A High-Level Notation For The Specification Of Network Management Applications

Alfredo C. S. C. Brites <snmp@mercurio.uc.pt>

Paulo A. F. Simões <psimoes@mercurio.uc.pt>

Paulo M. C. Leitão <pleitao@mercurio.uc.pt>

Edmundo H. S. Monteiro <edmundo@mercurio.uc.pt>

Fernando P. L. Boavida Fernandes <boavida@mercurio.uc.pt>

Abstract

The standardisation and the subsequent widespread use of open network management services and protocols has not lightened the tremendous burden of the developers of network management tools and applications. One of the main problems consists of bridging the gap between the low-level management functionality provided by network management protocols like SNMP or CMIP and the elaborate functionality desired in most network management tools/applications [1].

This paper presents the main aspects of a project that is being developed at the Computer Communications Laboratory of the University of Coimbra, that addresses the problems of specifying and implementing open network management applications in an easy, rapid and versatile way.

The project is based on the definition and use of a high level notation for the specification of network management applications, that frees the applications developer from the details of network management communication protocols and provides the necessary abstractions that enable him/her to concentrate on the desired solutions. In addition, the proposed notation can be used as a common language for the developers of network management tools, as a way to rapidly specify prototype applications and, possibly, in the development of network management application compilers.

I. Overview

This section provides an overview of the developed notation, covering its purpose, main characteristics, architecture and possible uses. Subsequent sections of this paper will present the basic syntax rules, a specialised module directed towards fault-management applications, and an example of the use of that specialised module.

One of the main requirements in the network management field is to be able to rapidly develop prototype implementations. This ability enables a network manager to do a preliminary study on the feasibility of a set of solutions for a given situation. The objective is to compare a set of functionalities in order to select the most appropriate for the problem under study [2]. In this development stage, the main issue is functionality in its broadest sense; the process used to accomplish this functionality and the related details are of minor importance.

Another important requirement in this field is flexibility, in order to provide the means to easily introduce changes in the network management applications. This points to the necessity of network management tools that can interpret user specifications, which, in turn, requires a way to write these specifications.

The above mentioned high-level functionalities are not directly supported by the existing network management services and protocols (SNMP [3], SNMPv2 [4], and CMIP [5]) since they only provide the basic mechanisms to retrieve and manipulate management information, and to trigger actions on managed entities [6]. Their focus is on the low-level processes necessary to accomplish those mechanisms, not on the possible uses for them. Clearly, these are not the sort of issues which should concern the prototype/application developer. He/she needs an appropriate environment to test his/her ideas in a simple and comfortable manner, without wasting efforts in needless details.

I.A Specification of network management applications

The proposed approach attempts to fill the gap between the functionality supplied by current open network management APIs/frameworks [7] [8] and the one needed by the applications developer, through the definition of a high-level notation for the Specification of Network Management Applications (SNMA notation). The notation is generic enough to be used in a broad range of contexts, providing a powerful prototyping and development environment. It represents a first step towards a full network management applications generator.

The SNMA notation incorporates the following key features:

- It provides a high-level means to represent management information. Such representation hides the process used to gather the information (i. e., the communications protocol) and the related details.

- It supports a means for defining and manipulating high-level entities such as relations between objects, events and actions.

A compromise had to be found in the definition of this notation: a too generalist approach would have lacked the capability to manipulate useful objects for specific although frequently used management tasks; a task-oriented notation would have not been sufficiently flexible to accomplish the intended objectives. This compromise was achieved by the adoption of a layered approach (figure 1).

The Basic Syntax Rules (BSR) layer was developed in order to provide a common level of functionality to all the tools built upon this model. Its role is twofold:

- To hide low level communications details from the user.

- To provide a clean high level interface for management information manipulation, conceptually distant from the one supplied by the network management protocol (in this case, the SNMP protocol).

In this layer, the management information gathering process is not modelled in a request-response fashion. Instead, the adopted interface is very close to that of a classic programming language: the existing entities are typed variables (mapped to MIB objects) and a set of operations which can be performed on them. These operations can be used, for instance, to define relations between objects, which is a considerably richer concept than considering objects alone. There are also some incorporated features reflecting the nature of the addressed problem, such as time indexing and the presence of an implicit agent set in every variable.

Since the interface supplied by the BSR is very general, its level is still too low for prototype/application specification. In order to incorporate higher level objects reflecting concepts used in specific, although commonly used tasks and to avoid an excessive task orientation for the BSR, an extension mechanism was envisaged: each broad need commonly felt in Network Management (e.g., fault-management) is addressed by a specialised syntax module, where entities related to particular needs are modelled. Rather than trying to define a general notation capable of covering all imaginable functionalities, each specialised module is restricted to a specific area, making it possible to define deeper features. Moreover, since the design is modular, a new syntax module can be added without having to redefine existing ones.

Specialised modules are already defined for the following areas:

- Monitoring/fault-management.This specialised module, used by an application called *Vigilante*, will be described later;

- Network parameters visualisation;

- Gathering and statistical processing of management information.

A specialised module oriented towards accounting management is currently being developed.

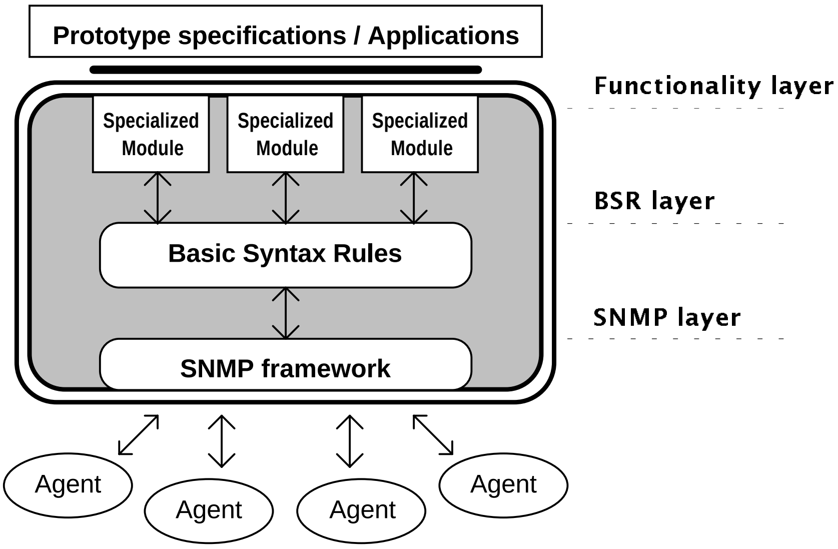

Figure 1: SNMA notation architecture

II. Basic Syntax Rules

The BSR layer of the SNMA notation provides both an uniform representation for available management information and a way to define new entities from relationships between existing objects.

Several important issues were addressed in the syntax definition. Clearly, it must be possible for applications to gather and process management information coming from more than one agent. It is important to include a facility to refer to data polled in past instants in order to track variations over time. It is also important to provide a flexible treatment to columnar SNMP objects.

The most basic element comprising an expression is a MIB object. These objects are referred by its textual name; the BSR layer maps this name to the corresponding low level representation and processes it as a typed variable. Such objects are always associated with an agent set: a reference to an object means "the object's value in all the agents of the current agent set". The definition of the agents comprising the current agent set is application dependent: for instance, in some applications it may be defined interactively while others, due to their nature, may not incorporate such facility.

An object reference may optionally include a time index in order to reference its past values. In general, `var[n]` represents the value of `var` n polls ago.

Expressions may include columnar variables (rather than instances of columnar variables), which refer to all rows of their respective tables. Since such variables do not represent a single value, a need is created for the expression to be evaluated across all agents and/or table rows to which it relates. Note that this expansion is performed automatically in run-time. It is also possible to refer a specific column instance and/or a specific agent. For instance, if `col_var` is a columnar variable, `col_var:[neptune].3.4.1` represents its value for the line identified as "3.4.1" at the agent whose name is *neptune.*

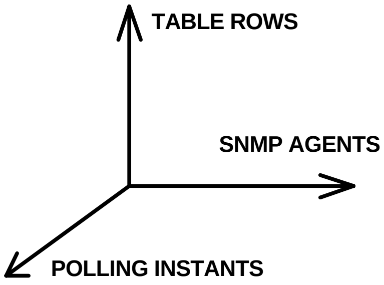

Figure 2: BSR 3-dimensional nature

Since an expression may be evaluated across several agents, MIB table rows and time instances, there is a 3-dimensional evaluation space as shown in figure 2.

Another feature supported by the BSR is the capability to refer to information regarding the applications' functioning parameters whose values can not be known in advance (e. g. , the "current agent set" or the "number of polls done so far"). This information is referenced by a name starting with the @ symbol and is designated as *special variable*.

Finally the BSR provide a standard function set which may take expressions as arguments. One example of such a function is `strlen()`, whose argument is an object of type string, and returns the length of its argument. Each application is allowed to define new functions, thereby extending the standard set.

Besides providing a means to reference SNMP information, the BSR layer also provides a way to define new high-level objects from numerical and logical expressions comprising existing MIB objects and previously defined expressions. This feature is very important since these relationships provide an "added-value" to MIB objects.

As an example of a high-level object defined using the BSR consider the received bytes rate of an interface. Keeping in mind that the MIB-II object `ifInOctets` is a columnar object (one instance per agent interface) that yields the number of octets received so far [9], the expression

```
(ifInOctets - ifInOctets[1]) /
        @PollInterval
```

means, according to the BSR, "the received octets rate for every interface of each agent of the current agent set". This expression illustrates several BSR features:

- Agent set expansion;
- Columnar objects expansion;
- Time indexing (`ifInOctets[1]` means "the value of `ifInOctets` in the previous poll");
- Special variables (`@PollInterval`).

III. The *Vigilante*

As was already pointed out, the BSR are intended to be "generic", i.e. the syntax alone is not specialised enough to provide a means for solving concrete problems in a given network. Instead, it is aimed at offering the possibility of being used in a broad range of contexts and needs.

The SNMA notation offers a potential work basis for the development of management applications that will use the representation capabilities offered by the BSR. This section describes an application called *Vigilante* that has been developed using these capabilities.

The *Vigilante* is an application directed towards fault management. It uses a manager-defined specification written with the BSR notation to detect abnormal situations in the following manner. The notation is used to represent assertions (i.e. logical conditions) that describe "faults" in a network. Those conditions are expressions involving relationships between SNMP variables across the network. The SNMP variables are periodically polled to find out their current values, so as to subsequently compute the expression result. If a condition corresponding to a fault is met, the *Vigilante* will undertake the specified corrective action.

The nature of the BSR implies that the representation of abnormal conditions is flexible enough to span a broad range of fault detection needs, thereby rendering the *Vigilante* potentially useful in a multitude of fault management contexts.

A "user-defined functionality" philosophy was adopted, in order to make the most out of the potential for genericity that the BSR entail. Thus, the user is required to input the assertions that will yield the monitoring capabilities he/she desires, a concept opposed to the use of context-specific, hard-wired fault management features.

Of course, the BSR by themselves do not allow the user to completely define the *Vigilante*'s functionality. They permit a user-defined *monitoring* functionality, by allowing for the generic definition of abnormal situations; however, there are other aspects of fault management functionality that should be "user-definable" if the described philosophy is to be consistently adopted. For instance, the user should be allowed to define corrective actions to undertake upon detection of fault situations. Also, it would be desirable to implement asynchronous fault detection capabilities that are outside the scope of the BSR, namely, the asynchronous detection of trap-PDU arrivals.

Thus, there was a need for a complete "functionality definition" syntax. A script-like "Fault Management Specification Syntax" (FMSS) was therefore defined. The FMSS is a specialised module that extends the Basic Syntax Rules, so as to allow the user to write a full "functionality specification" for the *Vigilante*.

The conjunction of the *Vigilante* with a concrete user specification gives rise to a specific management application. Within the described paradigm, the *Vigilante* by itself does not accomplish fault management tasks; only when provided with an appropriate user specification does it achieve a (flexible) fault management functionality. This model, which represents a step towards the automatic generation of applications, is diagramatically illustrated in figure 3.

The corner-stone of this approach is the flexibility provided by the high-level syntaxes used. This allows for the fast and easy definition of fault management application prototypes, in accordance to the overall objectives of the presented work.

Bearing in mind what has been put forward, the potential uses for the *Vigilante* are:

- To respond to simple fault and performance management needs, by defining concrete user specifications directed to the handling of those needs.
- To support the development of more advanced management systems, allowing the rapid testing of potential solutions for undesirable situations.

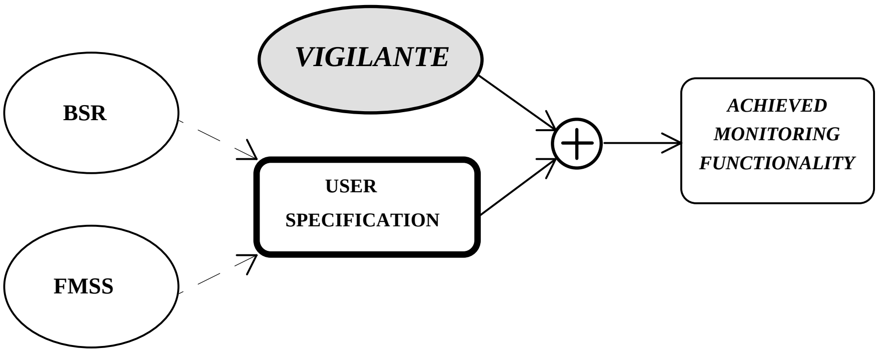

Figure 3: Vigilante's working paradigm

IV. An Example Of A User Specification For The *Vigilante*

The following example shows how a Network Administrator or an Applications Developer can use the *Vigilante* to achieve a specific management functionality. The desired behaviour is to detect all "defective" interfaces in SNMP-agents across a network which is formed by SNMP-agents named "agent1", "agent2", ... , "agent**n**".

Interfaces will be considered "defective" when their input packet errors exceed a threshold corresponding to 10% of the total input packets, with an accumulated total of at least 20 such errors.

Each interface is to be analysed every 15 seconds, to see if it crosses the error threshold.

In case the total number of errors exceeds 200, the interface is to be turned off. Also, a file named *vigilante.log* is to be produced, containing a description of every "defective interface" detection.

The presented problem requires that an expression describing a "defective interface" be written. For this purpose, the following columnar MIB-II variables (assumed to be supported by all the SNMP-agents in the network) will be used:

ifInErrors: number of input packets in error

ifInPkts: total number of input packets

ifDescr: strings with interface descriptions

ifAdminStatus:
current interface status (0 = down, 1 = up)

These columnar variables all belong to the MIB-II table **ifTable**, in which each row contains information on a particular interface of the SNMP-agent where the table resides.

The expression that describes a defective interface can be expressed using the Basic Syntax Rules as (for **p**% input error threshold and **q** accumulated errors):

```
(ifInPkts-ifInPkts[1])*p/100 <
ifInErrors-ifInErrors[1] &
ifInErrors>=q
```

The complete user-input specification is presented in figure 4 (using the values 10 for **p** and 20 for **q**).

The presented example illustrates some of the constructs supported by the FMSS: **actions**, **procedures** and **events**. Their roles in the example are described bellow.

- **Actions definition**

Actions are constructs that model an activity. An action is defined by specifying its name and the parameters necessary to fully describe it.

This example uses two types of actions: an SNMP-set action and a log action:

- SNMP-Set action

This action is named `turn_off`. It simply sets the value of the MIB variable `ifAdminStatus` to `DOWN` therefore turning off the monitored interface.

- Log action

Its purpose is to register an event occurrence in a file. In this example, such an action is triggered whenever a "defective interface event" is detected (see bellow). It invokes the registration (on a log file) of a text string containing the agent where the defective interface was detected (`@hit_location`) and the `ifDescr` and `ifInErrors` variables value on the agent and table row where that event occurred. A typical output for this action is:

```
Defective interface: Ethernet0
    Agent: venus
    Number of Errors: 25

Defective interface: Serial2
    Agent: neptune
    Number of Errors: 39

    <...>

    <...>
```

Although not illustrated in this example, it is possible to define other action types. The possibilities include but are not limited to:

- trouble-ticket generation (`Ticket` action).
- launching of independent processes (`Execute` action).
- writing past values of specified expressions in a file in order to be processed off-line by other applications (`Snapshot` action).

- **Procedure definition**

A procedure construct is basically a sequence of actions. Several flow- and thread-control mechanisms are available. The presented procedure is named `react`. When invoked it triggers the unconditional execution of the `message` action and, if the number of input errors (`@ifInErrors`) is greater than 200, the execution of the `turn_off` action.

```
# Anything following a '#' is a comment.

            # GENERAL PARAMETERS
{
agent-set         agent1, agent2, <...> agentn  # network composition
log-file          vigilante.log                 # log file name
}

            # DIRECTIVE FOR C-STYLE PRE-PROCESSOR
$INCLUDE definitions_file                       # file with useful
                                                # constant definitions

            # ACTION DEFINITION (SNMP-SET ACTION)
SNMP-sets turn_off {
      ifAdminStatus=DOWN      # turn off defective interface
}

            # ACTION DEFINITION (TEXT LOGGING ACTION)
Log message {
                              # log a message about defective interface
      eols(1)
      "Defective interface:"
      @ifDescr                # interface descriptor at last error location
      tabs(1)
      "Agent: "
      @hit_location           # agent name at last error location
      tabs(1)
      "Number of errors:"
      @ifInErrors             # number of errors at last error location
}

            # PRE-PROCESSOR MACRO THAT FIGURES OUT WHETHER
            # ANY INTERFACE EXCEEDS p% INPUT ERRORS THRESHOLD
$DEFINE ERRORS(p) (ifInPkts-ifInPkts[1])*p/100 < ifInErrors-ifInErrors[1]

                              # note that ifInErrors[1] means
                              # "ifInErrors 1 polling period ago"

            # ERROR PROCEDURE FOR DEFECTIVE INTERFACE DETECTION
PROCEDURE react   {
      1     message                             # log the message
      2     if @ifInErrors>200 then turn_off    # conditionally turn off
                                                # the detected interface
}

            # "EVENT DEFINITION" FOR DEFECTIVE INTERFACE DETECTION
POLLED-EVENT detect     {
                        # construct that binds synchronous condition
                        # monitoring definition to a corrective procedure

      condition   ERRORS(10) & ifInErrors>=20   # assertion for
                                          # detecting defective interfaces
      poll-period 15                      # 15 second period for polling

      procedure   react                   # do this if condition holds

}
```

Figure 4: user Specification Example

- **Event definition**

An important construct of the FMSS is the **event** since it may be used to model a fault situation. Its main fields are the fault definition and the procedure to execute when the fault occurs.

According to the fault situation nature, there are two classes of events: synchronously monitored (polled events) and asynchronously monitored. In a polled event, the fault situation is described by a condition (defined according to the BSR) that is periodically tested. For asynchronously monitored events the fault situation corresponds to the arrival of a specified number of traps comprising a defined set of characteristics. This class of events is not present in the example.

The presented polled event is named `detect`. When it is activated, the defined condition is tested every 10 seconds (`poll-period` field). If the condition is verified, the `procedure` specified in the procedure field is executed.

V. Conclusions And Future Work

This paper presented the basic ideas behind a project that addresses the development of a high level notation for the specification of network management applications. The notation is being successfully used at the University of Coimbra as a common language for the development of network management tools and prototype applications.

The proposed notation has the intent to bridge the gap between the basic mechanisms offered by the common network management protocols/APIs and the high-level functionality required by network management applications.

The paper highlighted the simple but powerful syntax rules of the notation, that enable an application developer to easily write complex expressions and object relations, and to define new objects from numerical and logical expressions, freeing him/her of the low-level details of network management services and protocols. In addition, the capability to use specialised syntax modules was illustrated with the presentation of a specification for a fault-management application.

The presented work opens a wide field for research and development activities in network management. At the moment, several specialised syntax modules are being developed as well as a set of tools that can interpret specifications written according to the proposed notation. Although these are very interesting activities, there are still two highly promising areas to address: the development of a notation that would be independent of the underlying protocol environment (SNMP or CMIP) and the development of network management application compilers capable of taking advantage of the presented notation. These constitute two challenging areas that can lead to the real openness and to a high availability of network management tools and applications.

VI. References

[1] Michael L. Kornegay, "Toward Useful - and Standardized - SNMP Management Applications", The Simple Times, vol. 2, no. 2, pp. 1-4, March/April 1993.

[2] Allan Leinwand, Karen Fang, "Network Management: a Practical Perspective". Addison-Wesley Publishing Company, New York, 1993.

[3] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstall, James R. Davin, "A Simple Network Management Protocol", Request for Comments 1157, DDN Network Information Center, SRI International, May 1990.

[4] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose, Steven Waldbusser, "Introduction to version 2 of the Internet-standard Network Management Framework", Request for Comments 1441, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting Inc., Carnegie Mellon University, April 1993.

[5] ISO/IEC 9596-1:1991 - Information technology - Open Systems Interconnection - Common management information protocol - Part 1: Specification, Ed. 2, International Organization for Standardization, International Electrotechnical Comission, 1991.

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

[7] Barry Bruins, "Windows SNMP: an SNMP API for MS Windows Applications", The Simple Times, vol. 2, no. 3, pp. 1-3, May/June 1993.

[8] Aiko Pras, Jacques Togtema, "SNMPv2 at Twente University", The Simple Times, vol. 3, no. 1, pp. 1-4, February 1994.

[9] Marshall T. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II", Request for Comments 1213, Performance Systems International, March 1991.

Acknowledgement

The work presented in this paper has been partially funded by JNICT (Junta Nacional de Investigação Científica e Tecnológica) under contract number PBIC/C/TIT/1219/92.

Author Information

Alfredo Brites is a software engineer for the Portuguese ALCATEL Software Center in Cascais. He received his B.S. degree in Informatics Engineering in 1993 from the University of Coimbra, where he worked on network management research. He his currently interested in the OSI network management framework and the TMN network management concept.

Paulo Simões is involved in research in computer networks at the Computer Communications Laboratory of the University of Coimbra. He received his B.S. degree in Informatics Engineering in 1993 from the University of Coimbra, where he is currently working towards a M.S. degree in Information Systems and Technologies. His current research interests include network management, distributed decision support systems and high-speed networks.

Paulo Leitão is a member of the research team in computer networks at the Computer Communications Laboratory of the University of Coimbra. He received the B. S. degree in Informatics Engineering from the University of Coimbra in 1993, where he is currently working towards a M.S. degree in Information Systems and Technologies. His current research interests include network management, expert systems and intelligent networks.

Edmundo Monteiro is Research Assistant and Lecturer at the Electrical Engineering Department of the University of Coimbra where he has been involved in research in computer networks for the past nine years. Currently he is working towards a Ph.D. degree. He participates in RARE activities as a member of the Working Group on Lower Layer Technologies (WG-LLT) since 1987, and in the Portuguese Academic Network RCCN since its foundation. His current research interests include network interconnection, performance, congestion control and routing.

Fernando Boavida is a Professor at the University of Coimbra, Portugal, where he is the leader of the Computer Communications Group. He received the B.S. degree in Electrical Engineering in 1982, the M.S. degree in Telecommunications in 1986, and the PhD degree in Informatics in 1990, all from the University of Coimbra, Portugal. He is the chairman of the Portuguese National Standards Organization Technical Committees on Telecommunications and Data Communications. He participates in RARE activities and is a member of EWOS EGLL. His current research interests include network management, protocols for high-speed networks, internetworking, and performance evaluation of protocols. Mr. Boavida is a member of IEEE.