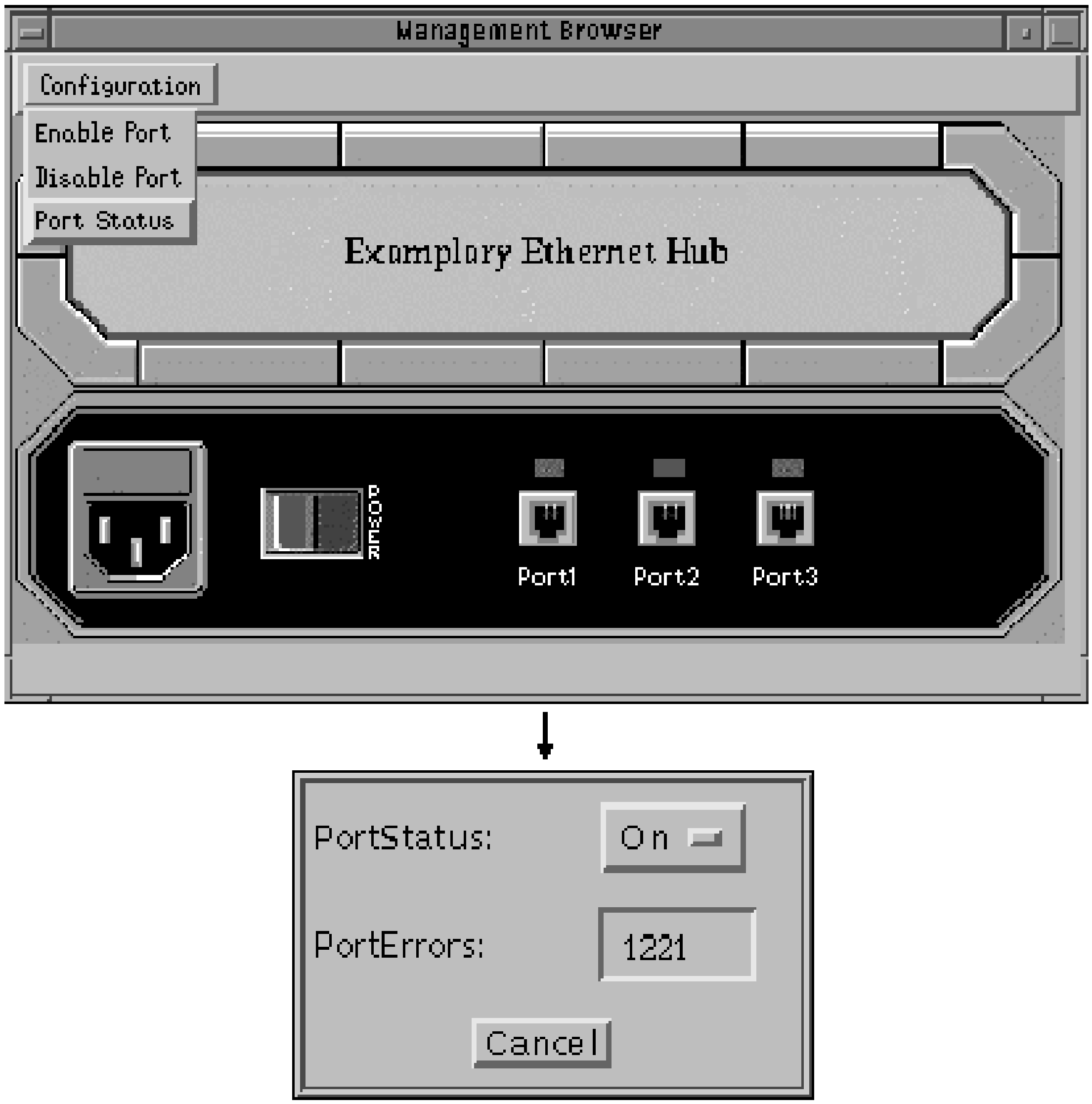
The device_control_page that implements the management application described above, follows:

```
MenuDef(EnablePortMenuItem)
{
  MenuItemName = Configuration|Enable Port;
  Action = SetAction;
  MibVarList = {
    exampleHub.portCommand Integer(2) wo 1;
  }
}
MenuDef(DisablePortMenuItem)
{
  MenuItemName = Configuration|Disable Port;
  Action = SetAction;
  MibVarList = {
    exampleHub.portCommand Integer(2) wo 2;
  }
}
MenuDef(PortStatusMenuItem)
{
  MenuItemName = Configuration|Port Status;
  Action = DialogAction;
  MibVarList = {
    exampleHub.portStatus Enum(<1, On>, <2, Off>) ro 'Port Status';
    exampleHub.portErrors Integer(5) ro 'Port Errors;
  }
}
ObjectDef(Hub)
{
```

```
  BitmapList = ( http://www.rad.co.il/NMML/ethernet/hub/images/hub.gif );
  Description = <Ethernet Hub>;
  NumberOfChildren = 6;
  ChildrenCoordinateList = ( <20, 20>, <120, 20>, <220, 20>,
                             <20, 100>, <120, 100>, <220, 100> );
  ChildType = (Port, Port, Port, Led, Led, Led);
}
ObjectDef(Port)
{
  BitmapList = ( http://www.rad.co.il/NMML/ethernet/hub/images/port.gif );
  MenuList = (EnablePortMenuItem, DisablePortMenuItem, PortStatusMenuItem);
  Description = <Ethernet Port>;
}
ObjectDef(Led)
{
  BitmapList = ( http://www.rad.co.il/NMML/ethernet/hub/images/led_on.gif,
                 http://www.rad.co.il/NMML/ethernet/hub/images/led_off.gif );
  Description = <Led>;
  ObjectStatus = exampleHub.portStatus;
}
```

6.5 Screen shots of example

The following figure represents a sequence of screen shots taken while running the *Management Browser* with the example NMML code.



7 Existing Web Management Approaches

This section describes other directions in the Web-Management area, and their advantages and disadvantages. The data in this section was gathered from several sources including [HMAP96, Ana96, WA96, Sys96a, Sys96b, Rel96, Inc96].

7.1 Device Embedded HTTP Server

By embedding an HTTP server in a managed device, anyone with a Web Browser (and indeed everyone has one today), can access the managed device directly from the Web Browser, as if it was an ordinary Web-Site [Ana96, WA96]. The Embedded HTTP server holds three classes of pages, static pages, dynamically created pages and forms (which allow the user to view and change parameters).

The usage of a standard Web Browser is the main advantage of this approach. It eliminates the need for development and installation of a management application. At the same time it enables management of devices from different platforms with familiar and standard interface, from any location in the network.

While the advantages of this approach are in the management application side, its disadvantages are in the agent side and in the suitability of HTML for management tasks. Since each agent needs a heavy HTTP engine installed, its latency increases, its memory requirements increase and so does the processing cost.

This approach is suitable for managing small and simple devices with sufficient memory, for which the GUI capabilities supported by HTML are sufficient. This will require vendors to invest in embedded HTTP servers technology.

7.2 Using HTTP as an Access Method

In this approach the underlying protocol used to communicate between the management application and the agent is HTTP (which is based on TCP), rather than SNMP (which is based on UDP) [HMAP96, WA96, Sys96b].

This approach allows management transactions to be secure, (while SNMP is not) and allows large amounts of data to be transfered in these transactions. On the other hand, this approach forces a TCP stack implementation in the devices, which increases the latency for small transactions (which are the majority of management transactions), Moreover, it requires the definition of the equivalent of MIB variables and operations in the HTTP domain. Using HTTP as an access method is feasible and practical, and will surely be implemented in devices for management tasks soon, especially when using the Embedded HTTP server approach. Once HTTP security is complete , security will be its biggest drive.

7.3 Web Management Proxy Server

In this approach the Web Browser used for management tasks accesses a proxy server in HTTP, and this server in turn access the devices using SNMP [Ana96]. The proxy server is also termed Mid-Level Manager. The approach enjoys the pros of the previous approaches and overcomes some of the cons.

The user uses the familiar interface of popular Web Browsers to manage devices from any location and using any platform. The approach is scalable and integrates well with current management platforms which use SNMP to access devices, and is a good integration of successful approaches and protocols, i.e., HTTP for the GUI side (man-to-machine interface), SNMP for the management (machine-to-machine interface). The only disadvantage of this approach lies in the need of implementing both the SNMP layer in the device, and the HTTP Server in the proxy, which adds complexity compared to the previous approaches. The approach is suitable for enterprise management and management of large homogeneous networks.

Cabletron Systems and HP OpenView already went in this direction, by introducing products that enable managers to gather data from their standard management (SNMP) platforms, using Web Browsers.

7.4 Web-Based Enterprise Management Proposal

A new proposal by key management applications developers (Microsoft, Intel, BMC, Cisco, Compaq and 75 others) for a standard aimed at making enterprise networks manageable over intranets using World Wide Web technology [Rel96, Inc96].

The initiative is based on the idea of using common WEB Browsers to monitor networks. By pooling disparate types of management data, the initiative promises to integrate network, application, desktop, and systems management, and to deliver on the promise of distributed management. It also promises to make life easier for network managers by using information about relationship between network elements to perform more intelligent handling of event alerts and alarms.

The proposal is made up of three technology pieces, a new schema called HMMS, a new protocol called HMMP and an object broker called HMOM which manages network elements as objects. The proposal will have the advantages of using Web Browsers GUI interface, and using HMMP instead of SNMP. The only problems are the fact that the new protocol and schema are not Internet standards, and the complexity of the proposal which ,in my opinion, may cause its fall.

8 Our Proposal - Advantages and Disadvantages

Our proposal has two important attributes:

1. It is a **generic** management application.

2. It is **web based** .

The first attribute is the important one. This attribute makes our proposal able to manage any communication device easily. The second attribute is also important, but it is simply the best way we found to implement our approach.

In this section we will compare our approach to the other Web Management approaches, and try to outline the advantages and disadvantages of our approach.

The advantages of our approach are:

- The device is a self contained product, which include its own management function. There is no need to install a management application at the manager workstation, except for the generic management application, which can also be loaded through the Web when using it as a JAVA applet. Same as in the Embedded HTTP Server approach.
- Our generic management application offers User Interface in a sufficient level for management purposes, in contrast to the Embedded HTTP server limited User Interface. The application is platform and location independent. And handles multiple versions of the same device.
- We use the SNMP protocol for the management operations. This allows our application to integrate well with common management platforms. It uses the same technology used today in the devices, and by this does not require implementation of TCP stacks in the device, and allow vendors to develop the device part of the management, in the same familiar manner. This is contrast to the Embedded HTTP Server approach in which an additional software investment must be made in developing the device's software. The HTTP protocol is used for getting the device_control_page, and space-consuming elements like images of the device.
- System Administrators can use our approach to develop their own management applications for third-party devices.
- We use a simple and special language, and not a standard language, because we learned from our experience that management tasks repeat themselves over and over in different management application, regardless of vendor and product type. By using a special language we can make the fact of things repeating themselves a leverage to make development of management application easily and quickly.
- Our approach and NMML are extensible.

The disadvantages of our approach are:

- The User Interface and the management capabilities and functions are limited to those offered by NMML. In order to extend these capabilities vendors must extend NMML, and re-implement the generic application. While this is possible, HTML has a much bigger use and community, and surely will be extended faster than NMML. Due to that, using approaches that uses HTML for the UI, like the Embedded HTTP Server approach, has some advantage over our approach.

- The user has to get our Management Browser, instead of using his Web Browser, like in the Embedded HTTP Server approach. This disadvantage is temporary. Once our system works as a JAVA applet it would be accessible from any JAVA enabled browser.

9 Conclusions and future directions

We propose a simple, yet powerful approach of device management, that allows an easy and rapid development of management applications. The approach has its limitations, but in our belief is adequate for the management of small and medium devices. The extensibility of NMML and the generic management application, allows the approach to adjust itself to the needs of the network management community, and to the rapidly changing technology. Vendors can also use our approach to rapidly develop a preliminary version of a later complicated management application, in order to be able to market the device quickly.

Our work may be extended in the following directions:

1. Extending NMML capabilities, such as adding Graphs as management functions.

2. Extending NMML to execute dynamically loaded JAVA code. This will allow adding of any piece of code, and make the approach totally flexible.

3. Support for high-level management operations, like reasoning and automatic decision making.

4. The Management-Browser could be implemented as a Web-Browser Plug-in. This will allow the management of network devices directly from the Web-Browser in which the plug-in was installed, without the need for a stand-alone management browser, or a JAVA applet that needs to be downloaded from the network. Using this implementation the plug-in adds to standard Web-Browsers the capability to understand the NMML language and manage network devices.

5. We would have liked to implement our new scheme as a JAVA applet, and not as a stand-alone JAVA program as we currently do. This would allow the management browser to be executed from a standard Web-Browser, and would enable the dynamic loading of the management browser from the Internet. This, however, requires the use of a proxy server for the SNMP transactions of the management browser, in order to overcome the security limitations of JAVA applets. In future version we plan to use the applet approach once it would be possible to overcome the JAVA security obstacle.

References

[Ana96] Network Analasys. The next web wave - network management. *Data Communications*, January 1996.

[CFSD90] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstal, and James R. Davin. A simple network management protocol. *Request for Comments 1157*, May 1990.

[HMAP96] Brian Harison, Peter E. Mellquist, and Hewlett-Packard Company Adrian Pell. Web based system and network management. *Internet Draft*, June 1996.

[Inc96] Free Range Media Inc. Web-based enterprise management proposal. *Web*, July 1996.

[Per93] David T. Perkins. Understanding snmp mibs. 44 pages, September 1993.

[Rel96] Press Release. Industry leaders propose web-based enterprise management standards effort. *Web*, July 1996.

[Ros91a] Marshall T. Rose. Management information base for network management of tcp/ip-based internets: Mib-ii. *Request for Comments 1213*, March 1991.

[Ros91b] Marshall T. Rose. *The Simple Book - An Introduction to Management of TCP/IP-based Internets*. Prentice-Hall International Inc., 1991.

[Ros94] Marshall T. Rose. *The Simple Book - An Introduction to Internet Management 2nd Edition*. Prentice-Hall International Inc., 1994.

[Sys96a] Barry Bruins Cisco Systems. Some experience with emerging management technologies. *Simple Times*, July 1996.

[Sys96b] Patrick Mullaney Cabletron Systems. Overview of a web-based agent. *Simple Times*, July 1996.

[WA96] Chris Wellens and Karl Auerbach. Towards useful management. *Simple Times*, July 1996.

A NMML Grammer

```
ALPHA=[A-Za-z]
DIGIT=[0-9]
NUMBER={DIGIT}+
MIBVAR=((\.{NUMBER})|(\.#{NUMBER}))+
STRING=({ALPHA}|{DIGIT}|_|\.)*({ALPHA}|_|\.)({ALPHA}|{DIGIT}|_|\.)*
STRINGS={STRING}({STRING}|\ )*
URL=({STRING})://({STRING})?(:{NUMBER})?(/{STRING}:)?(/{STRING})+
MENUITEMNAME={STRINGS}(\|{STRINGS})+
SEMI=';'
```

```
mng_description          ::=     mng_description mng_definition |
                                 mng_definition ;

mng_definition           ::=     object_definition |
                                 menu_definition ;

object_definition        ::=     objectdef_header { object_expressions } ;

objectdef_header         ::=     OBJECTDEF ( STRING ) ;

object_expressions       ::=     object_expressions bitmap_list_expr |
                                 object_expressions bitmap_type_expr |
                                 object_expressions menu_list_expr   |
                                 object_expressions description_expr |
                                 object_expressions numofchilds_expr |
                                 object_expressions childcoor_expr   |
                                 object_expressions childobj_expr    |
                                 bitmap_list_expr ;

bitmap_list_expr         ::=     BITMAPLIST = ( bitmap_list ) SEMI ;

bitmap_list              ::=     bitmap_list , URL |
                                 URL ;

bitmap_type_expr         ::=     OBJECTSTATUS = MIBVAR SEMI ;

menu_list_expr           ::=     MENULIST = ( menu_list ) SEMI ;

menu_list                ::=     menu_list , STRING |
                                 STRING ;

description_expr         ::=     DESCRIPTION = < STRING > SEMI ;

numofchilds_expr         ::=     NUMBEROFCHILDS = NUMBER SEMI |
```

```
                                NUMBEROFCHILDS = MIBVAR SEMI ;

childcoor_expr          ::=     CHILDSCOORDINATELIST = ( childcoor_list ) SEMI;

childcoor_list          ::=     childcoor_list , < NUMBER , NUMBER >  |
                                < NUMBER , NUMBER > ;

childobj_expr           ::=     CHILDTYPE = MIBVAR SEMI |
                                CHILDTYPE = ( childobj_list ) SEMI ;

childobj_list           ::=     childobj_list , STRING |
                                STRING ;

menu_definition         ::=     menudef_header { menu_expressions } ;

menudef_header          ::=     MENUDEF ( STRING ) ;

menu_expressions        ::=     menu_expressions menu_name_expr   |
                                menu_expressions menu_action_expr |
                                menu_expressions menu_mib_expr    |
                                menu_name_expr                    |
                                menu_action_expr                  |
                                menu_mib_expr                     ;

menu_name_expr          ::=     MENUITEMNAMEDEF = MENUITEMNAME SEMI ;

menu_action_expr        ::=     MENUITEMACTION = menu_actions SEMI ;

menu_actions            ::=     SETACTION               |
                                DIALOGACTION            |
                                LISTACTION              |
                                GRAPHACTION             |
                                EXECACTION STRING       ;

menu_mib_expr           ::=     MENUITEMVARLIST = { var_list_expr } ;

var_list_expr           ::=     var_list_expr var_expr |
                                var_expr ;

var_expr                ::=     var_oid var_type_expr STRING STRING SEMI |
                                var_oid var_type_expr STRING NUMBER SEMI ;

var_oid                 ::=     MIBVAR ;

var_type_expr           ::=     INTEGERDEF ( NUMBER )           |
                                HEXDEF ( NUMBER )               |
```

```
                              STRINGDEF ( NUMBER )             |
                              BOOLEANDEF ( NUMBER , NUMBER )   |
                              enum_def ( enums_expr )          ;

enum_def               ::=    ENUMDEF ;

enums_expr             ::=    enum_expr , enums_expr |
                              enum_expr ;

enum_expr              ::=    < STRING , NUMBER > ;
```