The SNMP evolution: lost on simplicity or on functionality

Rui Pedro Lopes
ESTiG, Instituto Politécnico de Bragança,
5300 Bragança, Portugal
{rlopes@ipb.pt}

and

José Luís Oliveira
DET/IEETA, Universidade de Aveiro
3810 Aveiro, Portugal

ABSTRACT

The SNMP framework has gained a new stimulus with the efficient emergence of the third version (SNMPv3). Beyond its enrichments, namely the security model, the enormous base of legacy knowledge and legacy systems leads the SNMP management framework to a necessary choice in nowadays management scenarios.

However, its services correspond roughly to low-level operations for setting or retrieving network equipment parameters. Traditionally, high-level management operations were outside the scope of IETF strategy.

The IETF Distributed Management working group have been producing normalization documents that intent to apply to the enrichment of SNMP semantics, especially in what concerns the processing of management information. One of such deliverables is the Expression MIB that, up till now, is in the Internet draft standard track.

This paper will highlight the recent outcome of this WG, will present an Expression MIB implementation and will discuss the cost of these more powerful solutions on the "keep simple" and "low inference" principles of SNMP engines.

Keywords: Distributed management, SNMP, Disman, Expression MIB.

I. INTRODUCTION

For several years the network management buzzword was mostly associated with SNMP. Guided by the simplicity and the shorter inference principles soon has conquer the attention of a market with a big appetite for this solutions. However, its evolution has suffered from several drawbacks and has open space for other approaches.

The straight path that was maintained by SNMPv3 working group, which last results were published as draft standards by the IETF, may have provide a new breath into the SNMP management framework. SNMPv3 tries to eliminate previous versions weaknesses by the inclusion of some new features. Among these are the security support and a flexible architecture that allows the redefinition of current modules or the introduction of new parts inside the framework. Each SNMP configuration is classified as a "SNMP Entity" composed by several interacting modules: Dispatcher, Message Processing, Security, Access Control and Application module. The combination of these modules allows providing different SNMP roles (i.e. an agent, proxy or manager) [1].

The Application(s) use services from the SNMPv3 engine to send and receive messages, authenticate, encrypt and control the access to managed objects [2].

The Dispatcher subsystem coordinates the communication between SNMPv3 engine subsystems and differentiates modules belonging to the same subsystem. Based on the PDU information, it determines which application should be invoked and coordinates the respective transport mappings.

The SNMP framework is a centralized approach, i.e. a NMS uses distributed agents to collect management information. This data is retrieved on demand by the NMS to be processed.

This approach has some drawbacks, in particular due to the lack of extensibility and scalability of the model on very large networks. This constraint results from the inability of a centralized manager to handle huge amounts of management information and also because centralized polling across geographically distributed sites is infeasible and expensive [3]. Moreover, system updates usually entail the modification of several agents or of the management station itself. In addition, there are occasions where it is necessary to cope with situations where the management station is not accessible. The classic management architectures are not well suited for low-bandwidth or disconnected operation.

Several authors have addressed these problems along the past years [4-6] resulting in ad-hoc and partial solutions typically based on management distribution and delegation. Inside the IETF, the Distributed Management (Disman) WG was chartered to define an architecture where a main manager can delegate control above several distributed management stations thus improving scalability through distribution and allowing “off-line” operations.

II. DISMAN

Management distribution allows reducing the processing load on traditional centralized management station (NMS) by delegation tasks upon several Distributed Managers (DM) or upon more powerful agents. A DM is an SNMP entity that receives requests from another manager and executes those requests by performing management operations on agents or other managers.

Since the management entities are split over the network and collaborate between themselves by assignment, a hierarchy of several “islands” is created increasing the robustness and fault tolerance of the overall management system. Although if the access to the central manager is not possible, each DM may handle locally critical situations.

The IETF Disman framework is based on distributed applications and services. This kind of application performs some management function, often by monitoring and controlling managed elements. The distributed management services can perform functions or store information once for all applications on the local system thus making a set of applications more efficient. Each service is provided by a specific MIB interface.

Currently there are being proposed several MIB to address different but complementary issues of management operations distribution [7]:

- Event MIB
- Notification Log MIB
- Remote Operations MIB
- Schedule MIB
- Script MIB
- Expression MIB

The Event MIB is the successor of the SNMPv2 Manager-to-Manager MIB. It provides the ability to monitor MIB objects either locally or remotely and takes an action when a trigger condition occurs.

The Notification Log MIB is intended mainly for notifications providers but may be also used by consumers. It defines a mechanism to cope with notifications lost by recording each notification data.

The Remote Operations MIBs group (ping, traceroute, lookup) enables the correspondent network-checking operation to be performed at a remote location. It provides a standard way to perform remote tests, to issue periodical sets of operations, and to generate notifications with test results.

The Schedule MIB provides the definitions to perform the scheduling of actions periodically or at specific times and dates. The actions are modeled by SNMP set operations on local MIB variables (restricted to INTEGER type). More complex actions can be realized by triggering a management script, which is responsible for performing complex state transitions.

The Script MIB module allows the delegation of management functions over distributed managers. Management functions are defined as management scripts written in a language supported by the managers. It may be a scripting language (such as TCL) or native code, if the remote site is able to execute this code. The module does not make any further assumptions on the language. The distributed manager may be decomposed in two blocks: the SNMP entity, which implements this MIB, and the runtime system, capable of executing the scripts. The Script MIB sees the runtime system as the managed resource, which is controlled by the MIB. The runtime system can be defined as an SNMP application, according to the SNMPv3 architecture.

The Expression MIB was planned to move to the agent side part of the management information processing typically performed by managers. In other words, it supports externally defined computation expressions over existing MIB objects. The Expression MIB allows providing the Event MIB with custom-defined objects. The result of an expression can trigger an event, resulting in an SNMP notification. Without the Expression MIB such monitoring is limited to the objects in predefined MIBs.

The work presented here is mostly based on an implementation of Expression MIB proposal.

III. EXPRESSION MIB DEVELOPMENT

There are several reasons for a manager to apply some kind of expression on management information.

Aggregation of data can be done in simple statistical tasks, such as the percentage of inbound discarded packets that contained errors (1), or in expressions with a higher degree of complexity.

$$100 \times \frac{\text{ifInErrors}}{\text{ifInDiscards} + \text{ifInErrors} + \text{ifInUnknownProtos}} \qquad (1)$$

The Expression MIB is currently an internet-draft (11th) of the Distributed Management working group, within the Operations and Management Area of the IETF. The MIB is divided in three main groups [8]:

- `expResource` – this group is related to resource control, with particular incidence on sampling parameters since this operation can have some impact on system resources.
- `expDefine` – is organized in three tables which gather information about the expression definition and about the errors occurred while evaluating it:
  a) `expExpressionTable`, defines the expression string, the result type as well as the sampling period.
  b) `expErrorTable` maintains a table of errors' registers gathering information such as: the last time an error occurred on evaluating the expression, the operation in which it occurred, the error type.
  c) `expObjectTable` controls each element characteristics inside the expression. The expression string may contain variables and each variable may have different sampling types and be or not wild-carded.
- `expValue` – this group has a single table which instantiates the evaluation objects. It is by querying this table that the result from the expression is known.

Sampling and Wildcards

The Expression MIB supports three types of sampling:

1. *absolute* – the objects are sampled just before calculating the result.
2. *delta* – the difference from one sample to the next. It is necessary to maintain the last sample. Creates a constant overhead whether or not anyone is looking at the results, so not very suitable for severely limited environments.
3. *changed* – boolean indicating whether or not the object changed its value since the last sample.

In addition to sampling, the MIB also defines wildcarding, allowing the usage of a single expression over multiple instances of the same MIB object. While regular objects are resolved by a SNMP Get operation, wild-carded objects are controlled through the GetNext operation. Users are familiar with wildcarding for referencing multiple files (such as "cp foo.* /tmp"). On this MIB, wild-carded objects are attributes. If there is more than one wildcard variable on an expression they all must have the same OID termination (semantics) to maintain coherence on the result.

For example, the expression (2) has two variables each corresponding to a wild-carded OID, ($1= "1.3.6.1.32.1.4" and $2= "1.3.6.1.50.2.7.1.321").

$$100*\$1/\$2 \qquad (2)$$

The object values are retrieved by GetNext operations thus retrieving the instance INDEX. If the result from GetNext $1 is "1.3.6.1.32.1.4.1.2.3", the INDEX part is "1.2.3". So $2 will be "1.3.6.1.50.2.7.1.321.1.2.3".

An OID can be specified (`expExpressionPrefix`) in order to help retrieve the INDEX. In this example it can be captured in each of the two OIDs since both follow a MIB definition where it is possible to look at the INDEX clause.

Subsets

According to the conformant statements the implementation of the Expression MIB can leave out several parts.

1. *No wildcards* - significantly reduces complexity. Suitable for expressions made up of individual MIB objects but not suitable for expressions applied across large tables.
2. *No Deltas* - reduces state that must be kept and the burden of ongoing processing unnecessary sampling threads. Suitable for applications that do not require expressions or events on counters.
3. *One object expressions* - reduces the complexity of parsing expressions, retrieving multiple objects per expression and doing expression evaluation. This is the slightest implementation of the Expression MIB that supports the threshold of the Event MIB.

Expressions

The key aspects in defining expressions are *parameters*, *results* and *operators*. We can define an expression as: "*result = parameter operator parameter*"; where "*parameter = constant | variable | function | result*". The Expression MIB allows several operators and a set functions that helps to build something similar to typical expressions in any programming language.

An expression is executed through a row on the `expValueTable`. Each row has only one column, formatted according to the result type of the expression. The value is accessed by an OID containing the OID for the data type, the expression name and a fragment (Fig. 1).

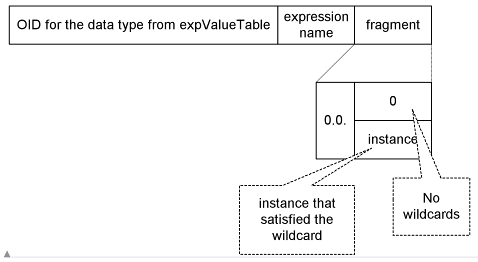
OID for the data type from expValueTable
expression name
fragment
0.0.
0
instance
instance that satisfied the wildcard
No wildcards

Fig. 1. Value identification OID.

The expression name has the form of *x*."owner".*y*."name" converted to dot separated integers. The integer *x* is the length of the owner and *y* is the length of the string which identifies this expression to the particular owner. Each word character is converted to integer and separated from the other integers by a dot.

The fragment starts with "0.0." and ends with a zero, if there is no wildcard or, otherwise, with the instance that satisfied the wildcard.

Implementation Issues

The implementation of the Expression MIB can be divided in two sections:

1. The communication module, responsible for receiving and sending SNMP commands.
2. The agent, responsible for the SNMP agent behavior.

With a well-established interface between the communication mechanism and the SNMP engine it is possible to switch modules maintaining the agent. This feature is useful if we want, in runtime, to use SNMP or other communication method, for example, to check CORBA or RMI performance, or to add mobility to the agent [9].

Considering the SNMP operations and the tree-like organization of objects in the agent, some decisions can be made to help on the agent architecture planning.

Management operations have information about "which" object and "what" to do with it. In "which", it is possible to point precisely the object (the case of *get* and *set*) and to define a walking procedure (*get-next* and *get-bulk*). In "what", the operations are retrieval (*get*, *get-next* and *get-bulk*) and restore (*set*).

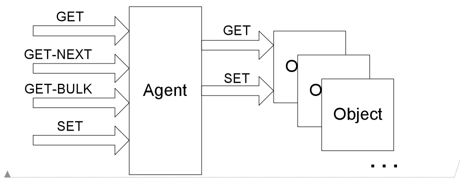

Fig. 2 – SNMP operations on Agents.

Adapting these concepts to an O-O language, the "which" is modeled by a container class (`Agent`) and the "what" are methods to call on contained objects (`Object`) (Fig. 2).

To evaluate an expression it is necessary to recognize the expression components (operators, functions, constants and variables), i.e. the lexicon, and the grammar (the expression organization). There are, available as public domain software, lexical and grammar analysis tools, which generate code such as C [10] or Java [11]. As this implementation is Java based, the chosen tools were JLex, a lexical compiler, and JavaCup, a grammar compiler [12]. Both compilers generate source code based on specification files. These routines are then compiled (into Java .class files) and included in the Expression MIB agent. The lexical analyzer starts reading the stream of characters and tries to matches the sequences identifying tokens. The tokens information is forward to the grammar, which groups tokens into meaningful sequences and invokes action routines to act upon them. In this particular case, it must recognize a complete expression and evaluate the result.

Lets now discuss how the communication model and Expression MIB agents it together (Fig. 3). When started, the agent waits for input. When receiving a SET message it inspects to which table it is destined. After populating the appropriate table, it confirms if both the `expExpressionEntryStatus` and all the related `expObjectEntryStatus` are set to 'active'. If so, it creates an entry on `expValueTable`, after checking for syntax errors.

Unknown
Field Code Changed

Unknown
Field Code Changed

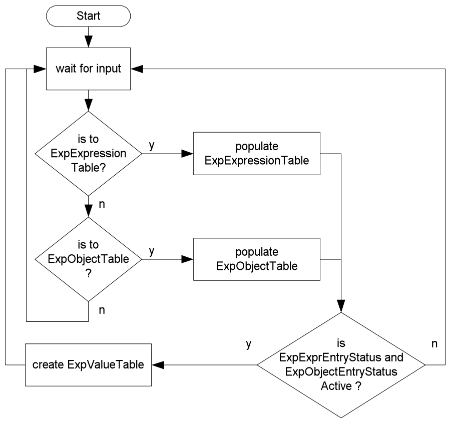
Start
wait for input
is to ExpExpression Table?
y
populate ExpExpressionTable
n
is to ExpObjectTable ?
y
populate ExpObjectTable
n
y
is ExpExprEntryStatus and ExpObjectEntryStatus Active ?
n
create ExpValueTable

Fig. 3. Expression definition.

This object is then responsible for calculating the expression. If the expression has some sort of delta sampling, it launches a thread to periodically calculate the expression and store the result. If the expression is 'absolute', meaning that there are no periodic sampling involved, the expression is calculated only when the `expValueTable` is queried.

The process of calculating the expression is, on the whole, the most complex part, particularly when wildcarding is used. For this purpose, the agent:

1. Retrieves the expression string (`expExpression`).
2. Creates a parser object (based on the code generated by JLex and JavaCup).
3. Checks to see if the expression is wild-carded (`expExpressionPrefix`).
4. Builds a list of objects (variables) that the expression contains.
5. Retrieves the value of each object (`expObjectTable`).
6. Calculates the expression value and stores it in the appropriate `expValueTable` instance.

IV. EVALUATION

The Expression MIB documentation is clear and the included examples also help the deployment phase. However, while the `expExpressionTable` and the `expErrorTable` are quite straightforward the `expObjectTable` has some aspects that require further explanation (i.e., the wildcarding aspect is somehow very scattered on the recommendation).

To study the impact of adding a parser to an agent we have performed some preliminary load tests. In these tests we were mainly concerned with the overload of delta sampling by comparing this situation to the situation of 'absolute' value.

We measured the agent used memory for 0, 1, 20 and 100 expressions with one (Fig. 4) and three (Fig. 5) variables both for absolute and delta sampling.

For reference, we measured the memory load of the JVM by launching a "do nothing" program and found that it uses 3780 Kbytes. The Expression MIB agent with no objects (0 expressions) uses 5884 Kbytes.

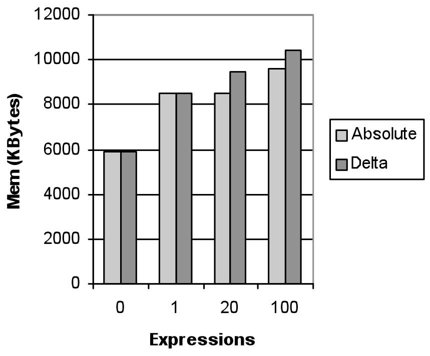

Fig. 4 – Memory load for 1 variable expression.

We can see that, as expected, the number of expressions is proportional to the used memory. Moreover, the difference between delta and absolute expressions is considerable (near 25% for 20 one variable expressions and 18% for 100 one variable expressions).

The memory requirements increase with the number of expressions and with the number of variables per expression.

For CPU utilization we also did some tests by changing the sampling interval (for absolute sampling the CPU is used only when a get message is received on a value object). For 100 expressions with evaluated every 5 seconds the processor (Intel Pentium II 333MHz) was near 100% load. For 20 expressions evaluated every 10 seconds it as near 10%.

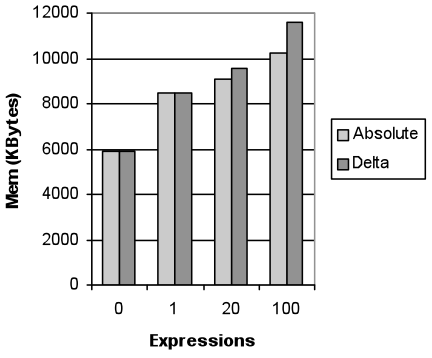

Fig. 5 – Memory load for 3 variables expression.

For the pointed values, the memory requirements are somewhat excessive for restrictive environments. The JVM we used (Java 2 Platform Standard Edition) is not targeted to such kind of platforms and we did not try a more adequate virtual machine, such as the Java 2 Micro Edition. In terms of CPU usage, it is very dependent of the sampling period and may be considered acceptable if the interval between samples is sufficiently long.

V. CONCLUSION

The management of enterprise networks, i.e., to monitor and to act on network components, is a task that involves commonly the use of heavy and complex applications. This difficulty is further enlarged in situations where network scale or connection characteristics inhibit the full use of the SNMP framework. The Disman framework, proposed by the IETF, addresses these problems by distributing some of the management application responsibility to the agents (Distributed Managers).

This paper has presented an implementation of the Disman Expression MIB that supports externally defined computation expressions over existing MIB objects. However, the upcoming of such solutions might compromise the performance and raise new requirements on the agents' side. As happen with IP during the past, SNMP supporters may struggle to maintain agents as simple as they can and continue throwing the management task to some higher level, typical centralized and real-time limited systems.

VI. REFERENCES

[1] J. Case, R. Mundy, D. Partain, B. Stewart, RFC2570 (I), "Introduction to Version 3 of the Internet-standard Network Management Framework", Internet Request for Comments 2570, April 1999.

[2] B. Wijnen, D. Harrington, R. Preshun, RFC2571 (DS), "An Architecture for Describing SNMP Management Frameworks", Internet Request for Comments 2571, April 1999.

[3] R. Sprenkels, J-P Martin-Flatin, "Bulk Transfer of MIB Data", The Simple Times, Vol. 7, N. 1, March 1999.

[4] G. Goldszmidt, Y. Yemini, "Delegated Agents for Network Management", IEEE Communications Magazine, Vol. 36 No. 3, pgs. 66-71, March 1998.

[5] José Luís Oliveira, Arquitectura para Desenvolvimento e Integração de Aplicações de Gestão, PhD Thesis, University of Aveiro, September 1995.

[6] José Luís Oliveira, J. Arnaldo Martins, "A Management Application Programming Interface", Proc. DSOM'94, Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, October 1994.

[7] Distributed Management (disman) Charter, http://www.ietf.org/html.charters/disman-charter.html

[8] Bob Stewart, Ramanathan R. Kavasseri, "Distributed Management Expression MIB", draft-ietf-disman-express-mib-11.txt, February 2000.

[9] José Luís Oliveira, Rui Pedro Lopes, "Distributed Management based on Mobile Agents", Proc. of the 1st International Workshop on Mobile Agents for Telecommunications Applications – MATA'99, October 1999, Ottawa, Canada.

[10] Manson, T., Brown, D., lex & yacc, O'Reilly & Associates, 1990, ISBN 0-837175-49-8.

[11] Appel, A., A Modern Compiler Implementation in Java, Cambridge University Press, 1998, ISBN 0-521-58388-8.

[12] JLex & JavaCup, http://www.cs.princeton.edu/~appel/modern/java/.