A network management agent as tool for the development of Management Information Bases

Jan Arend Jansen, Eric van Hengstum and Albert Nymeyer
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, Enschede, the Netherlands
{jansenj, hengstum, nymeyer}@ctit.utwente.nl
http://wwwsnmp.cs.utwente.nl/~{jansenj, hengstum, nymeyer}

Abstract
Appie (A): Bla bla.

Keywords: JA, A and E: SNMP, MIB, bla bla ....

1 Introduction

..
2 Functional structure of Management Information Base module

The MIB module provides different services to the surrounding 'outside world'. Via the MIB interface, a MIB prototyper has the opportunity to develop a new MIB specification and to prototype the belonging system interface. Additionally, a management application can issue requests to the MIB module for network management operations through the same MIB interface. On the contrary, the system interface provides the means for the managed system to interact with the MIB module. The managed system can, for instance, take initiative to start a specific operation in the MIB module in case a panic situation occurs in the managed system.

To support all these services?/functionalities?/tasks?, the MIB module is further structured, see figure 2, into four cooperating modules. These modules interact with each other via well-defined internal interfaces and communicate with the surrounding environment?/outside world? via well-defined external interfaces. The so-called MIB interface and the system interface are examples of external interfaces. The MIB scheduler is primary responsibility for the flow control and the data control inside the MIB module.

![Figure 2: Functional structure of MIB module](image)

The four cooperating modules have each own primary functionality. The operation module controls the processing of an requested operation. Moreover, operation initiated on behalf of an management application. The database module contains the actual MIB filled with management information describing the status of the managed system. The administration module holds information related to the PE module. The MIB module can obtain network management out of the managed system to populate the actual MIB by means of the system module.

2.1 The database module

The database module contains the internal representation of the specified MIB. A MIB specification is normally specified in the Abstract Syntax Notation 1 (ASN.1) language. Such a specification describes, for instance, the hierarchical naming structure, the typing and the access rights belonging to the nodes of this MIB.

In the database module the MIB is represented in an hierarchical tree datastructure. All the nodes in this tree are uniquely identified by a so-called object identifier. The object identifier of an leaf node combined with the value of the leaf node is called a variable binding.

The leaf nodes of such a tree contain the actual network management information describing the status of the managed system. Two types of leaf nodes are possible. The scalar type leaf node contains just a single, atomic value. The table type leaf node contains values structured in rows and columns. The value of a leaf node can be static or dynamic. A leaf node with a static value remains continuously unchanged and is permanently stored in that leaf. The value of leaf node containing a dynamic value can change over time. Obviously, this changing value reflects the corresponding change in the status of the managed system. This dynamic value is not stored in the leaf.
node, but rather the reference to this information in the managed system is stored in the leaf node. Or more concrete, the name of the script that can obtain the required dynamic information is stored in the leaf node. Other modules can access this module via the internal database interface. The management information contained in this tree is stored in a file through the external database interface. The actual structure and information of the tree is as well stored in an ASN.1 file through the external database interface.

2.2 The operations module

The operations module performs and controls the processing of requested operations. The operations that can be processed are the MibRead operation, the MibWrite operation and the MibTrap operation (see table 1). All these operations operate on zero, one or more variable bindings. Usually, these variable bindings are contained in a list, the so-called variable binding list. Such a variable binding list contains the references to the request variables and if appropriate contain the corresponding values, hence the name variable binding. The MibRead operation makes it possible to obtain network management information from the MIB module. The other way around, the MibWrite operation, i.e. the set operation in the simple network management protocol, provides the means to modify network management information. This modification changes immediately the behaviour of the managed system. Both types of operations are always initiated on behalf of a supervising management application.

The MibRead operation can operate in three different modes; the get mode, the getnext mode and the getbulk mode. The appropriate mode is selected by different values of the NonRepeaters and MaxRepetitions parameters. In the get mode, all corresponding values of the variables in the variable binding list are obtained. In the getnext mode, per variable in the variable binding list the lexicographical following leaf node with its value is obtained. The getbulk mode is a bit more complicated. The first number of NonRepeaters variables in the variable binding list is each processed in the getnext mode manner. Each variable of the remaining variables in the variable binding list is processed MaxRepetitions times the getnext mode way.

<table>
<thead>
<tr>
<th>Module interface function</th>
<th>Function parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MibRead</td>
<td>VariableBindingList (IN/OUT), NonRepeaters (IN/OUT), MaxRepetitions (IN/OUT)</td>
<td>processes some form of GET request</td>
</tr>
<tr>
<td>MibWrite</td>
<td>VariableBindingList (IN/OUT)</td>
<td>processes a SET request</td>
</tr>
<tr>
<td>MibTrap</td>
<td>VariableBindingList (OUT)</td>
<td>processes a pending TRAP situation</td>
</tr>
</tbody>
</table>

Table 1: The routines of the MIB interface relevant for a management application

The last type of operation, the MibTrap, i.e. the trap operation in the simple network management protocol, will be initiated on behalf of the monitoring agent. Whenever an panic situation arrises in the managed system, which requires immediate a problem solving action of the supervising management application, the monitoring agent will initiate a trap operation. During the processing of an operation, this module buffers intermediate results. This buffering supports two purposes. Usually a management application requests an operation related to several variable bindings. The actual processing of the requested operation in all the modules takes place on a per variable binding basis. All intermediate results must be collected and the composed overall result is returned to the requester. Another purpose of the buffering is related to the atomicity requirement. An operation is released whenever the processing was successful, or in case an failure happened, the rollback of the operation must be performed. This restoring of the original status before the operation started implicates the storing of previous information.

The MIB interface (table 1) provides to the UI module and PE module the possibility to access the MIB module. Via the internal operations interface interact the operations module with all the other MIB modules.

2.3 The administration module

This module contains administration belonging to the PE module. This administration is necessary for different purposes. The first application of this information is, for instance, for the checking on access rights of an intended operation, initiated by a management application, before the actual processing of the operation starts. Another
The purpose of this module is the storing of the values of variables counting the different error situations that occurred in the PE module during the processing of a protocol data unit.

Other modules can access this module via the internal administration interface. The information in this module can be loaded and stored into files through the external administration interface.

### 2.4 The system module

The system module, see figure 3, provides the means to physically access the managed system to obtain management information out of the system, to modify a piece of the managed system or to check for panic situations in the managed system.

![Diagram](image)

**Figure 3: Functional structure of system module**

The internal system interface, see table 2, provides this functionality to the other modules. The `SystemRead` function obtains information out of the managed system on a per variable binding basis. This functionality is very useful for the processing of the `MIBRead` and `MibTrap` functions. The remainder of the functions, `SystemWrite`, `SystemWriteCommit` and `SystemWriteRollback`, of the internal system interface form combined the functionality of the `MibWrite` function. These functions make it actually possible to modify the behaviour of the managed system. One function which implements the `MibWrite` function is not sufficient to guarantee the required atomicity of the `MibWrite` function.

<table>
<thead>
<tr>
<th>Module interface function</th>
<th>Function parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemRead</td>
<td>ScriptName (IN), VariableBinding (IN/OUT)</td>
<td>process a partly GET/Trap operation</td>
</tr>
<tr>
<td>SystemWrite</td>
<td>ScriptName(IN), VariableBinding (IN)</td>
<td>process a party SET operation</td>
</tr>
<tr>
<td>SystemWriteCommit</td>
<td>-</td>
<td>process an entire SET operation</td>
</tr>
<tr>
<td>SystemWriteRollback</td>
<td>-</td>
<td>undo an entire SET operation</td>
</tr>
</tbody>
</table>

**Table 2: The routines of the internal system interface**

The module is further separated in four modules. The `executor` module actually executes the loaded scripts to perform the required functionality. First, the necessary language interpreter is invoked. The intended script is loaded inside the language interpreter and maybe an additional library with very convenient pre-defined functions is as well loaded inside the language interpreter. The needed command line arguments, `function_type` and `value`, are passed to the loaded script. The execution of the scripts happens and the result is returned to scheduler.

The `memory` module contains the scripts which actually provide the real system interface to the managed system. These scripts can be loaded and saved via the external interface to an external device.

The `storage` module stores/holds all the original values, which are grabbed out of the managed system and are intended to be restored in case an rollback situation will be happening. After a `SystemWriteRollback` or
SystemWriteCommit has happened the storage module is cleaned up for the storing of new values. The stored values can be loaded or saved from/to an external device via the external interface. (Note: This module can be used to implement the different time domains, such as CurrentTime, RestartTime and CacheTime.)

The operations module actually controls the processing of the diverse functions. The MibTrap/MibRead are mapped onto the SystemRead and thus are processed identically inside this module. The operations module orders to load the appropriate SystemRead script in the executor. The script is executed and the required information is obtained from the managed system and returned/passed to the outside requestor.

The SystemWrite is processed a slightly different from the SystemRead. The appropriate SystemRead script is loaded. The script is executed to obtain the original system value out of the managed system, in order to restore a original situation in case an calamity during the processing of the MibWrite will happen. The obtained value is passed to the storage module and there /stored stacked for maybe future use. The appropriate SystemWrite script is loaded and with the belonging new value is executed and afterwards the initiative is passed back to the outside requestor.

In the case that the SystemWriteCommit is invoked/happens only the storage module is cleaned up. Obviously, the real execution of all the previous SystemWrite calls have already happened, so no additional functionality is needed. The SystemWriteRollback use all the stored/stacked original values in the storage module to restore the original status inside the managed system by means of the SystemWrite functionality, but now parametrized by original values. Afterwards, the storage module is cleanup and the initiative is returned to the outside requestor.

3 Realization of MIB module

3.1 Scalar processing

3.2 Table processing

bla ...

4 Conclusions and future work

Some questions which might be usefull to contribute to the formulation of conclusions and future directions of this research:

1) How about all the different time domains. How will they be processed? Suggestion is the Storage module.
2) Do we support one script language or more? How many more? What are common characteristics?
3) Do we enforce a certain script layout? Yes, why? No, why not? Is it necessary?
4) Do we define a library with predefined functions/values?
5) Is one script/functionality per variable needed? One script per set or get/variable needed? Is one script per set of variables efficient?
8) Do we generate scripts? If yes, how do we make them language independent?
9) Were will scripts be stored? Internal memory? File system?
10) What kind of mechanisms/tricks can be used to speed up or to increase efficiency?
11) How will the input/output between the script and MIB layer take place? Command line arguments?
    Shell environment variables? Other alternatives?
12) ...

Acknowledgements

We would like to thank our partners in the Damocles project. They provided us with an inspiring environment for fruitful discussions on the topics discussed in this article.

References


[bla...] ...